EMBEDDED ANDROID DEVELOPMENT

สู่เส้นทางนักพัฒนา



วิรุฬห์ ศรีบริรักษ์

วิศวกรรมระบบสมองกลฝังตัว
ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ มหาวิทยาลัยบูรพา

สนับสนุนโดย สำนักงานพัฒนาวิทยาศาสตร์และเทคโนโลยีแห่งชาติ
สำนักงานคณะกรรมการส่งเสริมการลงทุน และ สมาคมสมองกลฝังตัวไทย

EMBEDDED ANDROID DEVELOPMENT

จัดทำโดย ผศ. วิรุฬห์ ศรีบริรักษ์

ติดต่อ :

169 ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ มหาวิทยาลัยบูรพา

ถนนลงหาดบางแสน ตำบลแสนสุข อำเภอเมืองชลบุรี จังหวัดชลบุรี 20131

โทรศัพท์ 0-3810-2222 ต่อ 3380-82 ต่อ 203 โทรสาร 0-3874-5806 อีเมล์ wiroon@eng.buu.ac.th

ข้อมูลทางบรรณานุกรมของสำนักหอสมุดแห่งชาติ :
วิรุฬห์ ศรีบริรักษ์.
Embedded Android Development.-- : วิรุฬห์ ศรีบริรักษ์, 2557.
397 หน้า.
1. ระบบสมองกลฝังตัว 2. ระบบปฏิบัติการลีนุกซ์ 3. แอนดรอยด์
I. ชื่อเรื่อง.

ISBN (e-book) : 978-616-361-117-8

จัดจำหน่ายโดย :

บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน)
SE-EDUCATION PUBLIC COMPANY LIMITED

อาคารทีซีไอเอฟ ทาวเวอร์ ชั้น 19 เลขที่ 1858/87-90 ถนนบางนา-ตราด แขวงบางนา
เขตบางนา กรุงเทพฯ 10260 โทรศัพท์ 0-2739-8000

หนังสือ eBook เล่มนี้อนุญาตให้ใช้ได้ตามสัญญาอนุญาตครีเอทีฟคอมมอนส์
แสดงที่มา-ไม่ดัดแปลง-ไม่ใช้เพื่อการค้า (เผยแพร่ฟรี) 3.0 ประเทศไทย

คำนิยม

หนังสือเกี่ยวกับความรู้ทางด้านระบบสมองกลฝังตัว (Embedded Systems) ที่เราพบโดยทั่วไปมักจะเป็นหนังสือเกี่ยวกับไมโครคอนโทรเลอร์ ที่เน้นให้เข้าใจฮาร์ดแวร์และการเขียนโปรแกรมแบบง่ายๆ ซึ่งปัญหาหนึ่งที่ตามมาคือผู้สนใจหรือผู้ที่อยากจะเป็นนักพัฒนาไม่สามารถจะทำระบบที่ซับซ้อนได้ (เพื่อให้ใช้งานได้จริง) หรือการทำ User Interface ก็ทำได้ยากมากต้องใช้เวลามากเป็นต้น

ปัจจุบันแนวทางการพัฒนาระบบสมองกลฝังตัวเริ่มเปลี่ยนเป็นทำงานต่อยอดจากแพลตฟอร์มเพื่อช่วยลดเวลาการทำฮาร์ดแวร์ลง เช่น ไม่ต้องซื้อชิ้นส่วนมาประกอบเองแต่ซื้อบอร์ดมาใช้เลย นอกจากนั้นการพัฒนาโปรแกรมก็ไม่จำเป็นต้องเริ่มใหม่ทั้งหมด แต่เริ่มจากระบบปฏิบัติการขนาดเล็กและโปรแกรมฟังชั่นสนับสนุนต่างๆ ทำให้การพัฒนาในลักษณะนี้สามารถเน้นความคิดสร้างสรรค์ใหม่ๆโดยมีการทำงานที่ซับซ้อนได้อย่างรวดเร็ว นอกจากนั้น อุปกรณ์เช่น Smart Phone หรือ Tablet มีการกระจายตัวอย่างรวดเร็ว มีขีดความสามารถสูงรวมถึงมีเซนเซอร์และความสามารถสื่อสารต่างๆครบถ้วน จึงส่งเสริมให้เกิดการเขียนโปรแกรมประยุกต์ที่ดีๆอย่างมากมาย หรือถูกนำเป็นอุปกรณ์ร่วมกับระบบฮาร์ดแวร์อื่นๆได้อย่างดี

Smart Phone หรือ Tablet ที่สัดส่วนการครอบครองตลาดมากที่สุดคือระบบปฏิบัติการแอนดรอยด์ ซึ่งมีสัดส่วนประมาณ 56% ตามด้วยระบบปฏิบัติการ iOS ซึ่งมีสัดส่วนประมาณ 22% ดังนั้นการเข้าใจและสามารถพัฒนาบนระบบแอนดรอยด์ให้เก่งและชำนาญเป็นเรื่องที่จำเป็นมาก

หนังสือ “**Embedded Android Development สู่เส้นทางนักพัฒนา**" ที่เขียนโดย อ.วิรุฬห์ ศรีบริรักษ์ นี้นับว่ามีความโดดเด่นมากทั้งในแง่วิชาการ ประวัติศาสตร์ และการนำไปปฏิบัติจริง สำหรับระบบปฏิบัติการแอนดรอยด์มีรากเง้าการพัฒนาจากระบบปฏิบัติการลีนุกซ์และมาจากระบบปฏิบัติการยูนิกซ์ หนังสือเล่มนี้ก็บรรยายที่มาที่ไปให้ผู้สนใจโดยเฉพาะนักพัฒนาได้ซาบซึ้ง กับประวัติความเป็นมาละเอียดช่วยกระตุ้นความอยากเป็นนักพัฒนาระบบสมองกลฝังตัวได้อย่างดี

หนังสือเล่มนี้มุ่งสร้างนักพัฒนารุ่นใหม่ให้เริ่มเดินด้วยตนเองได้หรือเพื่อเป็นพี่เลี้ยงที่จุดเริ่มเท่านั้นซึ่งจำเป็นมาก นักพัฒนาเปรียบเหมือนนกน้อยที่เริ่มหัดบินซึ่งจริงๆแล้ว นกทุกตัวมีวิญญาณที่จะบินเองได้อยู่แล้ว ขอให้มีโอกาสให้เริ่มได้บิน

ผมเห็นว่าหนังสือเล่มนี้จะมอบโอกาสให้ผู้ที่สนใจ เป็นนักพัฒนาระบบสมองกลฝังตัว แอนดรอยด์สามารถเริ่มบินในโลกแอนดรอยด์ได้อย่างมีอิสระและสร้างสรรค์สิ่งใหม่ๆได้ครับ

ดร. พันธ์ศักดิ์ ศิริรัชตพงษ์

ผู้อำนวยการศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC)

FOREWORD

I am honoured to be writing the forward for Ajarn Wiroon Sriborrirux's new book. I have admired Ajarn Wiroon for a long time as one of the leaders and mentors in the embedded systems community in Thailand. He is well known and widely respected in terms of his generousity in sharing his deep technical knowledge, as a teacher, a professional consultant, an advisor to startup companies and as a colleague. It is very fitting for him to be writing this book as a way to continue to share his knowledge with our community in order to keep advancing Thailand forward in the technology field.

The topic of this book, "Embedded Android Development", is incredibly timely, given the newest developments in terms of the Internet of Things (IOT), cyberphysical trend and the move to a "post-PC era", or what we can call the "embedded systems era" or "device era".

Stage 1 of the "embedded systems era" (2000-2015) was about connectivity. We saw the rise of Ubiquitous Computing or Universal Communicator led by mobile phone and smart phones. By putting a very powerful personal computing and communication device into the hands of a major part of the world's population, this stage built up an important infrastructure for the IOT.

From 2010-2020, we are seeing Stage 2 of this era, with IOT being the keyword of this decade. We will see a rising need to make things "smart". The challenge is not just in the technology, but rather in creating innovation or value creation that utilises information or knowledge gathered from connected devices. CISCO predicted that the value created by the use of IOT is expected to reach $14 trillion by 2020 which is only a few years from now. This means tremendous opportunity for the development community.

Within this context, Embedded Android has become very popular in all types of devices requiring user interface such as measurement instruments, kiosk, access control, payment, health monitor, wearable devices, and cars. We are now currently only at the beginning of IOT innovation race, and I believe that we'll be seeing a lot of exciting innovations utilising Embedded Android in the next few years.

I am confident that this book should be very useful for key courses such as Unix Systems Programming or Embedded Systems Development. This book should also serve as a very good reference for students working on senior project related to advance embedded systems and device development. These days, students learn a lot of theory but have little practical skills, which makes it difficult for them to meet workforce requirements. It is refreshing to see a book with a practical knowledge focus that enhances development skills. It fills a key gap in this field, and particularly in Thailand.

Assistant Professor Apinetr Unakul

President of Thai Embedded Systems Association (TESA)

คำนำ

หนังสือ Embedded Android Development เล่มนี้ได้รับการสนับสนุนเงินทุนจากศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC) สำนักงานพัฒนาวิทยาศาสตร์และเทคโนโลยีแห่งชาติ (สวทช.) สำนักงานคณะกรรมการส่งเสริมการลงทุน (BOI) และสมาคมสมองกลฝังตัวไทย (TESA) เพื่อนำไปใช้ประโยชน์ภายใต้โครงการสร้างเสริมศักยภาพวิชาชีพ Embedded System เพื่อส่งเสริมการเรียนการสอนทั้งทักษะด้านการเขียนโปรแกรมและหลักการเข้าใจการทำงานฮาร์ดแวร์ทางด้านเทคโนโลยีสมองกลฝังตัวให้น่าสนใจและสามารถดึงดูดนักศึกษาให้ศึกษาต่อในสาขา Embedded System รวมทั้งนำไปเผยแพร่ยังมหาวิทยาลัยทั่วประเทศเพื่อใช้ประกอบในการเรียนการสอนหรือการฝึกอบรมทางด้านระบบสมองกลฝังตัว

หนังสือเล่มนี้ผู้เขียนตั้งใจเขียนขึ้นมาจากใจที่มุ่งมั่นและศรัทธาเพื่อต้องการส่งเสริมและกระตุ้นให้เกิดนักพัฒนารุ่นใหม่ภายในเมืองไทย โดยผู้เขียนตั้งใจนำประสบการณ์ที่สั่งสมอยู่ในวงการเทคโนโลยีคอมพิวเตอร์ ระบบปฏิบัติการลีนุกซ์ ระบบสมองกลฝังตัว และระบบปฏิบัติการแอนดรอยด์ ตลอดระยะเวลาไม่น้อยกว่า 18 ปี ซึ่งได้รวบรวมพื้นฐานความรู้สำคัญที่หาอ่านได้ยากแล้วนำมาเรียบเรียงให้เหมาะสมกับนิสิตที่เรียนในวิชาที่เกี่ยวข้องกับระบบสมองกลฝังตัว ตัวอย่างเช่น วิชาระบบปฏิบัติการยูนิกซ์/ลีนุกซ์ วิชาการเขียนโปรแกรม วิชาระบบสื่อสารและเครือข่าย วิชาระบบปฏิบัติการคอมพิวเตอร์ และวิชาระบบสมองกลฝังตัว เป็นต้น ซึ่งจะเป็นพื้นฐานสำคัญในการออกแบบและพัฒนาทางด้านระบบสมองกลฝังตัว โดยเฉพาะสำหรับนิสิตนักศึกษาทางด้านวิศวกรรมคอมพิวเตอร์ วิศวกรรมไฟฟ้าอิเล็กทรอนิกส์ วิศวกรรมไฟฟ้าสื่อสาร วิศวกรรมไฟฟ้าควบคุม วิทยาการคอมพิวเตอร์ และสาขาที่ใกล้เคียง

นอกจากนั้นยังเหมาะสำหรับนักพัฒนาและผู้ที่สนใจการทำงานภายในระบบสมองกลฝังตัว ซึ่งเนื้อหาในหนังสือเล่มนี้จะรวบรวมศาสตร์ความรู้ที่เกี่ยวข้องในการออกแบบและพัฒนาทางด้านระบบสมองกลฝังตัวเพื่อนำไปใช้ในการปรับปรุงหรือสร้างสรรค์ผลิตภัณฑ์ใหม่ในอนาคตได้ วัตถุประสงค์หลักของหนังสือเล่มนี้ต้องการปูพื้นความรู้ตั้งแต่การใช้งานระบบปฏิบัติการลีนุกซ์เพื่อให้คุ้นเคยกับคำสั่งที่เกี่ยวข้องตั้งแต่ระดับพื้นฐานจนถึงระดับกลางให้ผู้อ่านเข้าใจระบบลีนุกซ์เคอร์เนล การพัฒนาโปรแกรมระดับล่าง การเตรียมสภาพแวดล้อมให้พร้อมสำหรับการปรับแต่งระบบปฏิบัติการลีนุกซ์ เครื่องมือที่เกี่ยวข้องทั้งหมดสำหรับการพัฒนาโปรแกรมบนระบบสมองกลฝังตัว ทำความเข้าใจโครงสร้างระบบปฏิบัติการแอนดรอยด์ การพัฒนาโปรแกรมด้วยโปรแกรมภาษา C/C++/JAVA/Python และการประยุกต์บนบอร์ดสมองกลฝังตัว โดยในบทสุดท้ายเป็นการยกตัวอย่างกรณีศึกษาและแนวคิดสำคัญในการออกแบบและพัฒนาเพื่อให้ผู้อ่านสามารถเข้าใจและสามารถเชื่อมโยงความรู้ที่ได้เพื่อนำไปสู่การเริ่มต้นการออกแบบและพัฒนาทางด้านระบบสมองกลฝังตัวในอนาคต

กิตติกรรมประกาศ

หนังสือ Embedded Android Development เล่มนี้จะสมบูรณ์ไม่ได้ถ้าขาดการช่วยเตรียมข้อมูลประกอบสำหรับแต่ละบทจากลูกศิษย์ที่รักและนักวิจัยประจำห้องปฏิบัติการวิจัยและพัฒนาด้านสมองกลฝังตัวของข้าพเจ้า ซึ่งได้แก่ นายสรไกร ไกรปุย นายธิติพงษ์ สอนจันทร์ นายภานุวัฒน์ พรหมศิริ นายวิธ-วัช บุญแย้ม นายวีระเดช ขุมทอง นายวสันต์ วิยะรันดร์ นายปริวัฒน์ เลี่ยมสำราญ นายนิพนธ์ สมหมาย นายจเร ปิยฉัตรพนม นายนัฐพงษ์ แสงสง่าศรี และนายกิตติศักดิ์ พรมศรี รวมทั้งเพื่อนร่วมงานที่คอยเป็นกำลังใจ ซึ่งได้แก่ ดร.ภาณุวัฒน์ ด่านกลาง รศ.ดร.ณกร อินทร์พยุง และ ดร.อภิรัฐ ลิ่มมณี

ขอขอบคุณภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ มหาวิทยาลัยบูรพา ที่สนับสนุนให้มีการเปิดสาขาวิชาทางด้านวิศวกรรมอิเล็กทรอนิกส์และระบบสมองกลฝังตัว และหลักสูตร Embedded Systems Engineering (English Program) ซึ่งทำให้ข้าพเจ้าได้มีโอกาสสอนและทำการวิจัยและพัฒนาองค์ความรู้ทางด้านวิศวกรรมระบบสมองกลฝังตัว รวมทั้งขอขอบคุณนายกสมาคมสมองกลฝังตัวไทย ผู้ช่วยศาสตราจารย์อภิเนตร อูนากูล และ ผู้จัดการสมาคมฯ พี่ขนิษฐา ประสารสุข ที่ได้ให้ข้าพเจ้าได้มีโอกาสเข้ามาร่วมทำงานและสร้างกิจกรรมดีๆตั้งแต่ปี พ.ศ. 2548 เป็นต้นมา

ที่สำคัญด้วยกำลังใจที่เปี่ยมล้นจากครอบครัวที่รักของข้าพเจ้าเอง (นางณัฐณิชา หนูคง และ ด.ญ. ปวรนันทน์ ศรีบริรักษ์) ที่คอยเป็นกำลังใจตลอดระยะเวลาหลายเดือนของการเขียนหนังสือเล่มนี้ ขอบพระคุณบิดา (นายสามารถ ศรีบริรักษ์) มารดา (นางแดง ศรีบริรักษ์) ที่ให้ชีวิตที่มีค่านี้แก่ข้าพเจ้าได้มีโอกาสสร้างสรรผลงาน และสร้างคนสำหรับวงการระบบสมองกลฝังตัว เพื่อสักวันหนึ่งเมืองไทยจะได้เห็นความสำคัญที่แท้จริงของเทคโนโลยีเหล่านี้

สารบัญ

บทที่ 1 พื้นฐาน Unix/Linux สำหรับนักพัฒนา **13**

ประวัติระบบปฏิบัติการ Unix/Linux 13

ประเภทของ *Licenses* *18*

ปรัญชา และความรู้พื้นฐานของระบบปฏิบัติการลีนุกซ์ 19

กระบวนการทำงานของเชลล์ และชุดคำสั่งที่เกี่ยวข้อง *20*

ตัวแปรสภาพแวดล้อมของระบบ *28*

การเรียกใช้งานคำสั่งภายในเชลล์ และภายนอกเชลล์ *32*

คำสั่งพื้นฐานสำหรับนักพัฒนาด้านบนระบบสมองกลฝังตัว 33

คำสั่งตรวจสอบทรัพยากรระบบ *33*

คำสั่งตรวจสอบการใช้หน่วยความจำระบบ *36*

คำสั่งตรวจสอบการใช้พื้นที่สำหรับเก็บข้อมูล *37*

คำสั่งสำหรับการบริหารจัดการโปรเซส *38*

การอ่านสถานะของทรัพยากรระบบจากไดเรกทอรี */proc* *43*

คำสั่งเกี่ยวกับการเปิดอ่านข้อมูลภายในไฟล์ *45*

คำสั่งค้นหาข้อความและไฟล์ด้วยชุด *Regular Expressions* *48*

คำสั่งจัดการด้านระบบเครือข่าย *60*

บทที่ 2 พื้นฐานลีนุกซ์คอร์เนลสำหรับนักพัฒนา **67**

Linux Kernel 68

Linux Versioning *69*

โครงสร้างไดเรกทอรี และขนาดพื้นที่ของ *Linux Kernel 3.2* *70*

พื้นฐานการปรับแต่งและสร้าง Custom Kernel 73

5 ขั้นตอนพื้นฐานการคอมไพล์ *Linux Kernel* *74*

คอมไพล์ *Linux Kernel 3.x* สำหรับ *Ubuntu* ที่ใช้อยู่ *81*

การพัฒนา Linux Kernel Module 84

พื้นฐานการเขียน *Linux Module* *88*

พื้นฐานการเขียนโปรแกรมไดร์เวอร์สำหรับ *character device* *92*

การเพิ่ม *Linux Module* ใหม่เข้าไปยัง *Linux Source Tree* *97*

บทที่ 3 Embedded Linux Development **100**

ความเป็นมาของระบบสมองกลฝังตัว **100**

สถาปัตยกรรมในระบบสมองกลฝังตัว 104

สถาปัตยกรรมไมโครโปรเซสเซอร์และอุปกรณ์ฮาร์ดแวร์สำคัญสำหรับระบบสมองกลฝังตัว *104*

ก่อนจะเป็นบอร์ดสมองกลฝังตัว *106*

เริ่มต้นสู่การพัฒนาบนระบบปฏิบัติการ Embedded Linux 109

องค์ประกอบการเตรียมสภาพแวดล้อมสำหรับ *Embedded Linux* *109*

การเชื่อมต่อระหว่างเครื่อง *Host* และ บอร์ด *Target* *112*

เครื่องมือ Cross Toolchains 116

ประเภทของ *Cross Toolchains* *117*

องค์ประกอบหลักภายใน *Cross Toolchains* *118*

ขั้นตอนการเตรียมระบบสำหรับพัฒนาบน Embedded Linux 120

การเตรียมสภาพแวดล้อมให้กับเครื่อง *Host* *121*

Toolchain Options ที่สำคัญ *123*

Bootloaders 124

รายละเอียดภายใน *Kernel Image* ที่ใช้ในบอร์ดสมองกลฝังตัว *126*

Linux File Systems 127

Virtual Filesystems *127*

การป้องกันข้อมูลภายในระบบไฟล์ *134*

Embedded Linux File System 137

ระบบไฟล์ในระบบสมองกลฝังตัว *139*

Memory Technology Devices (MTD) *140*

ขั้นตอนการเข้าโหลดระบบไฟล์เพื่อเข้าสู่ระบบฏิบัติการ *Embedded Linux* *142*

BusyBox มีดพกพาสารพัดประโยชน์ 147

การพัฒนาระบบสมองกลฝังตัวภายใต้ระบบจำลองเสมือนจริง 152

ขั้นตอนการทดสอบการรันโปรแกรมภาษา C บน QEMU 155

ขั้นตอนการทดสอบ BusyBox ภายใน Root Filesystem บน QEMU 157

การสร้างระบบจำลองสถาปัตยกรรม ARM ด้วยชุดเครื่องมือ Buildroot บน QEMU 161

ตัวอย่างการสร้างระบบจำลองเสมือนของบอร์ด Raspberry Pi บน QEMU 168

ตัวอย่างการสร้างระบบจำลองเสมือนของบอร์ด Friendly ARM บน QEMU 171

บทที่ 4 พื้นฐานการเขียนโปรแกรมภาษา C/C++ และ Qt สำหรับนักพัฒนา 178

พื้นฐานเขียนโปรแกรมภาษา C/C++ สำหรับการพัฒนาระบบสมองกลฝังตัว 178

พื้นฐานการสร้าง *Makefile* *179*

การสร้างและอ้างอิงไลบรารี *181*

การพัฒนาโปรแกรมเพื่อเข้าถึงระบบไฟล์ *184*

การพัฒนาโปรแกรมติดต่อพอร์ตอนุกรม *185*

การพัฒนาโปรแกรมสื่อสารระหว่างโปรเซส *198*

การพัฒนาโปรแกรมสื่อสารบนระบบเครือข่าย *210*

การพัฒนาโปรแกรมเก็บข้อมูลด้วย *SQLite* *218*

การดีบักโปรแกรมภาษา *C/C++* *220*

การเขียนโปรแกรมภาษา C++ ด้วย Qt 227

การติดตั้ง *Qt* สำหรับแต่ละระบบปฏิบัติการ *227*

กลไกการทำงานของ *Signal* และ *Slot* *232*

การพัฒนาโปรแกรมติดต่อพอร์ตอนุกรม *236*

การพัฒนาโปรแกรมแบบ *Multi-threading* *240*

บทที่ 5 พื้นฐานระบบปฏิบัติการแอนดรอยด์สำหรับนักพัฒนา 244

ระบบปฏิบัติการแอนดรอยด์ 244

สถาปัตยกรรมของระบบปฏิบัติการแอนดรอยด์ *247*

แนวทางการพัฒนา Embedded Android 250

เตรียมสภาพแวดล้อมสำหรับการพัฒนา Embedded Android 252

เตรียมสภาพแวดล้อมบนเครื่อง *Host* *252*

Android Open Source Project (AOSP) 254

ดาวน์โหลดซอร์ส *AOSP* *254*

โครงสร้างภายใน *AOSP* *256*

Android Kernel 258

ขั้นตอนการคอมไพล์ *AOSP* มาเป็นระบบปฏิบัติการแอนดรอยด์ *259*

ระบบปฏิบัติการแอนดรอยด์บน Android Emulator 265

พื้นฐานการใช้ *Android Debug Bridge (ADB)* *267*

ขั้นตอนการปรับแต่ง Android Kernel สำหรับ Emulator 271

ขั้นตอนการติดตั้ง *Android Kernel* สำหรับ *Android Emulator (Goldfish)* *271*

การพัฒนา Kernel Module สำหรับระบบปฏิบัติการแอนดรอยด์ 276

การสร้างโปรแกรมประยุกต์เพื่อฝังลงระบบปฏิบัติการแอนดรอยด์ 280

ชุดเครื่องมือและคำสั่งภายใน Android Emulator 288

บทที่ 6 พื้นฐานการเขียนโปรแกรมภาษาจาวาบนแอนดรอยด์สำหรับนักพัฒนา302

เครื่องมือพัฒนา Android Studio IDE 303

วิธีการติดตั้ง *Android Studio IDE* *304*

การติดตั้งและเรียกใช้โปรแกรมบนอุปกรณ์แอนดรอยด์ *309*

การย้ายโค้ดโปรแกรมเดิม *Eclipse IDE* มาสู่ *Android Studio* *311*

Apache Ant สำหรับการนักพัฒนาแอนดรอยด์ *312*

Android Activity *313*

User Interface *320*

Android Adapter *329*

Android Intent *332*

Broadcast Receiver *333*

การพัฒนาโปรแกรมด้วย Android Native Development Kit 337

พื้นฐาน *Android NDK* *337*

เริ่มต้นการพัฒนาโปรแกรม *Android NDK* *340*

ตัวอย่างโปรแกรม *Hello World* ด้วย *Android NDK* *342*

พื้นฐานการพัฒนา *JAVA Native Interface (JNI)* *348*

ตัวอย่างการสร้างและเรียกใช้ *JNI Methods* *352*

ตัวอย่างการพัฒนา *Android NDK Multi-Threading* *359*

บทที่ 7 การพัฒนาโปรแกรมประยุกต์บนระบบสมองกลฝังตัว **366**

ตัวอย่างการพัฒนาโปรแกรมบนบอร์ด Raspberry Pi 366

เครื่องมือพัฒนาพื้นฐานสำหรับ Android และ Arduino 382

ตัวอย่างการเชื่อมต่อระหว่าง Android กับ Arduino ผ่าน ADK 386

ตัวอย่างการพัฒนาการแสดงสัญญาณไฟฟ้ากล้ามเนื้อ (EMG) 389

บทสรุปและก้าวต่อไป... **395**

ประวัติผู้เขียน **396**

บทที่ 1 พื้นฐาน UNIX/LINUX สำหรับนักพัฒนา

ประวัติระบบปฏิบัติการ Unix/Linux

เป็นระยะเวลาไม่น้อยกว่า 55 ปี เมื่อย้อนกลับไปตั้งแต่ก่อนที่นักพัฒนาจากห้องปฏิบัติการ Bell จะพัฒนาระบบปฏิบัติการแบบใหม่ชื่อระบบปฏิบัติการ UNIX โดยใช้ภาษาซี (C Language) เป็นภาษาโปรแกรมหลักในการเขียนระบบปฏิบัติการตัวนี้ขึ้นมา จนกลายเป็นภาษาที่ได้รับความนิยมในหมู่นักเขียนโปรแกรมมากที่สุด จุดเริ่มต้นของเรื่องนี้ทั้งหมดเกิดขึ้นในปี ค.ศ. 1960 ซึ่งได้เริ่มมีการพัฒนาระบบปฏิบัติการแบบแบ่งเวลา (timesharing) โดย Dartmouth College และ Massachusetts Institute of Technology (M.I.T.) ซึ่งมีจุดเด่นคือ ผู้ใช้หลายคนสามารถใช้เครื่องในเวลาเดียวกันได้ โดยอาศัยการแบ่งเวลาของหน่วยประมวลผลกลางให้แก่ผู้ใช้เวียนกันไป ถูกพัฒนาขึ้นโดยภาษาโปรแกรมเบสิค (BASIC Language) แต่ประสบความสำเร็จในการใช้งานทางธุรกิจแค่ช่วงระยะหนึ่ง ในขณะที่ระบบปฏิบัติการอีกตัวชื่อ CTSS (MIT's Compatible Time-Sharing System) จาก M.I.T. ซึ่งเป็นระบบปฏิบัติการที่ได้ถูกออกแบบเพื่อให้เป็นระบบปฏิบัติการอเนกประสงค์ ก็ได้รับความนิยมในกลุ่มนักวิทยาศาสตร์เป็นพิเศษ หลังจากนั้นไม่นาน ทาง M.I.T. ห้องปฏิบัติการ Bell และบริษัท GE (General Electric) ซึ่งเป็นบริษัทผู้ผลิตคอมพิวเตอร์ ได้รวมกลุ่มกันเพื่อทำการวิจัยและออกแบบระบบปฏิบัติการแบบแบ่งเวลาตัวใหม่ ให้มีความสามารถมากขึ้นและกำหนดชื่อระบบปฏิบัติใหม่นี้ชื่อว่า MULTICS (MULTiplexed Information and Computing Service)

แม้ว่าตัวระบบปฏิบัติการ MULTICS จะสามารถรองรับผู้ใช้ได้หลายร้อยคน แต่โครงการก็ยังเกิดปัญหาขึ้นหลายอย่าง โดยเฉพาะอย่างยิ่งตัวภาษา PL/I ที่ใช้ในการเขียนโปรแกรมนั้นยังอยู่ในระหว่างการพัฒนาและมีการพัฒนาล่าช้ากว่ากำหนดการที่กำหนดไว้มากและมีข้อบกพร่องมากมาย รวมทั้งปัจจัยอื่นที่เทคโนโลยีในขณะนั้นเองก็ยังไม่พร้อม ห้องปฏิบัติการ Bell จึงได้ถอนตัวออกจากโครงการหลังจากสิ้นสุดโครงการระยะแรก

หนึ่งในนักวิจัยของโครงการชื่อ Ken Thompson จึงเริ่มหาแนวทางในการทำวิจัยต่อไป โดยยังคงนำระบบปฏิบัติการ MULTICS มาทำการพัฒนาต่อ ซึ่งได้ทำการย่อส่วนโค้ดโปรแกรมโดยใช้ภาษาแอสเซมบลี้ (Assembly Language) ในการพัฒนาระบบปฏิบัติการตัวใหม่นี้บนเครื่องมินิคอมพิวเตอร์รุ่น PDP-7 จนกลายเป็นระบบปฏิบัติการที่สามารถทำงานได้เป็นอย่างดี และในที่สุดก็ได้ถูกตั้งชื่อใหม่โดยหนึ่งในนักวิจัยของห้องปฏิบัติการ Bell ชื่อ Brian Kernighan ว่า UNICS (Uniplexed Information and Computing Service) เพื่อเป็นการล้อเลียนโครงการ MULTICS และต่อมาก็ได้รับการเปลี่ยนชื่อเป็น UNIX ในที่สุด

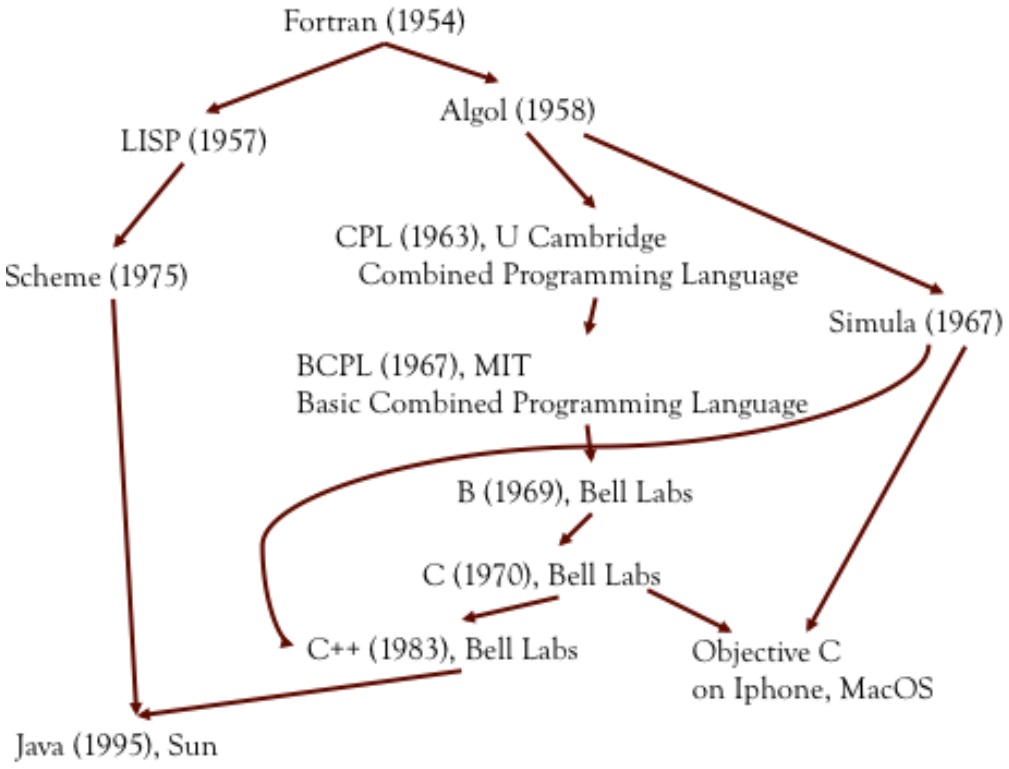
ต่อมาก็ได้มีนักวิจัยคนอื่นๆในห้องปฏิบัติการ Bell เริ่มสนใจที่จะขอเข้าร่วมโครงการเพื่อพัฒนาระบบปฏิบัติการ UNIX ของ Ken Thompson มากขึ้น และคนแรกที่ได้เข้าร่วมก็คือ Dennis Ritchie เพื่อพัฒนาระบบปฏิบัติการ UNIX อยู่บนเครื่องมินิคอมพิวเตอร์รุ่น PDP-11 ซึ่งมีขีดความสามารถสูงกว่า

เครื่องรุ่น PDP-7 เดิม ต่อมานักวิจัยที่เหลือก็ได้เข้าร่วมโครงการทั้งหมด จนทำให้ระบบปฏิบัติการ UNIX ที่อยู่บนเครื่องรุ่น PDP-11/45 และ PDP-11/70 ได้รับความนิยมในตลาดสูงในช่วงทศวรรษที่ 1970 เพราะได้เพิ่มขนาดของหน่วยความจำให้ขนาดใหญ่ และมีกลไกการป้องกันหน่วยความจำ ทำให้เป็นเครื่องคอมพิวเตอร์ที่สามารถรองรับการใช้งานผู้ใช้หลายคนในเวลาเดียวกันได้พร้อมกัน



รูปที่ 1.1 Ken Thompson และ Dennis Ritchie

นอกจากนั้นห้องปฏิบัติการ Bell ยังได้วางแผนพัฒนาภาษาโปรแกรมตัวใหม่ เพื่อใช้ในการเขียนระบบปฏิบัติการเนื่องจากคอมพิวเตอร์แต่ละรุ่นจะมีโครงสร้างทางฮาร์ดแวร์แตกต่างกัน ทำให้ต้องใช้เวลาในการพัฒนาด้วยภาษาโปรแกรมแอสเซมบลี้มากพอสมควร จึงเป็นเหตุผลสำคัญที่จะต้องเร่งสร้างภาษาโปรแกรมตัวใหม่ที่ยืดหยุ่นสำหรับระบบปฏิบัติการในอนาคต โดยในระยะแรกของการพัฒนาภาษาโปรแกรมนั้นทาง Ken Thompson ได้เลือกใช้ภาษาบี (B Language) ซึ่งเป็นภาษาโปรแกรมที่พัฒนาต่อมาจากภาษา BCPL (Basic Conbined Programming Lanaguge) พัฒนาโดย M.I.T. ดังวิวัฒนาการภาษาโปรแกรมดังรูปข้างล่าง



รูปที่ 1.2 วิวัฒนาการภาษาโปรแกรม

ตัวอย่างโปรแกรมแสดงค่า 5 Factorials โดยเปรียบเทียบรูปแบบการเขียนโปรแกรมด้วยภาษา BCPL และภาษาบี

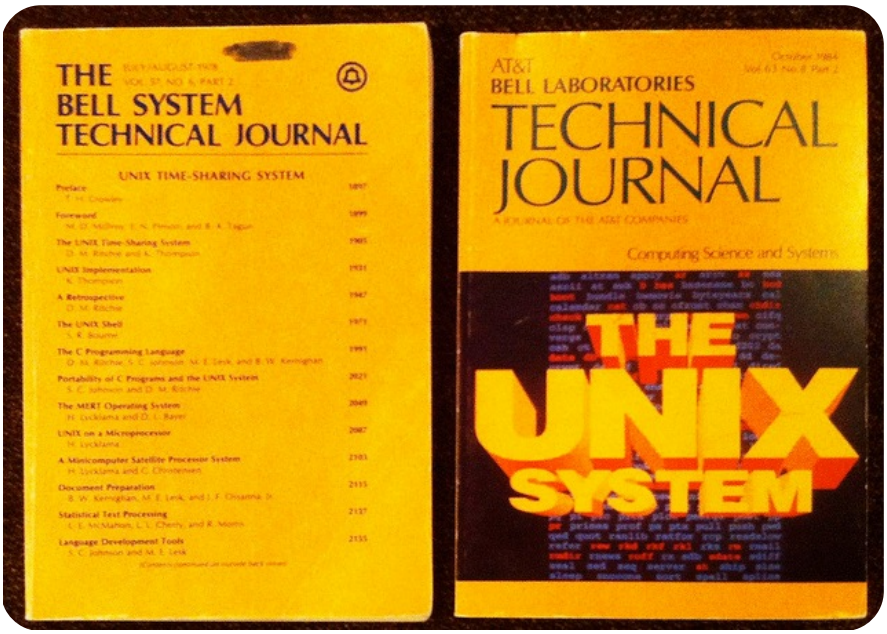
โปรแกรมด้วยภาษา BCPL:

```
GET "LIBHDR"
LET START() = VALOF $(
        FOR I = 1 TO 5 DO
                WRITEF("%N! = %I4*N", I, FACT(I))
        RESULTIS 0
)$
AND FACT(N) = N = 0 -> 1, N * FACT(N - 1)
```

ตัวอย่างโปรแกรมภาษาบีจากตัวอย่างหนังสือที่ Ken Thompson เป็นผู้เขียน

```
/* The following function will print a non-negative number, n, to
  the base b, where 2<=b<=10,  This routine uses the fact that
  in the ASCII character set, the digits 0 to 9 have sequential
  code values.  */
 printn(n,b) {
        extrn putchar;
        auto a;
        if(a=n/b) /* assignment, not test for equality */
                printn(a, b); /* recursive */
        putchar(n%b + '0');
}
```

แต่เนื่องจากตัวภาษาบีเองนั้น เป็นภาษาโปรแกรมที่มีโครงสร้างข้อมูลและรูปแบบการควบคุมภายในยังมีข้อจำกัดอยู่พอสมควร ทำให้การพัฒนาระบบปฏิบัติการ UNIX โดยใช้ภาษาบียังไม่ค่อยประสบความสำเร็จเท่าที่ควร ดังนั้นทาง Dennis Ritchie จึงได้พัฒนาและปรับปรุงภาษาบีให้มีคุณสมบัติที่เหมาะสมในการเขียนระบบปฏิบัติการมากยิ่งขึ้น และในที่สุดก็ได้กลายมาเป็นภาษาโปรแกรมตัวใหม่ชื่อว่าภาษาซี (C Language) ต่อมาทาง Thompson และ Ritchie ได้ร่วมกันพัฒนาระบบปฏิบัติการ UNIX จากภาษาซีใหม่ทั้งหมด ทำให้ภาษาซีกลายมาเป็นภาษาโปรแกรมที่ได้รับความนิยมในหมู่นักเขียนโปรแกรมอย่างมากเนื่องจากเป็นภาษาอเนกประสงค์ที่เหมาะกับการใช้เขียนโปรแกรมแบบต่างๆและหลากหลาย ทั้งยังเป็นภาษาโปรแกรมที่สามารถทำความเข้าใจได้ง่ายและสามารถนำโค้ดโปรแกรมเก่ามาใช้งานประยุกต์กับโปรแกรมใหม่ได้ ต่อมาในปี ค.ศ. 1974 Ritchie และ Thompson ได้ตีพิมพ์ผลงานการวิจัยและพัฒนาระบบปฏิบัติการ UNIX ตัวใหม่นี้ จนเป็นผลให้ทั้งสองได้รับรางวัล ACM Turing Award ในปี ค.ศ. 1984



รูปที่ 1.3 ระบบปฏิบัติการ Unix ในวารสาร The Bell System Technical Journal

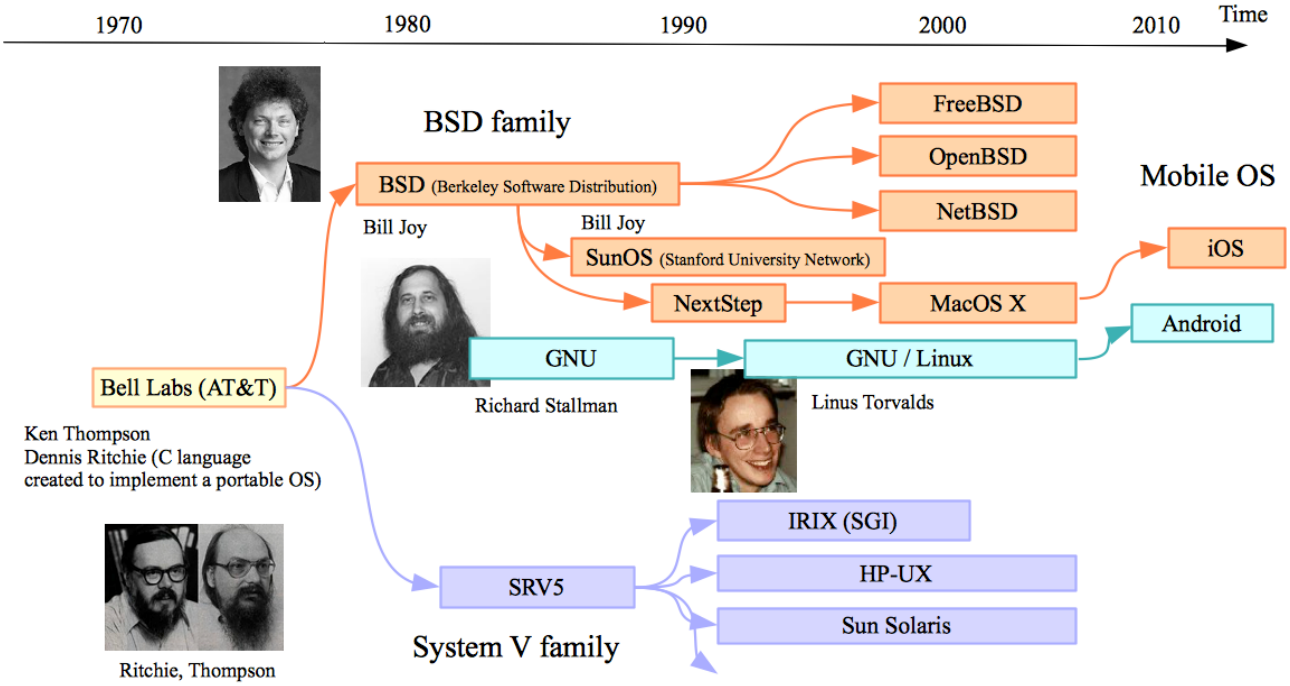
copies of the 1978 and 1984 special Unix issues of the *Bell System Technical Journal (Link)*

จากผลงานดังกล่าวทางบริษัท AT&T ผู้เป็นเจ้าของห้องปฏิบัติการ Bell และเป็นผู้ถือลิขสิทธิ์ระบบปฏิบัติการ UNIX ได้อนุญาตให้มหาวิทยาลัยต่างๆ ใช้ระบบปฏิบัติการ UNIX โดยเสียค่าธรรมเนียมเพียงเล็กน้อย เพื่อหวังให้เป็นที่นิยมมากยิ่งขึ้น ในขณะนั้นด้วยระบบปฏิบัติการจะมากับเครื่อง PDP-11 ซึ่งยังทำงานไม่มีประสิทธิภาพดีพอและยังใช้งานยากอยู่ จึงทำให้เกิดการวิจัยและพัฒนาปรับปรุงระบบปฏิบัติการ UNIX กันอย่างกว้างขวาง จนกระทั่งในที่สุดก็ได้เกิดตัวใหม่ขึ้นที่ชื่อว่า ระบบปฏิบัติการ BSD UNIX (Berkeley Software Distribution UNIX) ซึ่งถูกพัฒนาโดย University of California (UC Berkeley) จนกลายเป็นตัวหนึ่ง ที่ได้รับความนิยมและมีการใช้งานกันอย่างแพร่หลายในสถาบันการศึกษา ต่อมาหน่วยงานกระทรวงกลาโหมของสหรัฐฯ (Defense Advanced Research Projects Agency - DARPA) ก็ได้ให้ทุนกับทาง UC Berkeley เพื่อพัฒนาระบบปฏิบัติการ UNIX ต่อให้กลายเป็น Version 4 BSD เพื่อรองรับการสื่อสารของเครือข่าย DARPA ที่ใช้มาตราฐานในการสื่อสารชื่อว่า TCP/IP ต่อมาในปี ค.ศ. 1993 ทาง UC Berkley ก็ได้ออกตัว BSD รุ่น 4.4 ที่รองรับการสื่อสารแบบโปรโตคอล X.25 แต่เป็นที่น่าเสียดายที่ในที่สุด UC Berkeley ก็ได้หยุดการพัฒนาระบบปฏิบัติการ UNIX ในเวลาต่อมา

หลังจากนั้นไม่นานพัฒนาการของเครื่องคอมพิวเตอร์ก็เริ่มมีประสิทธิภาพสูงขึ้น ในขณะที่ราคาฮาร์ดแวร์ก็เริ่มถูกลง ในที่สุดระบบปฏิบัติการ UNIX ที่เคยอยู่ในระดับมินิคอมพิวเตอร์ก็สามารถนำมาใช้บนคอมพิวเตอร์ส่วนบุคคล (Personal Computer) ที่เรียกว่า “XENIX” แม้ในยุคแรกของเครื่อง XENIX ยังมีเพียงหน้าจอที่แสดงแต่เพียงตัวหนังสือ (Text Mode) จนกระทั่งช่วงกลางทศวรรษ 1980 ก็ได้มีการพัฒนา X-Window ขึ้นมาทำให้การใช้งาน UNIX ก็เริ่มมี GUI (Graphic User Interface) เกิดขึ้น

บริษัท AT&T ยังคงผลักดันการพัฒนาระบบปฏิบัติการ UNIX จนกระทั่งถึงรุ่น System V Release 4 (SVR4) ทาง AT&T ได้พยายามรวมข้อกำหนดและมาตราฐานต่างๆ ของ BSD UNIX และ XENIX

เข้าไปด้วยกัน ซึ่งทั้งสองสามารถถูกนำไปใช้งานได้บน SVR4 ได้ด้วยเหตุการณ์ครั้งนี้ทำให้กลุ่มนักพัฒนาและบริษัทที่เกี่ยวข้องเริ่มวิตกกังวลว่าจะเกิดการผูกขาดในการกำหนดมาตราฐานของระบบปฏิบัติการ UNIX จากบริษัท AT&T หรือไม่ ทั้งหมดจึงได้รวมตัวกันจัดตั้งองค์กร Open Software Foundation (OSF) ขึ้นเพื่อร่วมวิจัยและร่วมกำหนดมาตราฐานต่างๆของระบบ UNIX ในเวลาต่อมา



รูปที่ 1.4 TimeLine ของระบบปฏิบัติ UNIX/LINUX

นอกจาก UC Berkeley แล้ว ก็ยังมีบริษัทรายอื่นๆที่พัฒนาเครื่องคอมพิวเตอร์ระดับสูงเพื่อใช้ในด้านธุรกิจ เช่น บริษัทซันไมโครซิสเต็ม (SunOS และ Solaris) บริษัท DEC (เครื่อง Ultrix จนเปลี่ยนชื่อเป็น OSF/1) บริษัทไมโครซอฟต์ (เครื่อง XENIX) บริษัทไอบีเอ็ม (เครื่อง AIX) ซึ่งส่วนใหญ่จะยึดแนวระบบปฏิบัติการของ BSD หรือไม่ก็ System V สำหรับระบบปฏิบัติการ UNIX นั้น ปัจจุบันได้กลายเป็นเครื่องหมายการค้าจดทะเบียน (Registered Trademark) ของหน่วยงานที่ชื่อ The Open Group ซึ่งเป็นหน่วยงานที่กำหนดและรับรองมาตรฐานของระบบปฏิบัติการ UNIX ไว้ 2 แบบคือ

- ระบบปฏิบัติการที่ได้มาตรฐาน UNIX ซึ่งใช้มาตรฐานของ The Open Group ในการพัฒนาขึ้นมา เช่น Digital UNIX, SCO UNIX, IBM's OpenEdition MVS
- ระบบปฏิบัติการคล้าย UNIX (UNIX Compatible) เป็นระบบปฏิบัติการที่มีลักษณะคล้ายระบบปฏิบัติการ UNIX แต่ยังไม่ได้จดทะเบียนรับรองเป็นทางการ เช่น Sun Solaris, IBM AIX, Linux

โครงการ GNU (GNU Project) ได้ถูกริเริ่มจากนักวิจัยจาก M.I.T ชื่อ Richard Stallman เนื่องจากระบบปฏิบัติการ UNIX ไม่ได้ฟรีอีกต่อไปแล้ว โดยวัตถุประสงค์หลักของโครงการนี้คือการพยายามเริ่มต้นสร้าง C compiler (gcc), make (GNU make), Emacs, C library (glibc), และ coreutils (เช่นคำสั่ง ls, cp เป็นต้น) ใหม่ทั้งหมดเพื่อให้ทุกคนสามารถนำไปใช้ได้ฟรี แต่อย่างไรก็ตามโครงการนี้ก็ยังคงขาดตัวแกนกลางสำคัญสำหรับระบบปฏิบัติการที่เรียกว่าเคอร์เนล (Kernel) จนกระทั่งปี ค.ศ. 1991 นักศึกษา

สาขาวิทยาการคอมพิวเตอร์ มหาวิทยาลัย Helsinki ประเทศฟินแลนด์ชื่อ Linus Torvalds ได้มีแนวคิดที่จะสร้างระบบปฏิบัติการแบบเปิดและฟรี โดยมีพื้นฐานคล้ายระบบปฏิบัติการ UNIX ซึ่งได้ใช้เครื่องมือจากโครงการ GNU ทั้งหมด ไม่ว่าจะเป็น C library, gcc, binutils, fileutils, make, emacs เป็นต้น รวมทั้งการพัฒนา Kernel โดยพยายามพัฒนาโปรแกรมทั้งหมดตามมาตราฐาน POSIX เช่นเดียวกันระบบปฏิบัติการ UNIX จนในที่สุดก็สามารถออกมาได้สำเร็จในชื่อระบบปฏิบัติการ Minix และทาง Linus Torvalds ก็ได้เปิดเผยโค้ดโปรแกรมของระบบปฏิบัติการทั้งหมดสู่สาธารณะ เพื่อให้นักพัฒนาทั่วโลกช่วยกันปรับปรุงแก้ไข จนกระทั่งออกเวอร์ชัน 1.0 (ค.ศ. 1994) และ เวอร์ชัน 2.2 (ค.ศ. 1999) ตามลำดับ จนได้ชื่อว่าระบบปฏิบัติลีนุกซ์ (Linux) ในที่สุด

ประเภทของ Licenses

จากการเกิดขึ้นของโครงการ GNU โดย Richard Stallman ที่ต้องการให้นักพัฒนาและผู้ใช้งานสามารถนำไปใช้ได้ฟรีโดยไม่มีค่าใช้จ่าย แต่ก็จะต้องนำไปใช้ภายใต้เงื่อนไขการคุ้มครองตามลิขสิทธิ์ที่ผู้พัฒนากำหนดไว้ ดังตัวอย่างลิขสิทธิ์บางส่วนตามตารางข้างล่างนี้

License	เงื่อนไข
GNU General Public License (GPL)	หากมีการนำโค้ดโปรแกรมไปทำการแก้ไข หรือ เขียนโปรแกรมขึ้นมาใหม่เพื่อเรียกใช้ฟังก์ชัน คลาส หรือไลบรารี โปรแกรมใหม่ที่เกิดขึ้นนั้นจะต้องมี GPL license ติดไปด้วย ซึ่งลักษณะของสัญญาอนุญาต GPL มีลักษณะ “เสรี” ดังนี้ - เสรีภาพในการใช้งาน ไม่ว่าใช้สำหรับจุดประสงค์ใด - เสรีภาพในการศึกษาการทำงานของโปรแกรม และแก้ไขโค้ด - เสรีภาพในการจำหน่ายโปรแกรม -เสรีภาพในการปรับปรุงและเปิดให้บุคคลทั่วไปใช้และพัฒนาต่อ โดยมีเพียงเงื่อนไขว่าการนำไปใช้หรือนำไปพัฒนาต่อ จำเป็นต้องใช้สัญญาอนุญาตเดียวกัน
GNU Lesser General Public License (LGPL)	หากมีการนำโค้ดโปรแกรมไปทำการแก้ไข หรือ เขียนโปรแกรมขึ้นมาใหม่เพื่อเรียกใช้ฟังก์ชัน คลาส หรือไลบรารี โปรแกรมใหม่ที่เกิดขึ้นนั้นไม่จำเป็นต้องติด LGPL (Lesser GPL) ไปด้วย แต่ส่วนของโค้ดโปรแกรมชุดเดิมก็ยังคงต้องมี LGPL ติดไปด้วย โปรแกรมส่วนที่เขียนขึ้นมาเองสามารถนำไปขายได้แต่จะต้องระบุอย่างชัดเจนว่าได้นำโปรแกรมส่วนใดมาใช้ในการพัฒนาบ้าง

License	เงื่อนไข
Apache License	ให้แสดงในเอกสารว่าได้ใช้โค้ดโปรแกรม หรือ ไลบรารี ที่เป็น Apache License ส่วนโปรแกรมที่พัฒนาขึ้นมาใหม่เองสามารถเลือกใช้ license แบบไหนก็ได้
BSD Licenses	โค้ดโปรแกรมที่ได้ทำการแก้ไขไม่จำเป็นต้องเปิดเผยโค้ดโปรแกรม มีเพียงข้อความเจ้าของสัญญาอนุญาตเดิมเท่านั้นที่ต้องแสดง
Creative Commons Licenses	ถูกนำไปใช้ในลิขสิทธิ์ของผลงาน โดยอาจเป็นผลงานการเขียน รูปภาพ หรือการออกแบบ โดยเจ้าของผลงานสามารถเลือกได้ว่าผลงานของตัวเองจะให้มี license เป็นแบบไหน เช่น แสดงที่มา , แสดงที่มาไม่ใช้เพื่อการค้า , แสดงที่มาไม่ใช้เพื่อการค้าไม่ดัดแปลง เป็นต้น

ปรัชญา และความรู้พื้นฐานของระบบปฏิบัติการลีนุกซ์

ในอดีตการใช้งานระบบปฏิบัติการลีนุกซ์ เป็นเรื่องที่หลายคนคิดว่ามีความซับซ้อนและยากต่อการทำความเข้าใจ เพราะอาจต้องใช้ความเข้าใจในเชิงลึกเกี่ยวกับระบบคอมพิวเตอร์และระบบเครือข่ายพอสมควร เช่นการใช้คำสั่งในการสั่งการระบบปฏิบัติการ การตั้งค่าการทำงานบนบอร์ดสมองกลฝังตัว หรือการเขียนโปรแกรมประยุกต์เพื่อใช้ในการควบคุมจัดการระบบการสื่อสารต่างๆ หรือการติดต่อสื่อสารข้อมูลระหว่างฮาร์ดแวร์ เหล่านี้ ล้วนต้องใช้ความรู้และทักษะหลายด้านประกอบกันจึงทำให้มีใช้งานในอยู่วงจำกัด ตัวอย่างเช่น ศูนย์คอมพิวเตอร์แม่ข่าย บริษัทที่ออกแบบอุปกรณ์ที่มีระบบสมองกลฝังตัวทางด้านระบบสื่อสารและโทรคมนาคม ห้องปฏิบัติการภายในมหาวิทยาลัย หรือหน่วยวิจัยภาครัฐ/ภาคเอกชน เป็นต้น

แต่อย่างไรก็ตามในปัจจุบันหลายคนก็ไม่สามารถปฏิเสธได้ว่าระบบปฏิบัติการ UNIX/LINUX เริ่มเข้ามามีบทบาทในชีวิตประจำวันมากขึ้น ไม่ว่าจะเป็นระบบอีเมล์ในองค์กร ระบบไฟล์ภายในองค์กร เว็ปแอพพิเคชั่นที่โด่งดังต่างๆ (facebook, google, twitter) ล้วนแล้วแต่ทำงานอยู่ภายใต้ระบบปฏิบัติการ UNIX/LINUX เกือบทั้งสิ้น นอกจากนั้นภายในอุปกรณ์เครื่องใช้ไฟฟ้า อุปกรณ์สื่อสารชนิดพกพา ก็ถูกสั่งการด้วยระบบปฏิบัติการอันทรงพลังอย่างระบบปฏิบัติการลีนุกซ์อยู่เบื้องหลังในการทำหน้าที่ควบคุมดูแลการทำงานของระบบภายในอุปกรณ์ ทำหน้าที่ติดต่อกับผู้ใช้ผ่าน GUI (Graphic User Interface) และทำหน้าที่ในการสื่อสารผ่านระบบเครือข่ายไปยังศูนย์กลาง เป็นต้น ดังนั้นนักพัฒนาไม่ว่าจะระดับใดหรือนิสิตนักศึกษาที่เรียนในสายวิทยาศาสตร์คอมพิวเตอร์ เทคโนโลยีสารสนเทศ วิศกวรรมไฟฟ้า วิศวกรรมคอมพิวเตอร์ วิศวกรรมระบบควบคุม ทั้งหลาย จึงมีความจำเป็นอย่างยิ่งที่จะต้องเริ่มทำความเข้าใจหลักการทำงานของระบบปฏิบัติการลีนุกซ์ ไม่ว่าจะเป็นการใช้คำสั่งพื้นฐาน การใช้งานโปรแกรมประยุกต์

ต่างๆ รวมทั้งการพัฒนาภาษาโปรแกรม ภายใต้ระบบปฏิบัติการลีนุกซ์ เพื่อเพิ่มโอกาสให้กับตัวเองในอนาคต ในสายอาชีพนักพัฒนาโปรแกรม นักพัฒนาระบบสมองกลฝังตัว นักพัฒนาระบบสื่อสารแบบใหม่ นักวิจัยทางด้านการวิศวกรรมการแพทย์ วิศวกรรมชีวภาพ นักธุรกิจทางด้านบริการสารสนเทศ นักธุรกิจทางด้านอุปกรณ์ใช้เทคโนโลยีขั้นสูง เป็นต้น

จุดเริ่มต้นเพื่อก้าวเข้าสู่นักพัฒนาที่ดีที่สุดคือ การเริ่มต้นการใช้งานระบบปฏิบัติการลีนุกซ์ ทำความเข้าใจปรัญชาพื้นฐานของระบบปฏิบัติการลีนุกซ์แล้วลงมือปฏิบัติ ด้วยการทดลองใช้คำสั่งต่างๆจากง่ายๆ ไปสู่ยากและทำอย่างต่อเนื่อง ผลจากการเรียนรู้ฝึกฝนเหล่านี้จะนำไปสู่เส้นทางที่ยิ่งใหญ่และเต็มไปด้วยทางเลือกมากมาย จะพบเจอกับแหล่งความรู้อันมหาศาลที่จะมีโอกาสสร้างสรรสิ่งใหม่ๆให้เกิดขึ้นได้อย่างเหลือเชื่อ ซึ่งเนื้อหาภายในบทนี้เป็นการรวบรวมและสรุปคำสั่งและความรู้พื้นฐานที่สำคัญในการใช้ระบบปฏิบัติการลีนุกซ์ซึ่งจะเหมาะสำหรับนักพัฒนารุ่นเยาว์จนถึงนักพัฒนาระดับกลาง ประกอบไปด้วยหัวข้อดังต่อไปนี้

- เรียนรู้กระบวนการทำงานของเชลล์ (Shell) และ Standard I/O Stream
- ตัวแปรภายในระบบ (Environment Variable)
- การเรียกใช้งานคำสั่งภายในเชลล์ และภายนอกเชลล์
- คำสั่งพื้นฐานสำหรับนักพัฒนาด้านระบบสมองกลฝังตัว
- คำสั่งพื้นฐานในการตั้งค่าและดูแลจัดการบริการด้านระบบเครือข่าย

กระบวนการทำงานของเชลล์ และชุดคำสั่งที่เกี่ยวข้อง

ภายในระบบปฏิบัติการลีนุกซ์เกือบทุกสิ่งทุกอย่างที่อยู่ภายในระบบคือไฟล์ทั้งสิ้น (Almost Everything is File) ได้แก่

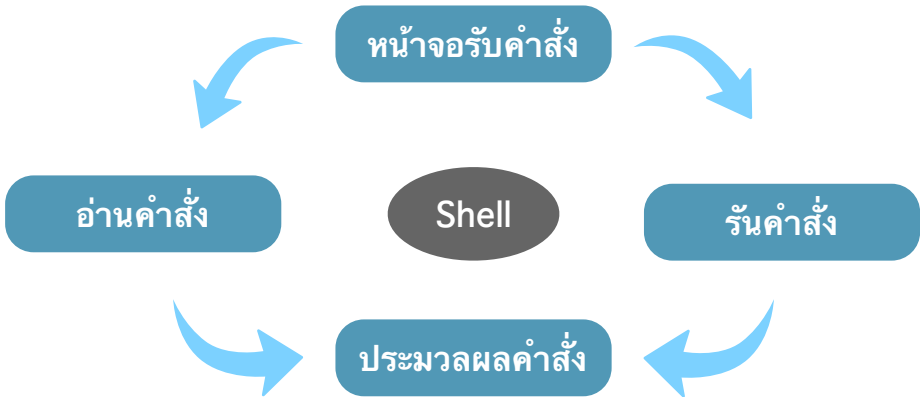
- ไฟล์ปกติทั่วไป (Regular files) เช่นไฟล์เอกสาร ไฟล์มัลติมีเดีย ไฟล์โปรแกรม
- ไดเรกทอรี (Directories) ไฟล์ที่บรรจุรายการของไฟล์ต่างๆ
- ไฟล์เชื่อมสัญลักษณ์ (Symbolic links) ไฟล์ที่อ้างอิงไปยังไฟล์อื่น
- ไฟล์อุปกรณ์ (Devices and Peripherals) ไฟล์ที่เชื่อมต่อกับอุปกรณ์ฮาร์ดแวร์ของระบบ
- ไฟล์ท่อส่งข้อมูล (Pipe) ไฟล์ที่ใช้เป็นท่อเชื่อมระหว่างโปรแกรม เพื่อส่งค่าผลลัพธ์ (output) ของโปรแกรมหนึ่ง ให้เป็นค่านำเข้า (input) ของอีกโปรแกรมหนึ่ง
- ไฟล์ท่อซ็อกเก็ต (Socket) ไฟล์ที่เป็นเป็นท่อเชื่อมต่อการสื่อสารข้อมูลระหว่างโปรเซส ที่อยู่ในเครื่องเดียวกัน หรือต่างเครื่องผ่านระบบเครือข่ายได้

เชลล์ (Shell)

shell เป็นตัวกลางในการรับคำสั่ง (Command Line) จากผู้ใช้แล้วจะทำการแปลชุดคำสั่ง (Command Line Interpreter - CLI) ที่ผู้ใช้ป้อนเข้ามา โดยขบวนการภายในตัว shell จะซ่อนรายละเอียดอันซับซ้อนของระบบปฏิบัติการเอาไว้ โดยที่ผู้ใช้จะไม่รู้ว่าหลังจากที่ป้อนคำสั่งไปแล้วภายในจะ

ต้องมีขบวนการเช่นไรบ้าง ตัวอย่างเช่น ถ้าผู้ใช้ต้องการทราบหน่วยความจำที่เหลือหรือพื้นที่ของฮาร์ดดิสก์ที่เหลืออยู่ ผู้ใช้เพียงพิมพ์คำสั่ง free/df ตัว shell ก็จะทำหน้าเชื่อมต่อและเข้าไปจัดการในหน่วยความจำหรือฮาร์ดดิสก์ให้ตรวจสอบตัวเองว่าขณะนี้มีการใช้งานเก็บข้อมูลไปเท่าไหร่แล้วและเหลือพื้นที่ให้ใช้งานได้อีกเท่าไหร่ ซึ่งขบวนการดังที่ได้กล่าวข้างต้นเป็นการติดต่อสื่อสารในระดับล่างที่เรียกว่าระดับเคอร์เนล ซึ่งถือได้ว่าเป็นแกนกลางสำคัญในการควบคุมการทำงานระบบทั้งหมดภายในเครื่องคอมพิวเตอร์ โดยจะกล่าวถึงรายละเอียดของเคอร์เนลในบทถัดๆไป

นอกจากนั้น shell ยังสามารถรองรับการเขียนชุดคำสั่งมากกว่าหนึ่งคำสั่งพร้อมกัน และสามารถรับชุดคำสั่งล่วงหน้าให้ทำงานตามเงื่อนไขต่างๆที่ได้กำหนดไว้ในลักษณะสคริปท์ (Script) ที่ถูกเก็บลงในไฟล์ได้ ทำให้เพิ่มความสะดวกและมีประสิทธิภาพในการใช้งานระบบปฏิบัติการลีนุกซ์ โดยเฉพาะผู้ใช้ที่มีทักษะอยู่ระดับหนึ่งในการเขียนสคริปท์หรือที่เรียกกันว่าเชลล์สคริปท์ (Shell script) นั้น และนอกจากนั้นลูกเล่นความสามารถจะมากน้อยเพียงใดก็ยังขึ้นอยู่กับยี่ห้อหรือรุ่นโปรแกรม shell นั้นด้วย ในปัจจุบัน shell ที่มีอิทธิพลและเป็นที่นิยมมากที่สุดตัวหนึ่งคือ Bourne shell (ปัจจุบันกลายเป็น Bourne Again shell) และ C shell



รูปที่ 1.5 ขบวนการทำงานของ Shell

ตัวอย่างการตรวจสอบชนิดของ shell ที่กำลังใช้งานอยู่ในระบบด้วยคำสั่ง echo และตรวจสอบรุ่นของ shell ด้วย --version

```
$ echo $SHELL
/bin/bash

$ bash --version
GNU bash, version 4.1.5(1)-release (x86_64-pc-linux-gnu)
Copyright (C) 2009 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

หากต้องการทราบถึงรายการชุดคำสั่งที่ถูกฝังอยู่ภายใน shell โดยใช้คำสั่งดังข้างล่างนี้

```
$ bash -c help
GNU bash, version 4.1.5(1)-release (x86_64-pc-linux-gnu)
These shell commands are defined internally.  Type `help' to see this list.
Type `help name' to find out more about the function `name'.
Use `info bash' to find out more about the shell in general.
Use `man -k' or `info' to find out more about commands not in this list.
A star (*) next to a name means that the command is disabled.

 job_spec [&]                            history [-c] [-d offset] [n] or hist>
 (( expression ))                        if COMMANDS; then COMMANDS; [ elif C>
 . filename [arguments]                  jobs [-lnprs] [jobspec ...] or jobs >
 :                                       kill [-s sigspec | -n signum | -sigs>
 [ arg... ]                              let arg [arg ...]
 [[ expression ]]                        local [option] name[=value] ...
 alias [-p] [name[=value] ... ]          logout [n]
 bg [job_spec ...]                       mapfile [-n count] [-O origin] [-s c>
 bind [-lpvsPVS] [-m keymap] [-f filen>  popd [-n] [+N | -N]
 break [n]                               printf [-v var] format [arguments]
 builtin [shell-builtin [arg ...]]       pushd [-n] [+N | -N | dir]
 caller [expr]                           pwd [-LP]
 case WORD in [PATTERN [| PATTERN]...)>  read [-ers] [-a array] [-d delim] [->
 cd [-L|-P] [dir]                        readarray [-n count] [-O origin] [-s>
 command [-pVv] command [arg ...]        readonly [-af] [name[=value] ...] or>
 compgen [-abcdefgjksuv] [-o option]  >  return [n]
 complete [-abcdefgjksuv] [-pr] [-DE] >  select NAME [in WORDS ... ;] do COMM>
 compopt [-o|+o option] [-DE] [name ..>  set [--abefhkmnptuvxBCHP] [-o option>
 continue [n]                            shift [n]
 coproc [NAME] command [redirections]    shopt [-pqsu] [-o] [optname ...]
 declare [-aAfFilrtux] [-p] [name[=val>  source filename [arguments]
 dirs [-clpv] [+N] [-N]                  suspend [-f]
 disown [-h] [-ar] [jobspec ...]         test [expr]
 echo [-neE] [arg ...]                   time [-p] pipeline
 enable [-a] [-dnps] [-f filename] [na>  times
 eval [arg ...]                          trap [-lp] [[arg] signal_spec ...]
 exec [-cl] [-a name] [command [argume>  true
 exit [n]                                type [-afptP] name [name ...]
 export [-fn] [name[=value] ...] or ex>  typeset [-aAfFilrtux] [-p] name[=val>
 false                                   ulimit [-SHacdefilmnpqrstuvx] [limit>
 fc [-e ename] [-lnr] [first] [last] o>  umask [-p] [-S] [mode]
 fg [job_spec]                           unalias [-a] name [name ...]
 for NAME [in WORDS ... ] ; do COMMAND>  unset [-f] [-v] [name ...]
 for (( exp1; exp2; exp3 )); do COMMAN>  until COMMANDS; do COMMANDS; done
 function name { COMMANDS ; } or name >  variables - Names and meanings of so>
 getopts optstring name [arg]            wait [id]
 hash [-lr] [-p pathname] [-dt] [name >  while COMMANDS; do COMMANDS; done
 help [-dms] [pattern ...]               { COMMANDS ; }
```

สามารถทราบรายละเอียดของแต่ละคำสั่งที่อยู่ภายใน shell โดยพิมพ์ help <*ชื่อคำสั่ง*> ตัวอย่างเช่น

```
$ help times
times: times
    Display process times.
    Prints the accumulated user and system times for the shell and all of its
    child processes.
    Exit Status:
    Always succeeds.
```

Echo

Echo เป็นคำสั่งที่ใช้ในการแสดงข้อความใดๆที่ต้องการให้ถูกปรากฏบนหน้าต่างเทอร์มินัล ดังตัวอย่างการใช้คำสั่งดังนี้

```
$ echo Welcome to Embedded System
Welcome to Embedded System!
$ echo Welcome      to      Embedded      System
Welcome to Embedded System!
$ echo 'Welcome      to      Embedded      System'
Welcome      to      Embedded      System!
```

จากตัวอย่างจะเห็นว่าคำสั่ง echo จะไม่สนใจช่องว่างว่าจะมีกี่ช่องว่าง การแสดงผลจะถูกตัดช่องว่างเหลือเพียง 1 ช่องว่างเสมอ แต่ถ้าต้องการให้แสดงผลลัพธ์ตามระยะห่างที่ต้องการจะต้องใช้เครื่องหมาย อัญประกาศ (‘ -Apostrophe) หรือเครื่องหมายคำพูด (“ - Quotation Mark) ครอบข้อความนั้น เนื่องจากใน Bash shell นั้นจะใช้ white space (โดยการกดปุ่ม space bar) ในการแยกข้อความออกเป็น token หรือเรียกว่า พารามิเตอร์เพื่อใช้ในการแสดงผล แต่ถ้ามีการใส่เครื่องหมายครอบข้อความเช่น ‘Burapha University’ ข้อความนั้นจะถูกเก็บไว้ใน token เพียงตัวเดียว

การใช้งานคำสั่ง echo จะมีตัวเลือก (Option) หลายแบบ (ซึ่งสามารถดูรายละเอียดคำสั่งได้โดยพิมพ์ man echo) ยกตัวอย่างเช่น การใช้ตัวเลือก *-n* ต่อท้ายคำสั่ง echo จะหมายถึงการไม่ขึ้นบรรทัดใหม่ (New Line)

```
$ echo -n "My name is Android"
My name is Android$
```

แต่ถ้าต้องการขึ้นบรรทัดใหม่ตามอักขระพิเศษ \n (new line) ต้องระบุตัวเลือก -e หลังคำสั่ง echo

```
$ echo -e "My name is \nAndroid"
My name is
Android
```

นอกจากอักขระ \n แล้วยังมีอักขระในการควบคุมการแสดงผลข้อความดังรายละเอียดในตารางข้างล่างนี้

ตาราง 1.1 แสดงการใช้งานอักขระพิเศษ

ESCAPE SEQUENCE	รายละเอียด
\a	Alert (bell)
\b	Backspace

ESCAPE SEQUENCE	รายละเอียด
\c	หยุดการขึ้นบรรทัดใหม่ (ลักษณะเดียวกับการใช้งาน -n)
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab

ในการใช้งาน Bash shell นั้นยังมีอักขระเฉพาะมากมายที่ไม่สามารถนำมาใช้แสดงเป็นตัวข้อความได้ เนื่องจากถูกใช้เป็นเครื่องมือในการจัดการอินพุทของคำสั่ง เช่น '|', '&', ';', '(', ')', '<', '>' แต่ถ้าต้องการแสดงอักขระเหล่านี้ให้ปรากฏอยู่ในข้อความจะต้องเติมสัญลักษณ์ \ นำหน้าอักขระเสมอ หรือใช้เครื่องหมายอัญประกาศครอบทั้งข้อความที่มีอักขระผสมอยู่ก็ได้

ตัวอย่างการใช้สัญลักษณ์ \ นำหน้าอักขระที่ต้องการแสดงเป็นข้อความ

```
$ echo I love Linux & Android
[1] 30136
I love Linux
Android: command not found
[1]+  Done                    echo I love Linux
[1]+  Done                    echo I love Linux
$ echo I love Linux \& Android
I love Linux & Android
$ echo 'I love Linux & Android'
I love Linux & Android
```

Command Sequences

ผู้ใช้สามารถป้อนคำสั่งมากกว่าหนึ่งคำสั่งไปพร้อมกันโดยใช้ตัวดำเนินการควบคุม (Control Operators) ได้แก่ ‘||’, ‘&&’, ‘&’, ‘;’, ‘;;’, ‘|’, ‘(‘, ‘)’ เป็นต้น ซึ่งการเรียงชุดคำสั่งอย่างง่ายที่สุดในกรณีที่มีมากกว่า 1 คำสั่งเป็นต้นไป จะใช้เครื่องหมาย ‘;’ อัฒภาค (semicolon) shell จะรับคำสั่งและดำเนินการทีละคำสั่งตามลำดับ ในกรณีที่จะมีการตรวจสอบผลลัพธ์การทำงานของแต่ละคำสั่งว่าสำเร็จหรือล้มเหลวนั้น ระบบปฏิบัติการลีนุกซ์จะมีการคืนค่ากลับคือถ้าคืนค่ากลับมาเลขศูนย์จะหมายถึงคำสั่งดำเนินการสำเร็จ แต่ถ้าเป็นตัวเลขอื่นๆจะถือว่าคำสั่งนั้นทำงานล้มเหลว ดังนั้นผู้ใช้สามารถกำหนดเส้นทางการทำงานตามเงื่อนไขที่ออกมาโดยใช้ตัวดำเนินการ AND ‘&&’ และ OR ‘||’ ตัวอย่างเช่น

```
$ ls
android  ccache     Downloads  kernel   output.txt  Templates  workspace
aosp     Desktop    error.log  Music    Pictures    test.txt
```

```
bin      Documents  git        one.txt  Public      Videos
$ ls test.txt && echo "OK... File exists"
test.txt
OK... File exists
$ ls mail.txt && echo "OK... File exists"
ls: cannot access mail.txt: No such file or directory
```

จากคำสั่งข้างต้น ไฟล์ test.txt มีอยู่ในไดเรกทอรีจึงทำให้ระบบปฏิบัติการลีนุกซ์คืนค่ากลับมาเป็นศูนย์ คำสั่งถัดไปจึงทำงานต่อได้ แต่ในขณะที่ไฟล์ mail.txt ไม่ได้อยู่ในไดเรกทอรี แสดงถึงการทำงานล้มเหลว ทำให้ไม่มีการทำคำสั่งตัวถัดไป แต่หากใช้ตัวดำเนินการ '||' คำสั่งถัดมาจะถูกทำงานในกรณีที่คำสั่งแรกมีการคืนค่ากลับมาไม่เท่ากับศูนย์ ดังตัวอย่างข้างล่าง

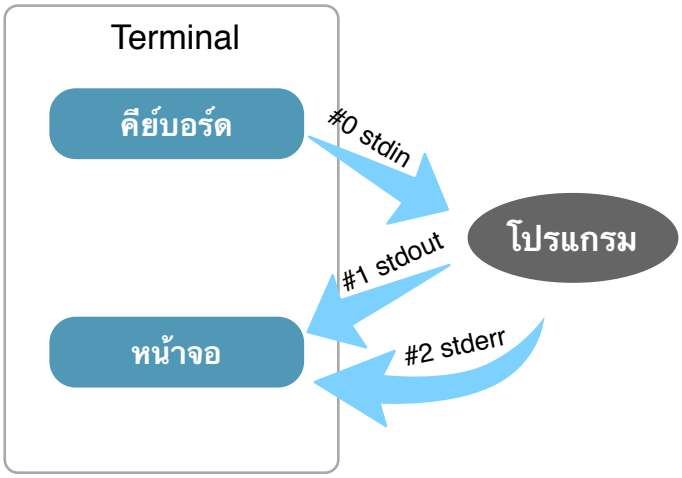
```
$ ls test.txt || echo "OK... File exists"
test.txt
$ ls mail.txt || echo "No File exists"
ls: cannot access mail.txt: No such file or directory
ON File exists
```

แต่หากต้องการนำตัวดำเนินการ && และ || มาประยุกต์ใช้ในการตรวจสอบเงื่อนไขเหมือนชุดคำสั่ง if (เงื่อนไข) *..(จริง)..* else *..(เท็จ)..* หรือ if (เงื่อนไข) ? *..(จริง)..* : *..(เท็จ)..* ในภาษาโปรแกรมทั่วไป จะมีรูปแบบการเขียนดังนี้

```
$ ls test.txt && echo "OK...File exists" || echo "Oh Bad... File not
found"
test.txt
OK...File exists
$ ls mail.txt && echo "OK...File exists" || echo "Oh Bad... File not
found"
ls: cannot access mail.txt: No such file or directory
Oh Bad... File not found
```

Standard I/O

รายละเอียดขบวนการทำงานของ Shell ในระบบปฏิบัติการลีนุกซ์ จะมีพื้นฐานสำคัญคือ วิธีการเชื่อมต่อระหว่างโปรแกรมและสิ่งแวดล้อมของตัวโปรแกรมเองภายในเทอร์มินัล (Terminal) ที่เรียกว่า I/O ซึ่งรูปข้างล่างนี้เป็นการแสดงการเชื่อมต่อของอุปกรณ์ I/O มาตราฐานพื้นฐานของระบบที่มีอยู่ 3 ชนิด ได้แก่



รูปที่ 1.6 Standard I/O

ตาราง 1.2 รายละเอียดพื้นฐานของ Standard I/O

STREAM มาตราฐาน	FD	รายละเอียด
stdin (Standard Input Stream)	0	สำหรับรับคำสั่งจากผู้ใช้เพื่อส่งต่อให้โปรแกรม อาทิเช่นการรับข้อมูลคำสั่งจากการกดคีย์บอร์ด
stdout (Standard Output)	1	สำหรับแสดงผลลัพธ์ที่ถูกส่งออกมาจากโปรแกรม เพื่อส่งข้อความผลลัพธ์ออกมาแสดงบนจอภาพ
stderr (Standard Error)	2	สำหรับแสดงผลความผิดพลาดเกิดจากการทำงานของโปรแกรมที่รับคำสั่งมาประมวลผล ออกจากหน้าจอภาพ

Redirections

ระบบปฏิบัติการลีนุกซ์นั้นได้เตรียมเครื่องมือตัวดำเนินการที่สามารถควบคุมกลไกการไหลของข้อมูลจากทิศทางหนึ่งไปยังอีกทิศทางหนึ่งหรืออธิบายง่ายๆคือการเปลี่ยนเส้นทางข้อมูลว่าจะให้ออกตัว standard stream ตัวใด โดยการใช้ตัวดำเนินการ “<” แทน stdin (Standard Input) และ “>” แทน stdout (Standard Output) ซึ่งตัวดำเนินการ redirection นั้นมีอยู่ด้วยกัน 5 แบบตามรายละเอียดในตารางข้างล่าง

ตาราง 1.3 ตัวดำเนินการ redirection

ตัวดำเนินการ	รายละเอียด
< *ไฟล์*	เปิดไฟล์สำหรับอ่านข้อมูลภายในไฟล์
<< *token*	ใช้ในกรณีที่เป็นคำสั่งหรือเชลล์สคริปท์ที่ต้องการรับค่าจนกระทั่งเจอ token
> *ไฟล์*	เปิดไฟล์สำหรับเขียนทับข้อมูลใหม่

ตัวดำเนินการ	รายละเอียด
>> ไฟล์	เปิดไฟล์สำหรับเขียนต่อท้ายจากข้อมูลเดิม
n>&m	เปลี่ยนเส้นทางของ File Descriptor (FD) เดิม n ไปที่ใหม่ m

ยกตัวอย่างเช่น การใช้งานการส่งอินพุทจากไฟล์ /etc/passwd ให้กับคำสั่ง grep และการส่งอินพุทจากการป้อนข้อความให้กับคำสั่ง sort ดังแสดงข้างล่าง

```
$ grep -i student < /etc/passwd
student:x:1000:1000:EE-Burapha Student,,,:/home/student:/bin/bash
$ sort << END
> 55233424 Wiroon Sriborrirux
> 55237346 Nayot Kurukitkoson
> 55236477 Panuwat Dankhang
> END
55233424 Wiroon Sriborrirux
55236477 Panuwat Dankhang
55237346 Nayot Kurukitkoson
$ sort -k2 << END
> 55233424 Wiroon Sriborrirux
> 55237346 Nayot Kurukitkoson
> 55236477 Panuwat Dankhang
> END
55237346 Nayot Kurukitkoson
55236477 Panuwat Dankhang
55233424 Wiroon Sriborrirux
```

เมื่อมีการใช้ตัวดำเนินการ “>” ผลลัพธ์จากคำสั่งจะถูกส่งไปเก็บไว้ในไฟล์ /tmp/results แทนที่จะออกหน้าจอ ดังตัวอย่างข้างล่าง

```
$ grep -i student /etc/passwd > /tmp/results
$ ls /tmp/
CRX_75DAF8CB7768    orbit-student       ssh-pqdoFN1336  vmware-root-2
hsperfdata_student  pulse-Xti8iSZ9STOh  VMwareDnD       vmware-student
keyring-27UIsq      results             vmware-root
$ cat /tmp/results
student:x:1000:1000:EE-Burapha Student,,,:/home/student:/bin/bash
```

ตัวอย่างการใช้คำสั่ง cat เพื่อแสดงข้อมูลภายในไฟล์ จะมีการแสดงข้อความผิดพลาด (Error message) ออกทางหน้าจอ ถ้าไฟล์นั้นไม่มีอยู่ในไดเรกทอรี

```
$ cat one.txt two.txt 2>&1
This is the data inside
cat: two.txt: No such file or directory
```

สามารถกรองข้อความผิดพลาดให้ไปเก็บไว้ในไฟล์ชื่อ error.log ด้วยตัวดำเนินการ “2>”

```
$ cat one.txt two.txt 2> error.log
This is the data inside
```

ในกรณีที่ไม่ต้องการเก็บข้อความผิดพลาด สามารถทำได้โดยส่งไปให้ไฟล์ชื่อ /dev/null

```
$ cat one.txt two.txt 2> /dev/null
This is the data inside
```

บันทึกข้อความผิดพลาดเพิ่มต่อเข้าไปในไฟล์ error.log ด้วยตัวดำเนินการ “2>>”

```
$ cat one.txt two.txt three.txt 2>> error.log
This is the data inside
$ cat error.log
cat: two.txt: No such file or directory
cat: two.txt: No such file or directory
cat: three.txt: No such file or directory
```

ตัวแปรสภาพแวดล้อมของระบบ

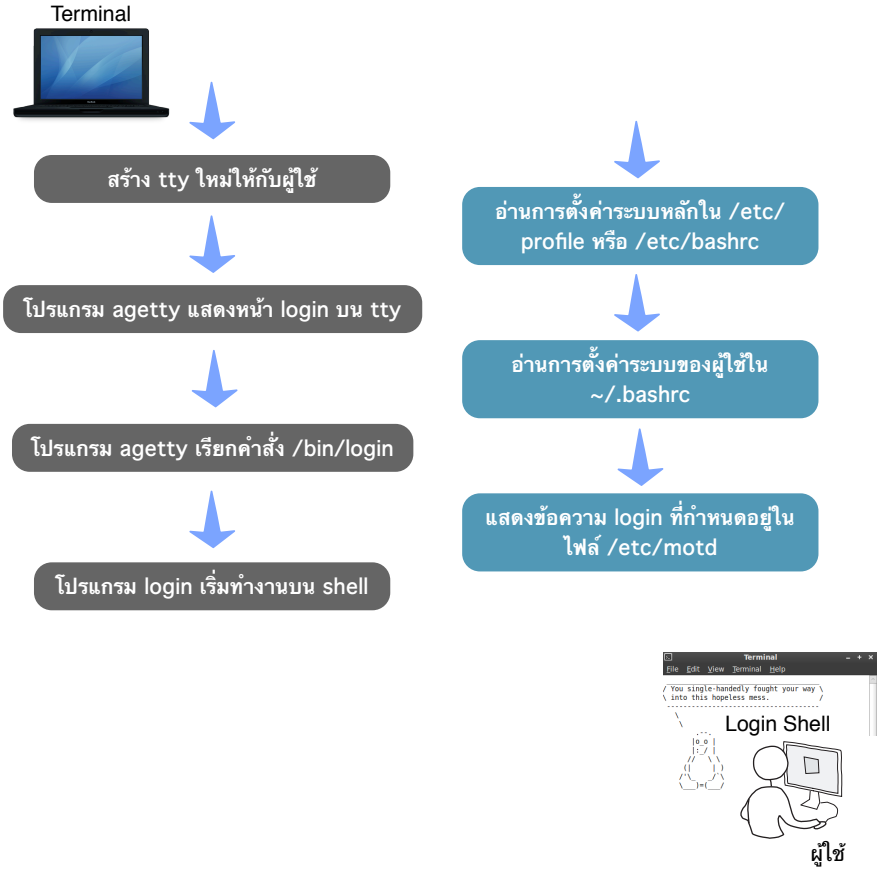
ตัวแปรสภาพแวดล้อม (Environment variables) เป็นตัวแปรที่มีความสำคัญมากในระบบปฏิบัติการลีนุกซ์ เนื่องจากเป็นตัวแปรกลางที่ shell หรือ โปรแกรมประยุกต์ต่างๆสามารถเรียกใช้งานได้ อาทิ เช่น ตัวแปรที่เก็บข้อมูลของ shell หรือตัวแปรที่เก็บข้อมูลของไดเรกทอรีประจำตัวของผู้ใช้ (Home Directory) และยังมีตัวแปรต่างๆ อีกมากมายที่ shell ได้มีการตั้งค่าไว้ในขณะที่ shell เริ่มทำงาน ตัวอย่าง เช่น bash shell จะมีการเรียกไฟล์เริ่มต้นอยู่ 2 ไฟล์ได้แก่ ไฟล์ .profile และไฟล์ .bashrc

```
$ cat .profile  <---- ในกรณี Ubuntu เวอร์ชัน 13.04 ขึ้นไป
# .bash_profile
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi
# User specific environment and startup programs
export USE_CCACHE=0
export CCACHE_DIR=~/ccache
export JAVA_HOME=/usr/lib/jvm/jdk
export ANT_HOME=~/android/ant
export ANDROID_SDK_HOME=~/android/sdk
export ANDROID_NDK_HOME=~/android/ndk
export AOSP_HOME=~/aosp
export
PATH=$HOME/bin:$JAVA_HOME/bin:$ANT_HOME/bin:$ANDROID_SDK_HOME/tools:$ANDROID_
SDK_HOME/platform-tools:$ANDROID_NDK_HOME:$PATH
$ cat .bashrc
# .bashrc
# User specific aliases and functions
HISTSIZE=1000
```

```
HISTFILESIZE=2000
# some more ls aliases
alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
```

เมื่อผู้ใช้เข้าสู่ระบบ Login โดยการใช้งานผ่านหน้าจอเทอร์มินัลซึ่งนิยมเรียกสั้นๆว่า tty (Teletype-writer) เมื่อหน้าจอเทอร์มินัลถูกสร้างขึ้น ตัวโปรแกรมสำหรับการเข้าสู่ระบบก็จะถูกเรียกขึ้นมาเพื่อแสดงข้อความให้ผู้ใช้กรอกชื่อผู้ใช้ (username) และรหัสผ่าน (password) เมื่อเข้าสู่ระบบได้สำเร็จก็จะทำการเรียกโปรแกรม shell เพื่ออ่านค่าพื้นฐานที่ตั้งไว้สำหรับสภาพแวดล้อมของระบบ (system environments) ดังขั้นตอนในรูปข้างล่าง



รูปที่ 1.7 ขั้นตอนการเข้าสู่ระบบปฏิบัติการผ่านเทอร์มินัล

นอกจากนั้นระบบจะมีการตั้งค่าตัวแปรระบบพื้นฐาน ซึ่งเป็นตัวแปรสำคัญที่จะพบเจอบ่อยได้แก่

ตาราง 1.4 แสดงตัวแปรระบบพื้นฐาน

ชื่อ	คำอธิบาย
USER	เรียกชื่อผู้ใช้งานที่ใช้ในการเข้าสู่ระบบ
UID	เรียกรหัสผู้ใช้งานที่ใช้ในการเข้าสู่ระบบ
HOME	เรียก Home Directory
PWD	เรียก Directory ที่กำลังใช้งาน
SHELL	เรียกชื่อ shell ที่กำลังใช้งาน
$	เรียก process id ที่กำลังใช้งาน
PPID	เรียก process id ที่ได้รับการสืบทอดมา
?	The exit code of the last command

ยกตัวอย่างการใช้คำสั่ง echo เพื่อแสดงค่าภายในตัวแปรระบบ

```
$ echo $SHELL
/bin/bash

$ echo -e " My Home directory is $HOME\n My username is $USER\n My
current directory is $PWD"
 My Home directory is /home/student
 My username is student
 My current directory is /home/student

$ echo $PPID
1927

$ ps -aux | grep 1927
student   1927  0.0  0.5 192052 11236 ?   Sl  08:15  0:33 gnome-terminal
```

env และ export

คำสั่ง export เป็นคำสั่งที่ผู้ใช้งานสามารถสร้างตัวแปรระบบเพิ่มเติมได้เอง โดยการระบุเข้าไปในไฟล์ .bashrc หรือพิมพ์คำสั่งการตั้งค่าได้โดยตรงใน terminal ดังตัวอย่างข้างล่าง

```
$ export my_project_codes=/home/student/embedded/sourcecode/
$ echo $my_project_codes
/home/student/embedded/sourcecode/
```

หากผู้ใช้ต้องการดูค่าตัวแปรระบบที่มีการใช้งานอยู่ สามารถใช้คำสั่ง env หรือ export -p ดังนี้

```
$ env
ORBIT_SOCKETDIR=/tmp/orbit-student
SSH_AGENT_PID=1383
TERM=xterm
```

```
SHELL=/bin/bash
...
USER=student
PATH=/home/student/bin:/usr/lib/jvm/jdk/bin:/home/student/android/ant/bin:/home/stud
ent/android/sdk/tools:/home/student/android/sdk/platform-tools:/home/student/android
/ndk:/home/student/bin:/home/student/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/u
sr/bin:/sbin:/bin:/usr/games
ANDROID_SDK_HOME=/home/student/android/sdk
PWD=/home/student
JAVA_HOME=/usr/lib/jvm/jdk
GDM_KEYBOARD_LAYOUT=us
LANG=en_US.utf8
GDM_LANG=en_US.utf8
MANDATORY_PATH=/usr/share/gconf/gnome.mandatory.path
GDMSESSION=gnome
SHLVL=1
HOME=/home/student
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
LOGNAME=student
DISPLAY=:0.0
AOSP_HOME=/home/student/aosp
LESSCLOSE=/usr/bin/lesspipe %s %s
COLORTERM=gnome-terminal
XAUTHORITY=/var/run/gdm/auth-for-student-9wtySF/database
_=/usr/bin/env

$ export -p
declare -x ANDROID_NDK_HOME="/home/student/android/ndk"
declare -x ANDROID_SDK_HOME="/home/student/android/sdk"
declare -x ANT_HOME="/home/student/android/ant"
...
declare -x USER="student"
declare -x USERNAME="student"
declare -x USE_CCACHE="0"
declare -x WINDOWID="14722574"
declare -x XAUTHORITY="/var/run/gdm/auth-for-student-9wtySF/database"
declare -x XDG_CONFIG_DIRS="/etc/xdg/xdg-gnome:/etc/xdg"
declare -x XDG_DATA_DIRS="/usr/share/gnome:/usr/local/share/:/usr/share/"
```

Exec

exec เป็นคำสั่งที่ใช้ในการเรียกโปรแกรมอื่นขึ้นมาแทนที่ shell เดิมที่ใช้งานอยู่ ดังตัวอย่างคำสั่งข้างล่างเป็นการใช้งานคำสั่ง exec โดยจะมีการสร้าง bash shell ขึ้นมาอีกตัวเพื่อให้เป็นโปรเซสลูก (**PID 30779**) ที่ทำงานภายใน bash shell หลักที่เป็นโปรเซสแม่ (**PID 19907**) จากนั้นมีการเรียกใช้งานคำสั่ง exec sh ซึ่งเป็นการเรียกตัว sh shell ขึ้นมาแทนที่ bash shell จะสังเกตได้จากหมายเลขโปรเซสเป็นหมายเลขเดียวกัน (**PID 19907**) ถ้าหากผู้ใช้งานต้องการออกจาก shell ที่ได้ทำการสร้างขึ้นมาจะต้องใช้คำสั่ง exit เพื่อเป็นจบการทำงานของโปรเซส (**PID 30779**) ที่ได้ทำการสร้างขึ้นมา

```
$ ps
  PID TTY          TIME CMD
19907 pts/1    00:00:00 bash     <---- หมายเลขโปรเซสแม่
30777 pts/1    00:00:00 ps
$ bash
```

```
$ ps
  PID TTY          TIME CMD
19907 pts/1    00:00:00 bash
30779 pts/1    00:00:00 bash     <---- หมายเลขโปรเซสลูก
30796 pts/1    00:00:00 ps
$ exec sh
$ ps
  PID TTY          TIME CMD
19907 pts/1    00:00:00 bash
30779 pts/1    00:00:00 sh
30800 pts/1    00:00:00 ps
$ exit
$ ps
  PID TTY          TIME CMD
19907 pts/1    00:00:00 bash
30802 pts/1    00:00:00 ps
```

การเรียกใช้งานคำสั่งภายในเชลล์ และภายนอกเชลล์

คำสั่งพื้นฐานภายใน shell เองนั้นประกอบไปด้วยคำสั่งหลักๆเพียงไม่กี่คำสั่งเท่านั้น สำหรับคำสั่งอื่นๆที่เป็นโปรแกรมประยุกต์หรือโปรแกรมอรรถประโยชน์จะถูกอ้างอิงมาจากชุดคำสั่งภายนอก shell (External Command) ทั้งสิ้น โดยคำสั่งต่างๆภายนอก shell จะเป็นไฟล์โปรแกรมที่ถูกเก็บอยู่ในระบบไฟล์ (root filesystem) เช่น /usr/bin เป็นต้น ดังนั้นการเข้าถึงไฟล์โปรแกรมเหล่านี้จำเป็นต้องรู้ตำแหน่งไดเรกทอรีที่เก็บไฟล์โปรแกรมเหล่านั้นอยู่ เมื่อผู้ใช้พิมพ์คำสั่ง โปรแกรม shell จะเข้าไปอ่านไดเรกทอรีที่ถูกตั้งค่าไว้ในตัวแปรสภาพแวดล้อมของระบบ (Environment Variables) ที่ชื่อว่า $PATH โดยในการเข้าถึงคำสั่งนั้น shell จะทำการตรวจสอบคำสั่งตั้งแต่ไดเรกทอรีแรกจนถึงไดเรกทอรีสุดท้าย (โดยมี ':' คั่นระหว่างไดเรกทอรี) ภายในตัวแปร PATH ดังนั้นกรณีที่ผู้ใช้งานต้องการทราบว่าคำสั่งที่ใช้นั้นถูกเก็บอยู่ในไดเรกทอรีใด ก็สามารถตรวจสอบได้ด้วยคำสั่ง which ดังนี้

```
$ which ls cat date ifconfig uname whoami ps find grep echo set
/bin/ls
/bin/cat
/bin/date
/sbin/ifconfig
/bin/uname
/usr/bin/whoami
/bin/ps
/usr/bin/find
/bin/grep
/bin/echo
```

ถ้าต้องการทราบรายละเอียดที่สำคัญบางอย่าง เช่น เป็นคำสั่งที่อยู่ภายใน shell หรือ เป็นคำสั่งที่เกิดจากการทำ alias (การทำนามแฝงให้กับคำสั่งที่จำยาก หรือซับซ้อน ให้กลายเป็นคำสั่งที่เราจำได้ง่ายขึ้น หรือ

เรียกได้สั้นๆ เช่น `alias ls=`ls -al --color=auto`) ก็สามารถใช้คำสั่ง type เพื่อตรวจสอบได้ดังนี้

```
$ type ls cat date ifconfig uname whoami ps find grep echo set
ls is aliased to `ls -al --color=auto'
cat is hashed (/bin/cat)
date is /bin/date
ifconfig is /sbin/ifconfig
uname is /bin/uname
whoami is /usr/bin/whoami
ps is hashed (/bin/ps)
find is /usr/bin/find
grep is aliased to `grep --color=auto'
echo is a shell builtin
set is a shell builtin
```

สังเกตว่าไดเรกทอรีส่วนใหญ่ที่อยู่ใน PATH จะลงท้ายด้วย bin ด้วยเหตุผลการวางโครงสร้างของระบบไฟล์ภายในระบบปฏิบัติการลีนุกซ์ ให้เก็บไฟล์โปรแกรมไว้ในไดเรกทอรี /bin และ /usr/bin

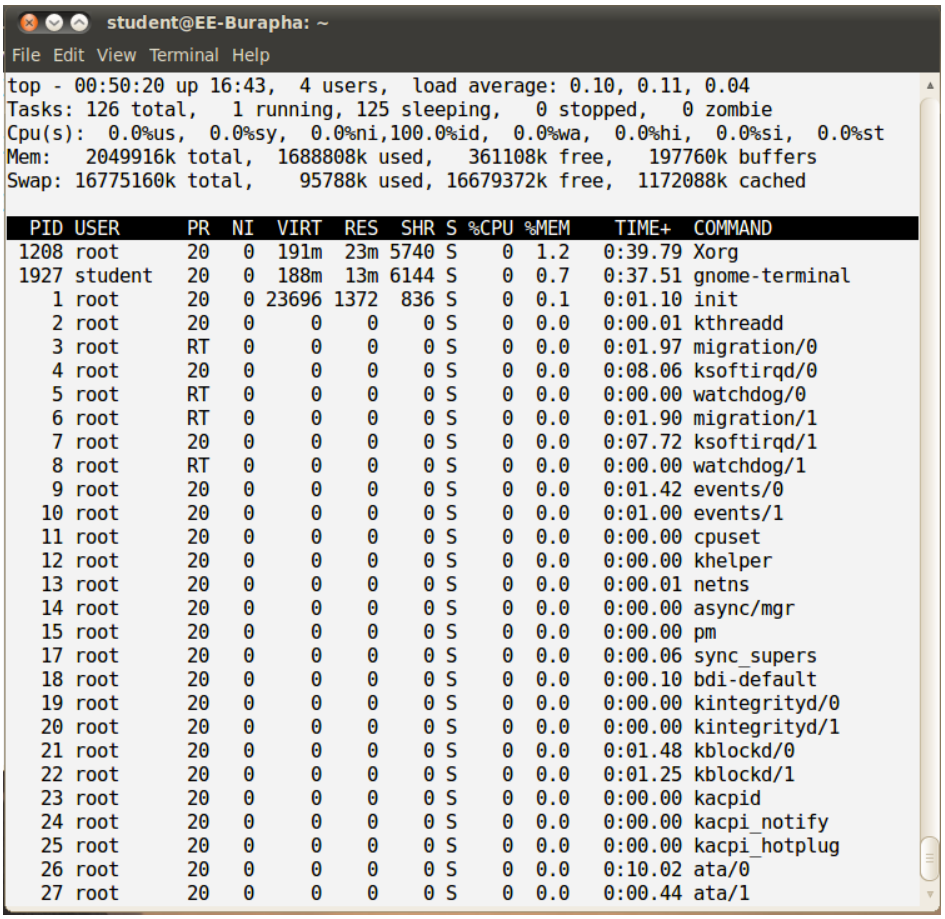
คำสั่งพื้นฐานสำหรับนักพัฒนาบนระบบสมองกลฝังตัว

คำสั่งตรวจสอบทรัพยากรระบบ

การใช้งานระบบปฏิบัติการลีนุกซ์ภายใต้ระบบสมองกลฝังตัวที่มีทรัพยากรจำกัด ไม่ว่าจะเป็นความเร็วของหน่วยประมวลผลกลาง ขนาดของหน่วยความจำ หรือขนาดพื้นที่จัดเก็บข้อมูล นักพัฒนาจำเป็นจะต้องศึกษาเกี่ยวกับวิธีการติดตามการใช้ทรัพยากรของระบบภายในสมองกลฝังตัว และสามารถประเมินเพื่อหลีกเลี่ยงความผิดพลาดที่อาจจะเกิดขึ้นในกรณีที่ทรัพยากรถูกใช้มากเกินไปเพื่อทำให้ระบบสามารถทำงานได้อย่างมีประสิทธิภาพและมีความเสถียรภาพสูงสุด

top

เป็นคำสั่งแรกๆที่ผู้ดูแลหรือนักพัฒนาระบบจำเป็นต้องทำความรู้จัก เพราะเป็นคำสั่งที่ใช้แสดงสถานะการใช้งานทรัพยากรของระบบ รวมทั้งแสดงรายการโปรเซสที่ถูกเรียกใช้งานอยู่ โดยเรียงตามการครอบครองทรัพยากร ซึ่งเชื่อมโยงกับ virtual filesystem (VFS) ไฟล์ /proc/loadavg ดังตัวอย่างข้างล่าง



รูปที่ 1.8 แสดงผลลัพธ์คำสั่ง top

จากรูปข้างบน จะมีส่วนรายละเอียดสำคัญ ได้แก่

Uptime

```
top - 00:50:20 up 16:43,  4 users,
```

หมายถึง เวลาปัจจุบัน ระยะเวลารวมที่ระบบถูกเปิด และจำนวนผู้ใช้งานที่อยู่ในระบบ

load average

```
load average: 0.10, 0.11, 0.04
```

หมายถึง ข้อมูลที่แสดงการเปรียบเทียบการใช้งานหน่วยประมวลผลกลาง (CPU) ที่ใช้งานอยู่ กับความสามารถในการรองรับการใช้งาน โดยแสดงออกมา 3 ค่า ซึ่งได้แก่ ค่า 0.10 ณ ช่วงเวลา 1 นาทีก่อนหน้า, ค่า 0.11 ณ ช่วงเวลา 5 นาทีก่อนหน้า และ ค่า 0.04 ณ ช่วงเวลา 15 นาทีก่อนหน้า

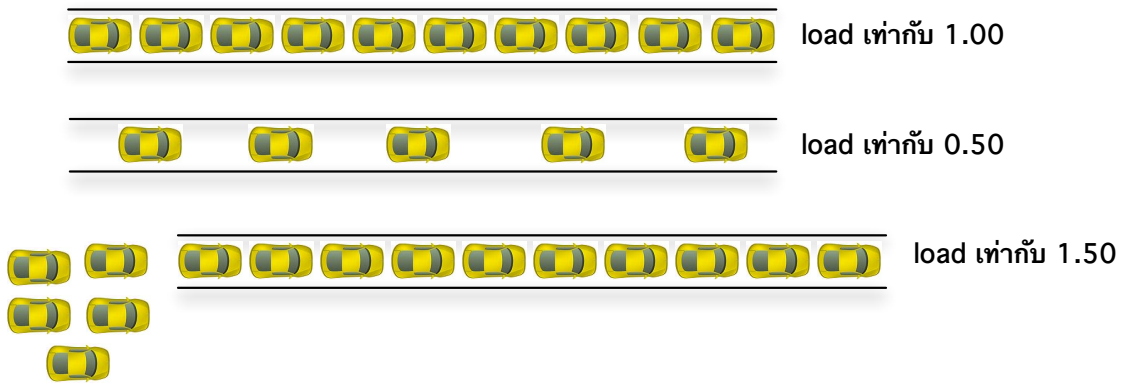
นอกจากนั้นสามารถใช้คำสั่ง uptime เมื่อต้องการดูค่าทั้งสองอย่างนี้โดยเฉพาะ (uptime และ load average) ด้วยคำสั่งข้างล่างนี้

```
$ uptime
 01:24:59 up 17:18,  4 users,  load average: 0.08, 0.08, 0.02
$ watch -n 1 uptime  <--- รันคำสั่ง uptime ทุกๆ 1 วินาที
Every 1.0s: uptime                      Tue Aug 20 01:26:37 2013
01:26:37 up 17:19,  4 users,  load average: 0.01, 0.05, 0.01
```

อย่างไรก็ตามการตีความค่า load average นั้นก็ยังมีนักพัฒนาจำนวนไม่น้อยที่ไม่เข้าใจความหมายที่แท้จริงของค่านี้ เนื่องจากค่าที่ได้ไม่ใช่จำนวนเปอร์เซ็นต์ บางครั้งค่าตัวเลขก็เกิน 1.0 ดังนั้นเพื่อให้เห็นภาพนักพัฒนาลองทำการจินตนาการตามว่าเครื่องที่มีหน่วยประมวลผลกลางแบบคอร์เดียว (single-core CPU) เปรียบได้กับถนนเดินรถทางเดียว (one-way lane) บนสะพานข้ามแม่น้ำ และสมมติให้ระบบปฏิบัติการลีนุกซ์เป็นตำรวจจราจรคอยจัดการการเดินรถบนสะพานแห่งนี้ ซึ่งในบางช่วงเวลาของแต่ละวันปริมาณรถที่วิ่งขึ้นสะพานอาจจะมากน้อยไม่เท่ากันขึ้นอยู่กับช่วงเวลาเร่งด่วน (rush hour) ดังนั้นตำรวจจราจรจำเป็นจะต้องมีการประเมินสภาพการจราจรในรูปแบบของระบบตัวเลข (numbering system) เช่น

- 0.00 หมายถึง ไม่มีการจราจรบนถนน
- 0.50 หมายถึง การจราจรคล่องตัว
- 1.00 หมายถึงการจราจรหนาแน่นแต่เคลื่อนตัวได้
- ถ้า 2.00 อาจหมายถึง รถเต็มตลอดถนนบนสะพาน และมีจำนวนรถที่ปริมาณเท่ากันรออยู่ต้นสะพาน
- ถ้า 3.00 อาจหมายถึง รถเต็มตลอดถนนบนสะพาน และมีจำนวนรถที่ปริมาณ 2 เท่า รออยู่ต้นสะพาน เป็นต้น

รูปข้างล่างแสดงปริมาณรถบนถนนทางเดียวบนสะพาน และรถที่ต้องรออยู่ที่ต้นสะพาน



รูปที่ 1.9 แสดงการเปรียบเทียบสภาพจราจรกับ Load

เปรียบเทียบรถที่กำลังเคลื่อนที่ข้ามสะพาน ก็คือโปรเซสที่จะเข้ามาใช้เวลาของหน่วยประมวลผลกลางให้ทำงานให้ (slice of CPU time) หรือการรอคิวเพื่อใช้หน่วยประมวลผลกลาง โดยระบบปฏิบัติการลีนุกซ์ จะมีการอ้างถึงความยาวของคิว (run-queue length) ซึ่งมาจากผลรวมของจำนวนโปรเซสบวกด้วยจำนวนของโปรเซสที่กำลังรอคิว ซึ่งค่าที่เป็นอุดมคติที่ดีสุดคือ 1.00 ซึ่งหมายถึงสะพานสามารถรับปริมาณรถได้เต็มที่ และยังคงเคลื่อนตัวได้ดี เปรียบได้กับ จำนวนโปรเซสรออยู่เต็มคิวพอดี และหน่วยประมวณผลกลางก็ยังสามารถทำงานให้ได้อย่างต่อเนื่อง ทางเทคนิคจะเรียกค่านี้ว่า ค่า “Utilization”

ดังนั้นการประเมินสภาพภาระโดยเฉลี่ย (load average) ของหน่วยประมวลผลกลางนั้น โดยหลักการทั่วไป (Rule of Thumb) แล้ว ควรไม่เกิน 0.70 ซึ่งนักพัฒนาจะต้องรีบตรวจสอบภาระโดยเฉลี่ยในกรณีที่เกิน 0.70 ที่เกิดต่อเนื่องมาสักระยะนึง เพราะอาจจะเกิดปัญหาใหญ่ตามมาได้

คำสั่งตรวจสอบการใช้หน่วยความจำระบบ

free, pmap

หน่วยความจำ (Memory) เป็นทรัพยากรสำคัญในระบบสมองกลฝังตัว ดังนั้นการตรวจสอบว่ามีการใช้งานหน่วยความจำมากน้อยเพียงใดจึงมีความสำคัญสูง ในระบบปฏิบัติการ Embedded Linux นั้นการตรวจสอบและแสดงผลการใช้งานหน่วยความจำได้มีการกำหนดให้ใช้คำสั่ง free ซึ่งเชื่อมโยงกับ virtual filesystem (VFS) ไฟล์ /proc/meminfo สำหรับการแสดงผลสามารถเลือกการแสดงผลโดยการใส่ตัวเลือกเข้าไปได้ดังตัวอย่างต่อไปนี้ โดยที่ -k แสดงในรูปแบบกิโลไบต์ , -m แสดงในรูปแบบเมกาไบต์, -g แสดงผลในรูปแบบกิกาไบต์

```
$ free -m -t
             total       used       free     shared    buffers     cached
Mem:          2001       1649        351          0        193       1145
-/+ buffers/cache:        311       1690
Swap:        16381         93      16288
Total:       18383       1743      16640
```

นอกจากนั้นการดูรายละเอียดของการเข้าใช้จับจองหน่วยความจำและการอ้างอิงกับไลบรารีต่างๆของโปรเซส ก็เป็นสิ่งจำเป็นสำหรับนักพัฒนาที่จะต้องรู้ว่าโปรแกรมที่ได้พัฒนามีการทำงานเป็นอย่างไรเมื่อถูกโหลดขึ้นสู่หน่วยความจำ ดังนั้นคำสั่ง pmap จึงถูกนำมาช่วยวิเคราะห์ได้ดี ดังตัวอย่างคำสั่งข้างล่างนี้

สร้างไฟล์ hello.c โดยมีโค้ดภาษาซีดังนี้

```
$ cat hello.c

#include <stdio.h>
int main()

{
  printf("Hello World!\n");
  while (1)
  {

  }
 return 0;
}
```

```
$ gcc -o hello hello.c
$ ./hello &
[1] 11958
```

```
Hello World!

$ pmap 11958
11958:   ./hello
0000000000400000      4K r-x--  /home/student/hello
0000000000600000      4K r----  /home/student/hello
0000000000601000      4K rw---  /home/student/hello
00007f34087ca000   1512K r-x--  /lib/libc-2.11.1.so
00007f3408944000   2044K -----  /lib/libc-2.11.1.so
00007f3408b43000     16K r----  /lib/libc-2.11.1.so
00007f3408b47000      4K rw---  /lib/libc-2.11.1.so
00007f3408b48000     20K rw---    [ anon ]
00007f3408b4d000    128K r-x--  /lib/ld-2.11.1.so
00007f3408d52000     12K rw---    [ anon ]
00007f3408d69000     12K rw---    [ anon ]
00007f3408d6c000      4K r----  /lib/ld-2.11.1.so
00007f3408d6d000      4K rw---  /lib/ld-2.11.1.so
00007f3408d6e000      4K rw---    [ anon ]
00007fff77ad1000     84K rw---    [ stack ]
00007fff77bff000      4K r-x--    [ anon ]
ffffffffff600000      4K r-x--    [ anon ]
 total             3864K
```

คำสั่งตรวจสอบการใช้พื้นที่สำหรับเก็บข้อมูล

df, du

การเก็บข้อมูลในระบบสมองกลฝังตัว อุปกรณ์ที่ถูกใช้เป็นตัวเก็บข้อมูลจะแตกต่างกับเครื่องคอมพิวเตอร์ทั่วไป เนื่องจากไม่มีฮาร์ดดิสก์ แต่จะใช้อุปกรณ์เก็บข้อมูลชนิดแฟลชแทน เช่น NAND Flash, USB Storage หรือ MMC/SD Card เป็นต้น ซึ่งมีขนาดความจุน้อยมากเมื่อเทียบกับฮาร์ดดิสก์ทั่วไป ดังนั้นการเฝ้าตรวจสอบพื้นที่ของตัวเก็บข้อมูลจะต้องให้ความสำคัญมากเป็นพิเศษ คำสั่งที่มีการใช้งานกันอย่างแพร่หลายในระบบปฏิบัติการลีนุกซ์ ที่สามารถแสดงผลได้ทั้งจำนวนพื้นที่ที่มีการใช้งานไปแล้วในระบบ และพื้นที่ว่างที่สามารถใช้งาน ได้แก่คำสั่ง df (disk free) ดังตัวอย่างต่อไปนี้

```
$ df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda1              99G   62G   37G  63% /
none                  997M  276K  997M   1% /dev
none                 1001M   92K 1001M   1% /dev/shm
none                 1001M   88K 1001M   1% /var/run
none                 1001M     0 1001M   0% /var/lock
none                 1001M     0 1001M   0% /lib/init/rw
```

คำสั่ง du (disk usage) หมายถึงคำสั่งที่ใช้ในการตรวจสอบขนาดการใช้งานไดเรกทอรีที่ชี้อยู่ (mount point) รวมถึงไดเรกทอรีย่อยๆลงไปจากตำแหน่งปัจจุบัน และการใช้งานคำสั่ง du จะมีลักษณะที่คล้าย

กับการใช้งานคำสั่ง df นั่นคือคำสั่ง du สามารถที่จะเติมตัวเลือก -h เข้าไปท้ายคำสั่งเพื่อเป็นการแปลงข้อมูลให้อยู่ในลักษณะที่บุคคลทั่วไปสามารถอ่านได้ง่ายดังตัวอย่างต่อไปนี้

```
$ du -h busybox-1.21.0/
44K     busybox-1.21.0/libpwdgrp
108K     busybox-1.21.0/findutils
536K     busybox-1.21.0/miscutils
24K     busybox-1.21.0/coreutils/libcoreutils
684K     busybox-1.21.0/coreutils
.
.
.
116K     busybox-1.21.0/archival/libarchive/unxz
484K     busybox-1.21.0/archival/libarchive
824K     busybox-1.21.0/archival
16M     busybox-1.21.0/
```

นอกจากตัวเลือก -h แล้ว สามารถใช้อีกตัวเลือกคือ -s ซึ่งจะแสดงผลรวมที่ได้จากไดเรกทอรีต่างๆ เข้ามาอยู่ในบรรทัดเดียว ดังตัวอย่างข้างล่าง

```
$ du -sh busybox-1.21.0/
16M     busybox-1.21.0/
```

คำสั่งสำหรับการบริหารจัดการโปรเซส

ในการบริหารจัดการโปรเซสที่เกิดขึ้นในระหว่างเครื่องทำงานอยู่ จะเป็นหน้าที่หลักของระบบปฏิบัติการลีนุกซ์ ที่จะต้องจัดลำดับการร้องขอจากแต่ละโปรเซสเพื่อเข้าใช้ทรัพยากรของระบบ ดังนั้นโปรเซสทุกตัวจะต้องมีหมายเลขประจำของตัวเอง ซึ่งจะถูกกำหนดมาให้จากระบบปฏิบัติการเอง เรียกว่า PID (process id) แต่อย่างไรก็ตามผู้ใช้หรือนักพัฒนาก็สามารถควบคุมและบริหารจัดการกับโปรเซสที่เกิดจากการเรียกด้วยตัวผู้ใช้เอง โดยสามารถบริหารจัดการแยกเป็นกลุ่มได้แก่

- การติดตามดูสถานะของโปรเซสโดยรวมด้วยคำสั่ง jobs
- เลือกดูเฉพาะโปรเซสที่ต้องการ และค่าต่างที่ต้องการ
- การส่งสัญญาณ (signals) แบบต่างไปยังโปรเซส

jobs, ps

คำสั่งที่เกี่ยวกับการติดตามดูสถานะของโปรเซสโดยรวมนั้น หลักมีอยู่ 2 คำสั่งคือ jobs และ ps (process snapshot) โดยคำสั่ง jobs นั้นจะแสดงหมายเลขของงานที่ถูกเรียกขึ้นมาตามลำดับ และมีหมายเลขโปรเซส (PID) อยู่ด้วย นอกจากนั้นคำสั่งนี้จะบอกสถานะของโปรเซสทั้งหมดว่าเป็นอย่างไรบ้าง เช่น กำลังทำงาน (running) ถูกหยุดการทำงาน (stopped) ถูกสั่งให้สิ้นสุดการทำงาน (killed) เป็นต้น ดังตัวอย่างคำสั่งข้างล่าง

```

```

```
$ jobs -l
[1]  11958 Running                 ./hello &
[2]- 12458 Stopped                 emulator &  (wd: ~/aosp)
[3]+ 12794 Killed                  gcalctool
```

สำหรับคำสั่ง ps มีหน้าที่ในการแสดงรายละเอียดโปรเซส ตัวอย่างเช่น ถ้ารับรายการ PID มาจากคำสั่ง jobs ที่ระบุตัวเลือก -p (การลำดับ process group leader) ก็จะแสดงรายละเอียดของหมายเลขโปรเซสเหล่านั้น ดังตัวอย่างข้างล่าง

```
$ jobs -p
11958
12458
12818
$ ps $(jobs -p)
  PID TTY      STAT   TIME COMMAND
11958 pts/0    R     63:41 ./hello
12458 pts/0    Sl    2:25 /home/student/android/sdk/tools/emulator64-x86
12818 pts/0    S     0:00 gcalctool
```

ในกรณีที่ผู้ดูแลระบบต้องการทราบถึงสถานะโปรเซสทั้งหมดที่ทำงานอยู่ในระบบ ผู้ดูแลระบบจำเป็นต้องมีการเพิ่มตัวเลือก -ef ต่อท้ายคำสั่ง ps เพื่อแสดงผลโปรเซสทั้งหมดที่ทำงานอยู่ในระบบในขณะนั้น ดังตัวอย่างข้างล่าง

```
$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Aug19 ?        00:00:01 /sbin/init
root         2     0  0 Aug19 ?        00:00:00 [kthreadd]
root         3     2  0 Aug19 ?        00:00:01 [migration/0]
root         4     2  0 Aug19 ?        00:00:08 [ksoftirqd/0]
root         5     2  0 Aug19 ?        00:00:00 [watchdog/0]
root         6     2  0 Aug19 ?        00:00:01 [migration/1]
root         7     2  0 Aug19 ?        00:00:08 [ksoftirqd/1]
root         8     2  0 Aug19 ?        00:00:00 [watchdog/1]
root        10     2  0 Aug19 ?        00:00:01 [events/1]
root        11     2  0 Aug19 ?        00:00:00 [cpuset]
root        12     2  0 Aug19 ?        00:00:00 [khelper]
root        13     2  0 Aug19 ?        00:00:00 [netns]
root        14     2  0 Aug19 ?        00:00:00 [async/mgr]
root        18     2  0 Aug19 ?        00:00:00 [bdi-default]
root        19     2  0 Aug19 ?        00:00:00 [kintegrityd/0]
root        20     2  0 Aug19 ?        00:00:00 [kintegrityd/1]
root        21     2  0 Aug19 ?        00:00:01 [kblockd/0]
root       306     2  0 Aug19 ?        00:00:05 [jbd2/sda1-8]
root      1268     1  0 Aug19 tty3     00:00:00 /sbin/getty -8 38400 tty3
root      1272     1  0 Aug19 tty6     00:00:00 /sbin/getty -8 38400 tty6
root      1275     1  0 Aug19 ?        00:00:07 /usr/sbin/irqbalance
student  11825 11824  0 05:21 ?        00:00:00 gnome-pty-helper
student  11826 11824  0 05:21 pts/0    00:00:00 bash
student  11958 11826 99 05:22 pts/0    01:24:29 ./hello
student  12458 11826  5 05:42 pts/0    00:03:43 /home/student/android/sdk/tools/
student  12515     1  0 05:43 pts/0    00:00:00 adb fork-server server
student  12818 11826  0 06:26 pts/0    00:00:00 gcalctool
```

```
root     12856     2  0 06:32 ?        00:00:00 [flush-8:0]
student  12929 11826  0 06:47 pts/0    00:00:00 ps -ef
```

จากผลลัพธ์ที่ได้ จะสามารถทราบได้ว่ามีโปรเซสไหนบ้างที่เกิดจากการเรียกของผู้ใช้คนใด (UID) มีหมายเลขโปรเซสอะไร (PID) และเกิดขึ้นจากการเรียกของโปรเซสใด (PPID) เป็นต้น รายละเอียดดังอธิบายในตารางต่อไปนี้

ตาราง 1.5 รายละเอียดของคำสั่ง ps

คอลัมน์	ความหมาย
UID	แสดง user เจ้าของโปรเซส
PID	แสดงหมายเลขโปรเซส (Process ID)
PPID	แสดงหมายเลขโปรเซสหลักที่ได้รับการสืบทอดมา (Parent Process ID)
C	แสดงการใช้งาน CPU
STIME	แสดงเวลาที่โปรเซสเริ่มต้น
TTY	แสดงรหัสเทอมินอลที่โปรเซสทำงานอยู่
TIME	เวลาโดยรวมที่เข้าไปใช้งาน CPU
CMD	ชื่อคำสั่งที่ใช้ในการสร้างโปรเซสขึ้นมา

เมื่อต้องการดูเฉพาะโปรเซสที่เกิดขึ้นจากการเรียกใช้ใน bash shell สามารถใช้ตัวเลือก เช่น -f (full), -j (jobs), -l (long), --forest, --sort ดังตัวอย่างการใช้งานข้างล่าง

```
$ ps
  PID TTY          TIME CMD
11826 pts/0    00:00:00 bash
11958 pts/0    01:19:16 hello
12458 pts/0    00:03:31 emulator64-x86
12515 pts/0    00:00:00 adb
12818 pts/0    00:00:00 gcalctool
12905 pts/0    00:00:00 ps

$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
student  11826 11824  0 05:21 pts/0    00:00:00 bash
student  11958 11826 99 05:22 pts/0    01:19:17 ./hello
student  12458 11826  5 05:42 pts/0    00:03:31 /home/student/android/sdk/
tools/
student  12515     1  0 05:43 pts/0    00:00:00 adb fork-server server
student  12818 11826  0 06:26 pts/0    00:00:00 gcalctool
student  12906 11826  0 06:42 pts/0    00:00:00 ps -f

$ ps -j --forest
  PID  PGID   SID TTY          TIME CMD
```

```
11826 11826 11826 pts/0    00:00:00 bash
11958 11958 11826 pts/0    01:19:31  \_ hello
12458 12458 11826 pts/0    00:03:31  \_ emulator64-x86
12818 12818 11826 pts/0    00:00:00  \_ gcalctool
12908 12908 11826 pts/0    00:00:00  \_ ps
12515 12514 11826 pts/0    00:00:00 adb

$ ps -aj --sort sid,comm
  PID  PGID   SID TTY          TIME CMD
12515 12514 11826 pts/0    00:00:00 adb
12458 12458 11826 pts/0    00:04:26 emulator64-x86
12818 12818 11826 pts/0    00:00:00 gcalctool
11958 11958 11826 pts/0    01:42:27 hello
13020 13020 11826 pts/0    00:00:00 ps
```

การหยุดการทำงานของโปรเซส (Process Killing) ในระบบปฏิบัติการลีนุกซ์นั้นจะใช้หลักการส่งข้อความของระบบในระดับล่าง (low-level system message) ที่เรียกกันว่า สัญญาณ (Signal) โดยสัญญาณจะถูกส่งไปขัดจังหวะโปรเซส เพื่อบอกกับโปรเซสตามวัตถุประสงค์ของสัญญาณนั้น ซึ่งในระบบปฏิบัติการลีนุกซ์จะมีชนิดของสัญญาณอยู่ทั้งสิ้น 64 ชนิด สามารถดูรายละเอียดได้จากการพิมพ์คำสั่ง kill -l ดังข้างล่างนี้

$ kill -l

```
1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL      10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGCONT     24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

ตัวอย่างสัญญาณที่สำคัญและใช้งานบ่อย ตัวอย่างเช่น

- สัญญาณหมายเลข 1 (SIGHUP) คือ Hang Up Signal เป็นสัญญาณที่ถูกส่งออกไปในขณะที่ terminal ถูกปิดลงเพื่อทำให้โปรเซสที่ถูกเรียกภายใต้ terminal ถูกปิดลงตามด้วย
- สัญญาณหมายเลข 3 (SIGQUIT) คือ Quit Signal เป็นสัญญาณที่ถูกส่งออก ในขณะที่ผู้ใช้งานได้มีการสั่งให้ โปรเซสนั้นปิดตัวลง
- สัญญาณหมายเลข 6 (SIGABRT) คือ Abort Signal เป็นสัญญาณที่ถูกส่งออกมาจากโปรเซสที่ต้องการปิดตัวเองลง
- สัญญาณหมายเลข 9 (SIGKILL) คือ Kill Signal ลักษณะของ signal ประเภทนี้จะคล้ายกับการดึงสายไฟออกจากเครื่องคอมพิวเตอร์ กล่าวคือเป็นการสั่งให้โปรเซสปิดตัวลงทันทีไม่ว่าจะทำงานอะไรอยู่ก็ตาม
- สัญญาณหมายเลข 18 (SIGCONT) คือ Continue Signal เป็นสัญญาณที่สั่งให้โปรเซสนั้นกลับมาทำงานปกติเช่นเดิม เช่นเดียวกันการสั่งให้โปรเซสกลับมาทำงานเบื้องหน้า (foreground)
- สัญญาณหมายเลข 20 (SIGTSTP) คือ การส่ง Stop Signal ด้วย Keyboard (Ctrl+Z) เป็นสัญญาณที่สั่งให้โปรเซสนั้นถูกทำให้ไปทำงานเบื้องหลัง (background) เช่นเดียวกับการกดปุ่ม Ctrl+Z บนคีย์บอร์ด

ตัวอย่างการส่งสัญญาณผ่านคีย์บอร์ด

เป็นวิธีการหนึ่งของการส่งสัญญาณให้กับโปรเซส โดยเมื่อมีกดคำสั่งบนคีย์บอร์ดดังนี้

Ctrl-C

ส่งสัญญาณ **INT (SIGINT)** ไปยังโปรเซสที่ทำงานอยู่ยุติการทำงานทันที

Ctrl-Z

ส่งสัญญาณ **TSTP (SIGTSTP)** ไปยังโปรเซสที่ทำงานอยู่ให้หยุดชั่วคราว

Ctrl-

ส่งสัญญาณ **ABRT (SIGABRT)** ไปยังโปรเซสที่ทำงานอยู่ให้ยุติการทำงานของโปรเซสนั้นทันที เหมือน Ctrl-C แต่เป็นคำสั่งที่ดีกว่าคือสามารถแก้ไขได้

```
$ jobs -l
[1]  11958 Running                 ./hello &
[2]- 12458 Running                 emulator &  (wd: ~/aosp)
[3]+ 12818 Running                 gcalctool &

$ kill -s SIGTSTP 12818
$ jobs -l
[1]  11958 Running                 ./hello &
[2]- 12458 Running                 emulator &  (wd: ~/aosp)
[3]+ 12818 Stopped                 gcalctool

$ kill -s SIGCONT 12818
$ jobs -l
[1]  11958 Running                 ./hello &
[2]- 12458 Running                 emulator &  (wd: ~/aosp)
```

```
[3]+ 12818 Running                 gcalctool &

$ kill -s SIGKILL 12818
$ jobs -l
[1]  11958 Running                 ./hello &
[2]- 12458 Running                 emulator &  (wd: ~/aosp)
[3]+ 12818 Killed                  gcalctool

$ kill -s SIGTERM 11958
$ jobs -l
[1]  11958 Terminated              ./hello
[2]- 12458 Running                 emulator &  (wd: ~/aosp)
```

การอ่านสถานะของทรัพยากรระบบจากไดเรกทอรี /PROC

ไดเรกทอรี /proc เปรียบเสมือนหน้าต่างของลีนุกซ์คอร์เนล (Kernel Windows) ซึ่งเป็นไดเรกทอรีพิเศษที่เป็นระบบไฟล์เสมือน (virtual filesystem) ที่สามารถเข้าถึงสถานะการทำงานของอุปกรณ์ในระดับฮาร์ดแวร์ของเครื่อง และสถานะข้อมูลต่างๆของระบบโดยค่าทั้งหมดจะถูกเก็บอยู่ในหน่วยความจำ

เมื่อใช้คำสั่ง `ls /proc` จะพบกลุ่มของไฟล์จำนวนมาก รวมทั้งไดเรกทอรีที่เป็นตัวเลข ซึ่งไดเรกทอรีเหล่านี้คือสถานที่เก็บข้อมูลของระบบที่กำลังทำงานอยู่และถูกจะอ้างอิงจากหมายเลขโปรเซสนั่นเอง นอกจากนั้นแล้วในไดเรกทอรี /proc ยังมีไฟล์พิเศษที่แทนฮาร์แวร์ต่างๆในระบบ ได้แก่

ตาราง 1.6 ตัวอย่างไฟล์สำคัญในไดเรกทอรี /proc

ไฟล์	คำอธิบาย
/proc/modules	dynamically loaded modules
/proc/devices	registered character and block major numbers
/proc/iomem	on-system physical RAM and bus device addresses
/proc/ioports	on-system I/O port addresses (especially for x86 systems)
/proc/interrupts	registered interrupt request numbers
/proc/softirqs	registered soft IRQs
/proc/kallsyms	running kernel symbols, including from loaded modules
/proc/partitions	currently connected block devices and their partitions
/proc/filesystems	currently active filesystem drivers
/proc/swaps	currently active swaps
/proc/cpuinfo	information about the CPU(s) on the system
/proc/meminfo	information about the memory on the system, viz., RAM, swap

ตัวอย่างการอ่านค่าสถานะของระบบจากไฟล์ภายในไดเรกทอรี /proc/

```
$ cat /proc/partitions
major minor  #blocks  name

   8        0  104857600 sda
   8        1  104855552 sda1
   8       16   16777216 sdb
   8       17   16775168 sdb1
$ cat /proc/interrupts
           CPU0       CPU1
  0:        286          0   IO-APIC-edge      timer
  1:      14606        732   IO-APIC-edge      i8042
  3:          1          0   IO-APIC-edge
  4:      32225        881   IO-APIC-edge
  6:          3          0   IO-APIC-edge      floppy
...
TRM:          0          0   Thermal event interrupts
THR:          0          0   Threshold APIC interrupts
MCE:          0          0   Machine check exceptions
MCP:        259        259   Machine check polls

$ cat /proc/cpuinfo
processor     : 0
vendor_id     : GenuineIntel
cpu family     : 6
model          : 58
model name     : Intel(R) Core(TM) i7-3615QM CPU @ 2.30GHz
stepping     : 9
cpu MHz          : 2293.993
cache size     : 6144 KB
fpu          : yes
fpu_exception     : yes
cpuid level     : 13
wp          : yes
...
processor     : 1
vendor_id     : GenuineIntel
cpu family     : 6
model          : 58
model name     : Intel(R) Core(TM) i7-3615QM CPU @ 2.30GHz
stepping     : 9
cpu MHz          : 2293.993
cache size     : 6144 KB
fpu          : yes
fpu_exception     : yes
cpuid level     : 13
wp          : yes
flags          : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pat pse36 clflush dts mmx fxsr sse sse2 ss syscall nx rdtscp lm con-
stant_tsc arch_perfmon pebs bts rep_good xtopology tsc_reliable nonstop_tsc
aperfmperf pni pclmulqdq vmx ssse3 cx16 sse4_1 sse4_2 x2apic popcnt aes xsave
avx hypervisor lahf_lm ida arat tpr_shadow vnmi ept vpid
bogomips     : 4587.98
clflush size     : 64
cache_alignment     : 64
```

```
address sizes      : 40 bits physical, 48 bits virtual
power management:
```

คำสั่งเกี่ยวกับการเปิดอ่านข้อมูลภายในไฟล์

Cat, less, od, และ split

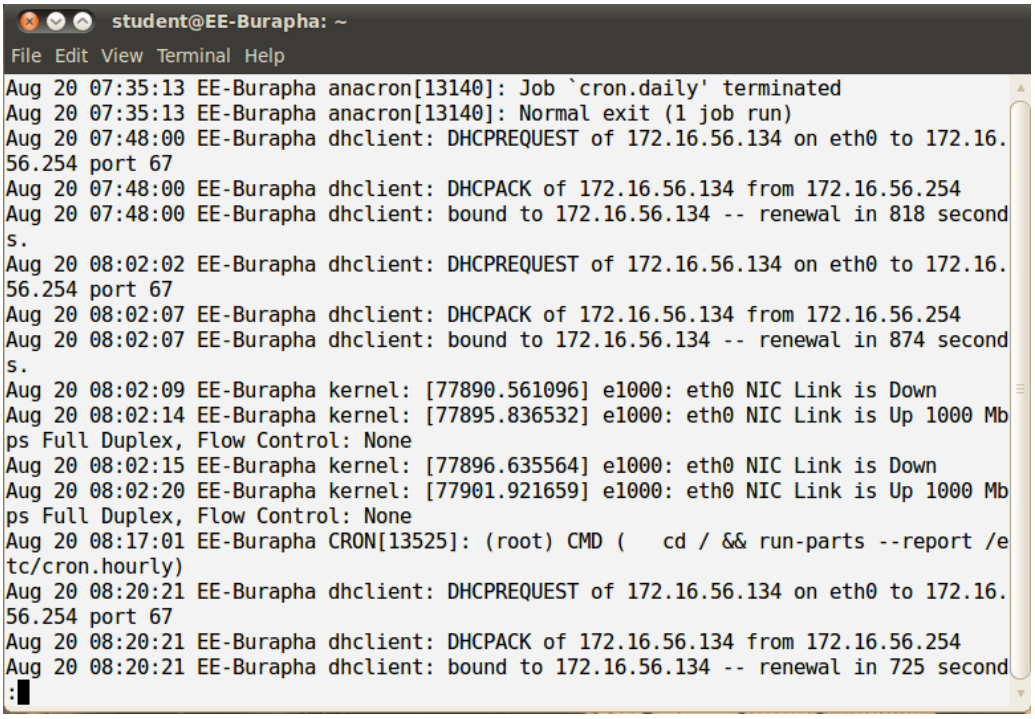
คำสั่ง cat คือคำสั่งที่ใช้แสดงข้อมูลที่อยู่ในไฟล์ออกมาแสดงครั้งเดียวพร้อมกันทั้งหมด ในบางครั้งก็ใช้ในการรวมไฟล์หลายไฟล์เข้าด้วยกันมาเป็นไฟล์เดียว แต่อย่างไรก็ตามถ้าไฟล์ที่ต้องการเปิดอ่านข้อมูลมีความยาวมาก ตัวอย่างเช่นไฟล์ที่เก็บ log ของระบบ ถ้าใช้คำสั่ง cat เพื่อเปิดไฟล์ log ย่อมไม่เป็นผลดีกับผู้อ่านเองเพราะจะแสดงข้อมูลออกมาอย่างรวดเร็วเกินกว่าจะจับใจความทันได้ ดังนั้นคำสั่ง less จึงเป็นคำสั่งยอดนิยมอีกคำสั่งหนึ่งที่ผู้ดูแลระบบหรือนักพัฒนาจะนิยมนำมาใช้ เนื่องจากมันจะช่วยให้การแสดงข้อมูลของไฟล์ที่มีขนาดใหญ่ให้สามารถเลื่อนหน้าจอขึ้นลงได้หรือแม้กระทั่งค้นหาคำได้ลักษณะคล้ายกับการใช้ text editor ในการเลื่อนหน้าจอของคำสั่ง less นั้นสามารถใช้งานได้ทั้งลูกศรขึ้นลงหรือการใช้ปุ่ม PageUp PageDown ในการควบคุม สำหรับในการค้นหานั้นจะค้นหาได้โดยการพิมพ์เครื่องหมาย / แล้วตามด้วยข้อความที่ต้องการค้นหา คำสั่ง less จะทำหน้าที่แสดงคำที่ผู้ดูแลระบบต้องการค้นหาออกมา ถ้าหากในกรณีที่ต้องการค้นหาคำถัดไป คำสั่ง less ได้ออกแบบให้มีการกดแป้นพิมพ์ n เพื่อเลื่อนไปยังคำถัดไป

นอกจากคำสั่ง less จะมีความสามารถในการค้นหาแล้วนั้นความสามารถของคำสั่ง less ยังสามารถแสดงข้อมูลที่มีการเปลี่ยนแปลงที่บรรทัดสุดท้ายได้ โดยการกดปุ่ม F เช่นกรณีของไฟล์ log ที่จะมีข้อมูลเข้ามาตลอดเวลา less สามารถแสดงข้อมูลเหล่านั้นได้ทันทีโดยไม่จำเป็นต้องเปิดไฟล์ใหม่ แม้ไฟล์ log จะถูกบีบอัดเป็น .gz ก็ยังสามารถเปิดได้ด้วยคำสั่ง less ได้ทันที ตัวอย่างการใช้คำสั่งดังแสดงข้างล่าง

```
$ cat Departments.txt     <---- เปิดไฟล์ที่มีอยู่แล้ว
1 Electrical
2 Computer
3 Telecommunication
$ cat >Universities.txt          <---- เปิดไฟล์ใหม่ และป้อนข้อมูลลงไฟล์ทันที
Burapha University
Prince of Songkhla University
Thammasat University
กด Ctrl+D                        <---- กดปุ่ม Ctrl+D เพื่อสิ้นสุดการใส่ข้อมูล

$ cat Departments.txt Universities.txt > All.txt
$ cat All.txt
1 Electrical
2 Computer
3 Telecommunication
Burapha University
Prince of Songkhla University
Thammasat University
```

```
$ less /var/log/syslog
```



รูปที่ 1.10 แสดงผลลัพธ์การเปิดไฟล์ /var/log/syslog ด้วยคำสั่ง less

จากคำสั่งข้างต้นจะเน้นการเปิดไฟล์ชนิดข้อความ (text file) ทั่วไป แล้วแสดงผลเป็นข้อความปกติ แต่ในบางกรณีที่นักพัฒนาระบบสมองกลฝังตัวต้องการมองข้อความภายในไฟล์ในลักษณะอื่น เช่น แปลงเป็นเลขฐานแปด ฐานสิบ หรือ ฐานสิบหก คำสั่งที่นิยมใช้กันทั่วไปคือ od (Octal Dump) ซึ่งมีตัวเลือก -A ที่ไว้สำหรับระบุรากที่ต้องการ (Radix) และ -t สำหรับระบุรูปแบบการแสดงว่าจะเป็นเลขฐานต่างๆ เช่น เลขฐานแปด เลขฐานสิบ เลขฐานสิบหก หรือ รหัสแอสกี้ ด้วย ค่า o, d, x, c ตามลำดับ ดังตัวอย่างข้างล่าง

```
$ od Departments.txt
0000000 020061 066105 061545 071164 061551 066141 031012 041440
0000020 066557 072560 062564 005162 020063 062524 062554 067543
0000040 066555 067165 061551 072141 067551 005156
0000054
$ od -t x Departments.txt
0000000 6c452031 72746365 6c616369 4320320a
0000020 75706d6f 0a726574 65542033 6f63656c
0000040 6e756d6d 74616369 0a6e6f69
0000054
$ od -A d -t c Departments.txt
0000000   1       E   l   e   c   t   r   i   c   a   l  \n   2       C
0000016   o   m   p   u   t   e   r  \n   3       T   e   l   e   c   o
0000032   m   m   u   n   i   c   a   t   i   o   n  \n
0000044
```

ไฟล์ที่ใช้กันส่วนใหญ่อาจจะมีขนาดเล็กแต่เมื่อใดที่มีไฟล์ขนาดใหญ่นักพัฒนาจำเป็นที่จะต้องทำการแยกออก (split) มาให้เป็นไฟล์ชิ้นเล็กๆที่สะดวกต่อระบบสมองกลฝังตัวในการส่งไปทีละก้อน หรือแม้แต่ต้องการส่งผ่านอีเมล์ในขนาดที่พอเหมาะต่อขนาดความจุของช่องข้อมูล ดังนั้นคำสั่งที่สามารถแยกไฟล์ออกเป็นชิ้นได้คือ คำสั่ง split ซึ่งสามารถจะเลือกแยกเป็นไฟล์ชิ้นเล็กๆตามขนาดไบท์ (Byte) หรือบรรทัด (line) ที่หั่นจากตัวไฟล์ต้นฉบับ เพื่อต้องการนำไฟล์ย่อยที่ถูกแบ่งออกมา กลับมารวมเป็นไฟล์เหมือนต้นฉบับดังเดิม ก็เพียงแค่ใช้คำสั่ง cat เพื่อทำการรวมทั้งหมด ดังตัวอย่างคำสั่งข้างล่างนี้

```
$ split -l 1  Departments.txt     <---- แบ่งออกทีละบรรทัด
$ ls x*
xaa  xab  xac
$ cat xaa xab xac      <---- แสดงข้อมูลในไฟล์ทั้งสามพร้อมกัน
1 Electrical
2 Computer
3 Telecommunication

$ cat xaa xab xac > Departments_copies.txt <---- รวมไฟล์ทั้งหมดมาเป็นไฟล์เดียว
$ cat Departments_copies.txt
1 Electrical
2 Computer
3 Telecommunication
```

สามารถใช้คำสั่ง md5sum ทำการตรวจสอบไฟล์ทั้งสองว่ามีขนาดและข้อมูลภายในเหมือนกันหรือไม่

```
$ md5sum Departments.txt
c4eef8b561262f39d7ae748566100fe5  Departments.txt
$ md5sum Departments_copies.txt
c4eef8b561262f39d7ae748566100fe5  Departments_copies.txt
```

Wc, head, and tail

คำสั่ง wc (Word Count) จะเหมาะกับการตรวจสอบรายละเอียดของข้อมูลภายในไฟล์ว่ามีขนาดใหญ่เท่าไหร่ มีบรรทัดทั้งหมดจำนวนเท่าไหร่ จำนวนคำทั้งหมดเท่าไหร่ นอกจากนั้น 2 คำสั่งที่เหมาะสมในการดูไฟล์ที่มีขนาดใหญ่เช่นเดียวกัน แต่จะเน้นดูเพียงแค่ส่วนหัวไฟล์ด้วยคำสั่ง head และส่วนท้ายไฟล์ด้วยคำสั่ง tail ซึ่งสามารถใช้ตัวเลือก option -n *<บรรทัด>* เพื่อระบุจำนวนบรรทัดที่ต้องการแสดง ดังตัวอย่างคำสั่งข้างล่าง

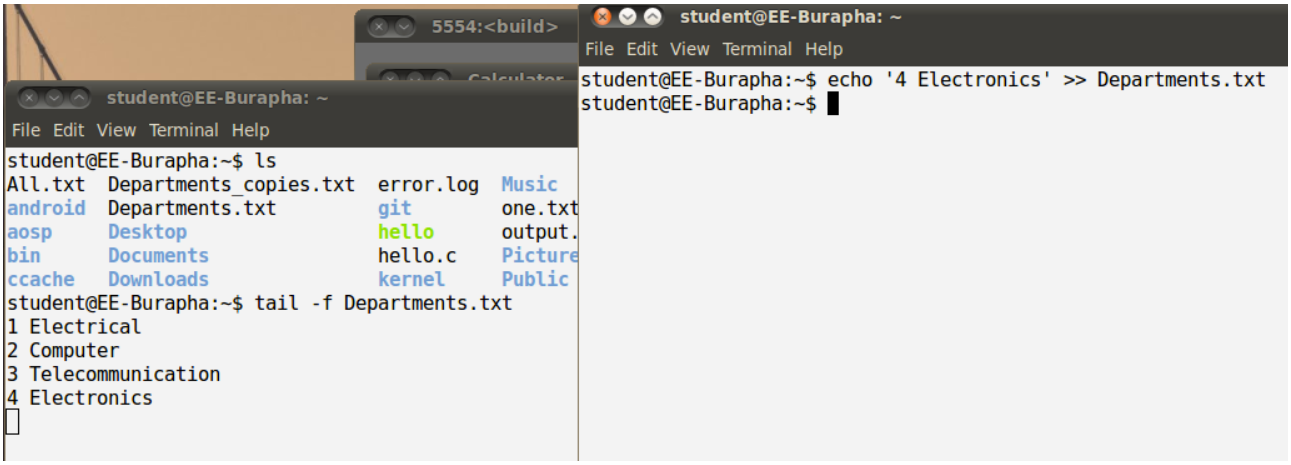
```
$ wc -l /var/log/dmesg
1386 /var/log/dmesg

$ head -n 5 /var/log/dmesg
[    0.000000] Initializing cgroup subsys cpuset
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Linux version 2.6.32-42-server (buildd@batsu) (gcc version 4.4.3
(Ubuntu 4.4.3-4ubuntu5.1) ) #96-Ubuntu SMP Wed Aug 15 19:52:20 UTC 2012 (Ubuntu
2.6.32-42.96-server 2.6.32.59+drm33.24)
```

```
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-2.6.32-42-server
root=UUID=4fb97098-124c-489b-a0d1-3eb8846d2f67 ro quiet
[    0.000000] KERNEL supported cpus:

$ tail -n 5 /var/log/dmesg
[   18.896413] acpiphp_glue: Slot 262 already registered by another hotplug driver
[   18.896434] acpiphp_glue: Slot 263 already registered by another hotplug driver
[   19.031705] ENS1371 0000:02:02.0: PCI INT A -> GSI 16 (level, low) -> IRQ 16
[   19.602812] vmmemctl: started kernel thread pid=907
[   19.602826] VMware memory control driver initialized
```

ในกรณีที่ผู้ดูแลระบบ หรือนักพัฒนาสมองกลฝังตัว ต้องการเฝ้าดูการเปลี่ยนแปลงของไฟล์ว่ามีข้อมูลเข้ามาหรือไม่ ก็สามารถใช้ตัวเลือก -f สำหรับคำสั่ง tail เพื่อเฝ้าดูเหตุการณ์การเปลี่ยนแปลงได้ทันที ดังรูป



รูปที่ 1.11 แสดงผลลัพธ์การติดตามการเปลี่ยนแปลงข้อมูลในไฟล์ด้วยคำสั่ง tail

คำสั่งค้นหาข้อความและไฟล์ด้วยชุด REGULAR EXPRESSIONS

find, grep

การค้นหาไฟล์ในระบบปฏิบัติการลีนุกซ์สามารถทำได้หลายวิธี หนึ่งในคำสั่งที่ได้รับความนิยมคือคำสั่ง find เพราะมีความยืดหยุ่นสูงในการค้นหาไฟล์แต่ละชนิดดังตารางแสดงพารามิเตอร์ (parameter) เพื่อใช้ระบุชนิดไฟล์ที่ต้องการค้นหา

ตาราง 1.7 สัญลักษณ์บอกชนิดไฟล์

สัญลักษณ์	ชนิดไฟล์
f	ไฟล์ทั่วไป
d	ไดเรกทอรี
b	block
c	character

สัญลักษณ์	ชนิดไฟล์
p	pipe
l	Symbolic link
s	Socket

ในกรณีที่ต้องการระบุให้ค้นหาเฉพาะไฟล์ทั่วไป (f) หรือ ไดเรกทอรี (d) สามารถใช้ตัวเลือก -type เพื่อระบุประเภทได้ดังตัวอย่างข้างล่าง

```
$ ls -l
total 28
-rw-r--r-- 1 student student   53 2013-08-21 05:16 error.log
-rwxr-xr-x 1 student student 8497 2013-08-21 06:05 hello
-rw-r--r-- 1 student student   92 2013-08-21 06:05 hello.c
drwxr-xr-x 2 student student 4096 2013-09-17 18:03 out
-rw-r--r-- 1 student student   57 2013-08-21 05:16 result.txt

$ find . -type f
./error.log
./result.txt
./.my_hidden_file
./hello
./hello.c

$ find . -type d
.
./.my_hidden_directory
./out
```

ในกรณีที่ต้องการค้นหาไฟล์หรือไดเรกทอรีที่ถูกซ่อนอยู่

```
$ find . -type f -name ".*"
./.my_hidden_file
```

เนื่องจากระบบไฟล์ภายใต้ระบบปฏิบัติการลีนุกซ์จะเป็นแบบ case sensitive (สนใจตัวพิมพ์เล็ก-พิมพ์ใหญ่) ดังนั้นในการค้นหาไฟล์ที่ใกล้เคียงทั้งหมดโดยไม่สนใจกรณี case sensitive สามารถระบุโดยการใช้ -iname เข้าไปดังตัวอย่างข้างล่าง

```
$ find -iname "MyProgram.c"
./myprogram.c
./unix/myprogram.c
./programming/MyProgram.c
./MyProgram.c
```

การระบุระดับความลึกของชั้นไดเรกทอรีในการค้นหา สามารถทำได้โดยการใช้ -mindepth *[level]* และ -maxdepth *[level]* ตัวอย่างเช่น ต้องการค้นหาตั้งแต่ไดเรกทอรี root (level 1) และลึกลงไปอีก 2 ชั้น (level 2 - level 3) สามารถใช้คำสั่งดังนี้

```
$ find / -maxdepth 3 -name shadow
/etc/bash_completion.d/shadow
/etc/shadow
```

เมื่อต้องการระบุว่าให้ค้นหาไฟล์ที่อยู่เฉพาะระดับความลึกที่ 2 ถึง 4 (level 2 - level 4)

```
$ find / -mindepth 2 -maxdepth 4 -name shadow
/etc/bash_completion.d/shadow
```

การค้นหาด้วยคำสั่ง find แต่ละครั้ง สามารถตั้งชุดคำสั่งพิเศษหลังตัวเลือก -exec ในกรณีที่เจอไฟล์ตามที่ต้องการค้นหา ตัวอย่างเช่นให้ทำการเช็คค่า checksum ไฟล์ที่เจอด้วยคำสั่ง md5sum

```
$ find -iname "MyProgram.c" -exec md5sum {} \;
ed832d42670ee3af4041e4a50f851755 ./myprogram.c
ed832d42670ee3af4041e4a50f851755 ./unix/myprogram.c
7562d93e8b9a19dc80959d4eb7e2c7ca ./programming/MyProgram.c
ae96bf65a87976abba3fdc57f47a55a2 ./MyProgram.c
```

ถ้าต้องการระบุการค้นหาตามสิทธิ์ของไฟล์นั้น (file permission) สามารถใช้ -perm ได้ดังตัวอย่างข้างล่าง

```
$ find . -perm -g=r -type f -exec ls -l {} \;
-rw-r--r-- 1 student student 53 2013-08-21 05:16 ./error.log
----r----- 1 student student 57 2013-08-21 05:16 ./result.txt
-rw-r--r-- 1 student student 0 2013-09-17 18:04 ./.my_hidden_file
-rwxr-xr-x 1 student student 8497 2013-08-21 06:05 ./hello
-rw-r--r-- 1 student student 92 2013-08-21 06:05 ./hello.c

$ find . -perm g=r -type f -exec ls -l {} \;
----r----- 1 student student 57 2013-08-21 05:16 ./result.txt
```

หรือระบุเป็นเลขฐานแปด (Octal)

```
$ find . -perm 040 -type f -exec ls -l {} \;
----r----- 1 student student 57 2013-08-21 05:16 ./result.txt
```

งานด้านระบบสมองกลฝังตัวขนาดของไฟล์ข้อมูลค่อนข้างเป็นประเด็นที่ต้องให้ความสนใจ เนื่องจากพื้นที่เก็บข้อมูลมีขนาดจำกัด ดังนั้นการนำคำสั่ง find มาประยุกต์ใช้เพื่อค้นหาไฟล์ตามขนาดไฟล์ที่ต้องการจึงมีความสำคัญเช่นกัน ดังตัวอย่างต่อไปนี้

เมื่อต้องการค้นหาไฟล์ที่มีขนาดใหญ่ 5 อันดับแรก

```
$ find . -type f -exec ls -s {} \; | sort -n -r | head -5
80 ./testing/ktest/ktest.pl
64 ./perf/util/trace-event-parse.c
64 ./perf/util/symbol.c
60 ./lguest/lguest.c
56 ./perf/util/header.c
```

หรือขนาดเล็กสุด 5 อันดับแรก แต่ไม่ต้องการรวมไฟล์ที่มีขนาดศูนย์ไบต์ (Empty file)

```
$ find . -not -empty -type f -exec ls -s {} \; | sort -n  | head -5
4 ./firewire/list.h
4 ./firewire/Makefile
4 ./firewire/nosy-dump.h
4 ./include/tools/be_byteshift.h
4 ./include/tools/le_byteshift.h
```

ในกรณีที่ต้องการระบุขนาดไฟล์ (byte) อย่างชัดเจน สามารถใช้ -size *[byte]* ดังตัวอย่างข้างล่าง

```
$ find . -size +200M    <------ ไฟล์ที่มีขนาดมากกว่า 200 เมกะไบต์
$ find . -size -200M    <------ ไฟล์ที่มีขนาดน้อยกว่า 200 เมกะไบต์
$ find . -size 200M     <------ ไฟล์ที่มีขนาด 200 เมกะไบต์
```

นักพัฒนาสามารถสร้างคำสั่งเฉพาะด้วย alias เพื่อความสะดวกในการค้นหาไฟล์ขนาดที่ต้องการ เพื่อลบทิ้ง ดังตัวอย่างข้างล่าง

```
$ alias rm50m="find / -type f -name *.tar -size +50M -exec rm -i {} \;"
$ alias rm1g="find / -type f -name *.tar -size +1G -exec rm -i {} \;"
$ alias rm3g="find / -type f -name *.tar -size +3G -exec rm -i {} \;"
$ alias rm5g="find / -type f -name *.tar -size +5G -exec rm -i {} \;"
```

นอกจากนั้นถ้าต้องการค้นหาไฟล์โดยดูจากเวลาของไฟล์ที่ถูกเข้าถึง (Access time) ถูกเปลี่ยนแปลงข้อมูลภายใน (Modification time) หรือ มีการเปลี่ยนแปลงไอโหนด/สถานะ (Change time) สามารถระบุตัวเลือกดังรายละเอียดในตารางข้างล่าง

ตาราง 1.8 ตัวเลือกการเข้าถึงไฟล์ของคำสั่ง find

ตัวเลือก	คำอธิบาย
--- Access Time ---	
amin *m*	เวลา *m* นาทีที่ผ่านมา ที่ไฟล์ถูกเข้าถึง
atime *d*	เวลา *d*24* ชั่วโมงที่ผ่านมา ที่ไฟล์ถูกเข้าถึง
--- Modification Time ---	
mmin *m*	เวลา *m* นาทีที่ผ่านมา ที่ไฟล์ถูกเปลี่ยนแปลงข้อมูลภายใน

ตัวเลือก	คำอธิบาย
mtime *d*	เวลา *d*24* ชั่วโมงที่ผ่านมา ที่ไฟล์ถูกเปลี่ยนแปลงข้อมูลภายใน
--- *Change Time* ---	
cmin *m*	เวลา *m* นาทีที่ผ่านมา ที่ไฟล์ถูกเปลี่ยนสถานะหรือค่าไอโหนด
ctime *d*	เวลา *d*24* ชั่วโมงที่ผ่านมา ที่ไฟล์ถูกสถานะหรือค่าไอโหนด

```
$ cd unix/
$ vim hello.c              <------ แก้ไขข้อมูลภายในไฟล์ hello.c
$ chmod 755 result.txt     <------ เปลี่ยนสิทธิ์ของไฟล์ result.txt
```

ค้นหาไฟล์ภายในไดเรกทอรี unix ที่มีการถูกแก้ไขข้อมูลภายในไฟล์ เมื่อ 10 นาทีที่ผ่านมา

```
$ find ~/unix/ -mmin -10
/home/student/unix/
/home/student/unix/hello.c
```

ค้นหาไฟล์ภายในไดเรกทอรี unix ที่มีการถูกเปลี่ยนแปลงสิทธิ์ไฟล์เมื่อ 10 นาทีที่ผ่านมา

```
$ find ~/unix/ -cmin -10
/home/student/unix/
/home/student/unix/result.txt
/home/student/unix/hello.c
```

ในกรณีที่ไม่ต้องการให้แสดงผลไฟล์ที่ถูกซ่อน (hidden file) ในผลลัพธ์ของการค้นหา สามารถใช้ -regex ดังตัวอย่างข้างล่าง

```
$ find ~/unix/ -amin -10 \( ! -regex ".*/\..*" \)
/home/student/unix/
/home/student/unix/hello.c
```

ในการค้นหาไฟล์ที่ต้องการบางครั้ง ถ้าผู้ใช้ต้องการตั้งเงื่อนไขแยกเป็น 2 ส่วนคือ ตรวจสอบสิทธิ์ และตรวจสอบขนาดไฟล์ สามารถเขียนให้อยู่ในรูปแบบดังตัวอย่างข้างล่าง

```
$ find / \( -perm -4000 -fprintf ~/unix/suid.txt '%#m %u %p\n' \) , \(
-size +100M -fprintf ~/unix/big.txt '%-10s %p\n' \)

$ cat big.txt
510743705  /home/student/aosp/out/host/linux-x86/bin/clang
357304855  /home/student/aosp/out/host/linux-x86/bin/llvm-rs-cc
```

```
138665118
/home/student/aosp/out/host/linux-x86/obj/STATIC_LIBRARIES/libclangSema_inter
mediates/libclangSema.a
138104274
/home/student/aosp/out/host/linux-x86/obj/STATIC_LIBRARIES/libclangStaticAnal
yzerCheckers_intermediates/libclangStaticAnalyzerCheckers.a
160314665  /home/student/aosp/out/host/linux-x86/obj/lib/libbcc.so
...

$ cat suid.txt
04755 root /bin/ping6
04755 root /bin/su
04755 root /bin/fusermount
04755 root /bin/umount
04755 root /bin/mount
04755 root /bin/ping
04755 root /usr/bin/chfn
06755 daemon /usr/bin/at
04755 root /usr/bin/lppasswd
04755 root /usr/bin/sudoedit
04755 root /usr/bin/arping
04755 root /usr/bin/gpasswd
...
```

grep

คำสั่ง grep (*Generalized Regular Expression Parser)* เป็นคำสั่งที่มีการใช้งานบ่อยครั้งตั้งแต่ระดับคอมพิวเตอร์ตั้งโต๊ะจนไปถึงบอร์ดสมองกลฝังตัวที่มีระบบปฏิบัติการลีนุกซ์อยู่ภายในโดยหน้าที่ของคำสั่งนี้คือการค้นหาข้อความที่อยู่ภายในไฟล์ที่ต้องการตามเงื่อนไขที่ตั้งไว้ ตัวอย่างเช่น

```
$ grep ".*passwd" suid.txt
04755 root /usr/bin/lppasswd
04755 root /usr/bin/gpasswd
04755 root /usr/bin/passwd
04755 student
/home/student/Downloads/poky-dylan-9.0.0/build/tmp/work/i586-poky-linux/shado
w/4.1.4.3-r13/package/usr/bin/passwd.shadow
...
```

ตาราง 1.9 ตัวดำเนินการสำหรับคำสั่ง grep

ตัวดำเนินการ	คำอธิบาย
?	เจออย่างน้อย 1 ตัวอักขระ
*	เจออย่างน้อย 0 หรือ มากกว่าหลายตัวอักขระ
+	เจออย่างน้อย 1 หรือ มากกว่าหลายตัวอักขระ
^	ค้นหาเฉพาะขึ้นต้นบรรทัด

ตัวดำเนินการ	คำอธิบาย
$	ค้นหาเฉพาะท้ายบรรทัด
[ ]	ค้นหาตัวอักขระใดตัวอักขระหนึ่งที่อยู่ใน [ ]
{*n*}	เจอซ้ำเป็นจำนวน *n* ครั้ง
{*n*,}	เจอซ้ำเป็นจำนวนตั้งแต่ *n* ครั้ง หรือมากกว่า
{*n*,*m*}	เจอซ้ำเป็นจำนวนตั้งแต่ *n* ครั้งจนถึง *m* ครั้ง
char	ค้นหาตัวอักขระ *char*
-E '*pattern1* .* *pattern2*'	ค้นหา *pattern1* และ *pattern2*
-E '*pattern1* | *pattern2*'	ค้นหา *pattern1* หรือ *pattern2*

ตัวอย่างการใช้คำสั่ง grep เพื่อค้นหาคำจากผลลัพธ์ของคำสั่ง `ls -al`

```
$ ls -al
total 61
drwxr-xr-x  4 student student  4096 2013-09-20 05:09 .
drwxr-xr-x 37 student student  4096 2013-09-11 20:53 ..
-rw-r--r--  1 student student  5160 2013-09-18 00:04 big.txt
-rw-r--r--  1 student student    53 2013-08-21 05:16 error.log
-rwxr-xr-x  1 student student  8497 2013-08-21 06:05 hello
-rw-r--r--  1 student student    93 2013-09-17 19:16 hello.c
drwxr-xr-x  2 student student  4096 2013-09-17 18:04 .my_hidden_directory
-rw-r--r--  1 student student     0 2013-09-17 18:04 .my_hidden_file
-rw-r--r--  1 student student     0 2013-09-20 05:09 mytest.txt
-rw-r--r--  1 student student    72 2013-09-20 05:09 number
-rw-r--r--  1 student student     0 2013-09-20 04:48 test1.txt
-rw-r--r--  1 student student     0 2013-09-20 04:48 test.txt
$ ls -al | grep test?.txt
-rw-r--r--  1 student student     0 2013-09-20 04:48 test1.txt
```

ในกรณีที่จะใช้ตัวดำเนินการ "+" จำเป็นต้องใช้โปรแกรมรุ่นใหม่ของโปรแกรม grep คือโปรแกรม egrep (Extended Grep) ดังตัวอย่างข้างล่าง

```
$ ls -al | egrep test+.txt
-rw-r--r--  1 student student     0 2013-09-20 05:09 mytest.txt
-rw-r--r--  1 student student     0 2013-09-20 04:48 test.txt
```

ในกรณีที่ต้องการค้นหาตัวเลขตามจำนวนหลักที่ต้องการ สามารถใช้ตัวดำเนินการ [ ] และ {n} ดังตัวอย่างข้างล่าง

```
$ cat numbers
ID 3710166
Telephone No. 0891234122
```

```
City Code 20131
Phone No. 038191772

$ grep  "[0-9]\{5\}$" numbers
City Code 20131
```

ถ้าต้องการค้นหาที่มีตัวเลขตั้งแต่ 7 หลักขึ้นไปจะใช้ตัวดำเนินการ {m, } ดังตัวอย่างข้างล่าง

```
$ grep  "[0-9]\{7,\}$" numbers
ID 3710166
Telephone No. 0891234122
Phone No. 038191772
```

การค้นหาข้อมูลภายในไฟล์ log นักพัฒนาสามารถทำการวิเคราะห์เบื้องต้นได้ด้วยตัวดำเนินการ -E '*pattern1*|*pattern2*' ดังตัวอย่างของข้อมูล log ข้างล่างนี้

```
$ cat data.log
1  Station01 Temp_Sensor 35.4
2  Station02 Flow_Sensor 12.4
3  Station03 Light_Sensor 55
4  Station04 Motion_Sensor ON
5  Station05 Current_Sensor 1.42
```

สามารถใช้คำสั่ง grep เพื่อค้นหาข้อมูลภายในได้ 4 รูปแบบ ได้แก่

```
(แบบที่ 1) $ grep -i 'station01\|station03' data.log
1  Station01 Temp_Sensor 35.4
3  Station03 Light_Sensor 55

(แบบที่ 2) $ grep -i -E 'station01|station03' data.log
1  Station01 Temp_Sensor 35.4
3  Station03 Light_Sensor 55

(แบบที่ 3) $ grep -i -e station01 -e station03 data.log
1  Station01 Temp_Sensor 35.4
3  Station03 Light_Sensor 55
(แบบที่ 4) $ egrep -i 'station01|station03' data.log
1  Station01 Temp_Sensor 35.4
3  Station03 Light_Sensor 55
```

เมื่อต้องการค้นหาคำที่อยู่ตำแหน่งต้นบรรทัด จะใช้ “^” นำหน้าคำที่ต้องการค้นหา หรือคำที่อยู่ตำแหน่งปลายบรรทัด จะใช้ “$” ต่อท้ายคำที่ต้องการค้นหา ดังตัวอย่างข้างล่าง

```
$ grep "^Sep 11" messages.1
Sep 11 21:16:58 EE-Burapha rsyslogd: [origin software="rsyslogd" swVersion="4.2.0"
x-pid="691" x-info="http://www.rsyslog.com"] rsyslogd was HUPed, type 'lightweight'.
Sep 11 21:44:29 EE-Burapha kernel: [ 3077.192891] e1000: eth0 NIC Link is Down
Sep 11 21:45:17 EE-Burapha kernel: [ 3125.462088] e1000: eth0 NIC Link is Up 1000
Mbps Full Duplex, Flow Control: None
```

```
Sep 11 21:46:58 EE-Burapha kernel: [ 3222.009399] e1000: eth0 NIC Link is Down
Sep 11 21:47:04 EE-Burapha kernel: [ 3227.275574] e1000: eth0 NIC Link is Up 1000
Mbps Full Duplex, Flow Control: None
...

$ grep "Link is Down$" messages.1
Sep 11 21:44:29 EE-Burapha kernel: [ 3077.192891] e1000: eth0 NIC Link is Down
Sep 11 21:46:58 EE-Burapha kernel: [ 3222.009399] e1000: eth0 NIC Link is Down
Sep 11 21:47:07 EE-Burapha kernel: [ 3230.682704] e1000: eth0 NIC Link is Down
Sep 11 21:53:34 EE-Burapha kernel: [ 3310.165309] e1000: eth0 NIC Link is Down
Sep 11 21:54:02 EE-Burapha kernel: [ 3338.102673] e1000: eth0 NIC Link is Down
...
```

การค้นหาคำที่มีอักขระพิเศษอยู่ด้วยนั้น จะต้องใช้ตัวดำเนินการ \ นำหน้าตัวอักขระพิเศษเสมอ ดังตัวอย่างข้างล่าง

```
$ grep "255\.255\.255\.255" syslog.1
Sep 18 11:36:12 EE-Burapha dhclient: DHCPDISCOVER on eth0 to 255.255.255.255 port 67
interval 3
Sep 18 11:36:13 EE-Burapha dhclient: DHCPREQUEST of 172.16.56.134 on eth0 to
255.255.255.255 port 67
Sep 18 19:54:11 EE-Burapha dhclient: DHCPDISCOVER on eth0 to 255.255.255.255 port 67
interval 3
Sep 18 19:54:12 EE-Burapha dhclient: DHCPREQUEST of 172.16.56.134 on eth0 to
255.255.255.255 port 67
Sep 20 04:50:05 EE-Burapha dhclient: DHCPDISCOVER on eth0 to 255.255.255.255 port 67
interval 5
Sep 20 04:50:06 EE-Burapha dhclient: DHCPREQUEST of 172.16.56.134 on eth0 to
255.255.255.255 port 67
```

ตาราง 1.10 ตัวดำเนินการเพิ่มเติมสำหรับคำสั่ง grep

OPERATOR	DESCRIPTION
[:digit:]	เฉพาะตัวเลขตั้งแต่ 0 ถึง 9
[:alnum:]	ทั้งตัวอักษร A ถึง Z หรือ a ถึง z และตัวเลขตั้งแต่ 0 ถึง 9
[:alpha:]	ตัวอักษร A ถึง Z หรือ a ถึง z
[:blank:]	เฉพาะช่องว่าง (Space) และแท็ป (TAB) เท่านั้น

จากตารางข้างต้น เป็นการระบุตัวอักขระในแต่ละประเภทเพื่อให้การค้นหามีประสิทธิภาพมากขึ้น ดังตัวอย่างการใช้งานดังนี้

```
$ grep "anacron\[[[:digit:]]\+\]" /var/log/syslog.1
Sep 18 08:50:17 EE-Burapha anacron[27155]: Job `cron.daily' terminated
Sep 18 08:50:17 EE-Burapha anacron[27155]: Job `cron.weekly' started
Sep 18 08:50:17 EE-Burapha anacron[27530]: Updated timestamp for job
`cron.weekly' to 2013-09-18
Sep 18 08:50:18 EE-Burapha anacron[27155]: Job `cron.weekly' terminated
...
```

การรวมคำสั่งเข้าด้วยกัน ด้วยสัญญลักษณ์ที่เรียกว่า Pipe (|) โดยเอาท์พุตของคำสั่งแรกจะเป็นอินพุตของคำสั่งถัดไป จนกระทั่งถึงคำสั่งสุดท้ายที่จะแสดงผลลัพธ์สู่หน้าจอ ดังตัวอย่างการใช้ดังนี้

```
$ ls <ไดเรกทอรี> | grep <คำค้นหา>
```

```
$ ls -al | grep "^d"
drwxr-xr-x  4 student student  4096 2013-09-21 02:21 .
drwxr-xr-x 37 student student  4096 2013-09-11 20:53 ..
drwxr-xr-x  2 student student  4096 2013-09-17 18:04 .hidden_directory
drwxr-xr-x  2 student student  4096 2013-09-17 18:03 out
```

awk

Awk ย่อมาจากตัวอักษรแรกของนักพัฒนาทั้งสามคนได้แก่ Aho, Weinberger, และ Kernighan ถือว่าเป็นภาษาโปรแกรมชนิดหนึ่งซึ่งมีไว้สำหรับการเข้าถึงข้อมูลที่มีลักษณะเป็นโครงสร้าง (structured data) และสามารถสร้างออกมาให้อยู่ในรูปแบบรายงานได้ คุณสมบัติสำคัญของ awk ประกอบไปด้วย

- awk จะมองไฟล์ข้อมูลในลักษณะคล้ายฐานข้อมูลที่มี เรคอร์ด (record) และ ฟิลด์ (field)
- awk จะมีตัวแปร (variables) ชุดคำสั่งตรวจสอบเงื่อนไข (conditional statement) และชุดคำสั่งทำ
  ซ้ำ (loop statement) เหมือนภาษาโปรแกรมทั่วไป
- awk เตรียมตัวดำเนินการทางด้านเลขคณิต (arithmetic) และข้อความ (string)
- awk สามารถสร้างออกมาให้อยู่ในรูปแบบรายงานได้

รูปแบบคำสั่งการใช้งาน

```
awk '/search pattern1/ {Actions}
     /search pattern2/ {Actions}' file
```

โดยที่ *search pattern* ก็คือชุด regular expression ส่วน *Actions* ก็คือชุดคำสั่งเชิงโปรแกรม ดังตัวอย่างการใช้คำสั่งข้างล่าง

```
$ cat data.log
1  Station01 Temp_Sensor 35.4
2  Station02 Flow_Sensor 12.4
3  Station03 Light_Sensor 55
4  Station04 Motion_Sensor ON
5  Station05 Current_Sensor 1.42
$ awk '/Station01/
> /Station02/' data.log
1  Station01 Temp_Sensor 35.4
2  Station02 Flow_Sensor 12.4
```

awk สามารถแยกเรคคอร์ดในแต่ละบรรทัดที่คั่นด้วยอักขระช่องว่างออกมาเป็นฟิลด์ย่อยๆและถูกเก็บไว้ในตัวแปร $*n* โดยที่ *n* คือจำนวนเลขฟิลด์ โดยเริ่มต้นที่ $0 จะแทนด้วยข้อความทั้งหมดของบรรทัดนั้นๆ ส่วน $1 จนถึง $*n* นั้นจะเก็บข้อความที่ถูกแยกด้วยช่องว่างแต่ละคอลัมน์ ดังตัวอย่างข้างล่าง

```
$ awk '{print $2,$4;}' data.log
Station01 35.4
Station02 12.4
Station03 55
Station04 ON
Station05 1.42
```

สามารถใช้ $NF เพื่อระบุค่าฟิลด์ตัวสุดท้ายได้ ดังตัวอย่างข้างล่าง

```
$ awk '{print $2,$NF;}' data.log
Station01 35.4
Station02 12.4
Station03 55
Station04 ON
Station05 1.42
```

คำสั่ง awk จะมีรูปแบบสำคัญอีกส่วนหนึ่งที่ใช้ในการกระทำคำสั่งตอนเริ่มต้น (Initialization) ก่อนที่จะเริ่มดำเนินการวิเคราะห์แต่ละบรรทัดภายในไฟล์ และจะกระทำคำสั่งตอนสิ้นสุด (Final) หลังจากบรรทัดสุดท้ายดำเนินการเสร็จเรียบร้อยแล้ว โดยการใช้ BEGIN และ END ดังรูปแบบการเรียกใช้คำสั่งดังนี้

```
BEGIN { Actions }
{Action} # Action for every lines in a file
END { Actions }
# is for comments in Awk
```

ตัวอย่างการใช้คำสั่ง awk แบบเต็มรูป

```
$ awk 'BEGIN {print "Station Name\tSensor Type\t\t\tValue";}
> {print $2,"\t",$3,"\t\t\t",$NF;}
> END{print "Report Generated\n--------------";
> }' data.log
Station Name      Sensor Type                Value
Station01         Temp_Sensor                35.4
Station02         Flow_Sensor                12.4
Station03         Light_Sensor               55
Station04         Motion_Sensor              ON
Station05         Voltage_Sensor             1.42
Report Generated
--------------
```

เพื่อความสะดวกยิ่งขึ้น นักพัฒนาสามารถเก็บคำสั่งลงในไฟล์สคริปท์ได้ เช่น เก็บลงในไฟล์ datalog.script ตัวอย่างการใช้คำสั่งเช่น

```
$ cat datalog.script
BEGIN {
      print "Station Name\tSensor Type\t\t\tValue";
}
{
      print $2,"\t",$3,"\t\t\t",$NF;
}
END {
      print "Report Generated\n--------------";
}
```

```
$ awk -f datalog.script data.log
Station Name      Sensor Type                    Value
Station01         Temp_Sensor                    35.4
Station02         Flow_Sensor                    12.4
Station03         Light_Sensor                   55
Station04         Motion_Sensor                  ON
Station05         Voltage_Sensor                 1.42
Report Generated
--------------
```

ในการใช้งานในระบบสมองกลฝังตัวนั้นการดูค่าสถานะต่างๆนั้นบอร์ดสมองกลฝังตัวอาจจะมีโปรแกรมจำลองตัวเองให้มีบริการ Web Server เพื่อให้ผู้ใช้สามารถเข้ามาดูสถานะผ่านโปรแกรมเว็ปบราวเซอร์ทั่วไปได้ ดังนั้นนักพัฒนาสามารถจัดรูปแบบการแสดงผลข้อมูลโดยเพิ่มแท็ก HTML (HTML Tag) ด้วยคำสั่ง awk ได้อย่างง่ายดาย ดังตัวอย่างข้างล่าง

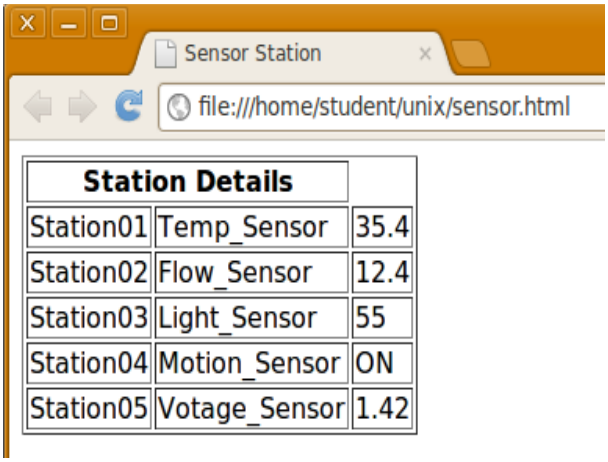
```
$ cat datalog_html.script
BEGIN{
title="Sensor Station";
print "<html>\n<title>"title"</title><body bgcolor=\"#ffffff\">\n<table bor-
der=1><th  colspan=2 align=centre>Station Details</th>";
}
{
station_name=$2;
sensor=$3;
value=$NF;
print "<tr><td>"station_name"</td><td>"sensor"</td><td>"value"</td></tr>";
}
END {
print "</table></body>\n</html>";
}
```

วิธีการเรียกใช้ script ที่สร้างขึ้นมา (datalog_html.script) และเก็บผลลัพธ์ที่ได้อยู่ในรูปแบบแท็ก HTML ลงในไฟล์ใหม่ชื่อ sensor.html ดังตัวอย่างคำสั่งข้างล่าง

```
$ awk -f datalog_html.script data.log > sensor.html
```

```
$ cat sensor.html
<html>
<title>Sensor Station</title><body bgcolor="#ffffff">
<table border=1><th  colspan=2 align=centre>Station Details</th>
<tr><td>Station01</td><td>Temp_Sensor</td><td>35.4</td></tr>
<tr><td>Station02</td><td>Flow_Sensor</td><td>12.4</td></tr>
<tr><td>Station03</td><td>Light_Sensor</td><td>55</td></tr>
<tr><td>Station04</td><td>Motion_Sensor</td><td>ON</td></tr>
<tr><td>Station05</td><td>Voltage_Sensor</td><td>1.42</td></tr>
</table></body>
</html>
```

ทดสอบเปิดไฟล์ sensor.html ผ่านโปรแกรมบราวเซอร์เพื่อดูผลลัพธ์ที่เกิดขึ้น



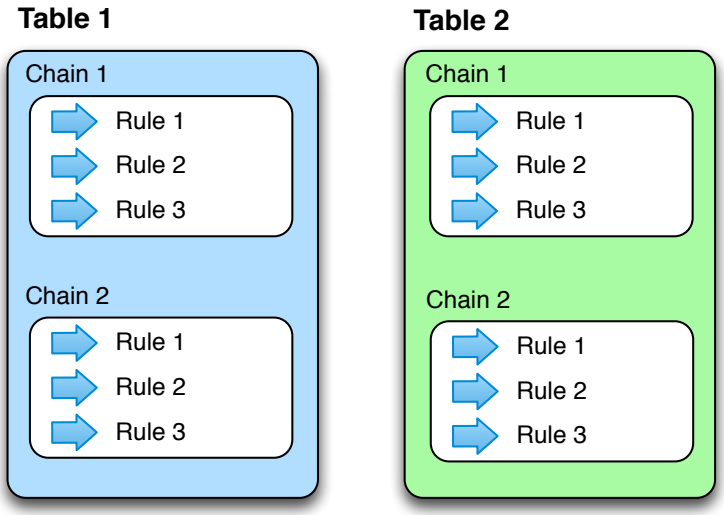
รูปที่ 1.12 แสดงการนำเสนอข้อมูลในรูปแบบ HTML

คำสั่งจัดการด้านระบบเครือข่าย

iptables

iptables เป็นเครื่องมือสำคัญอีกตัวหนึ่งที่ถูกนำมาใช้ในการจัดการระบบความปลอดภัยในระบบเครือข่ายที่เรียกว่า ไฟร์วอลล์ (firewall) โดยพื้นฐานในการกำหนดการควบคุมการเข้าออกของข้อมูลนั้น ผู้ดูแลระบบหรือแม้แต่นักพัฒนาทางด้านระบบสมองกลฝังตัวควรเข้าใจโครงสร้างและหลักการทำงานของ iptables เป็นอย่างดี

ภายใน iptables นั้นมีด้วยกันหลายตาราง (tables) โดยแต่ละตารางจะประกอบไปด้วยหลายรายการที่เรียกว่า chain (ที่ iptables เตรียมมาให้แล้วหรือผู้ใช้กำหนดขึ้นเอง) และภายในแต่ละ chain จะประกอบไปด้วยข้อกำหนดการควบคุมการเข้าออกของข้อมูลที่เรียกว่า rule ดังโครงสร้างแสดงในรูปข้างล่าง



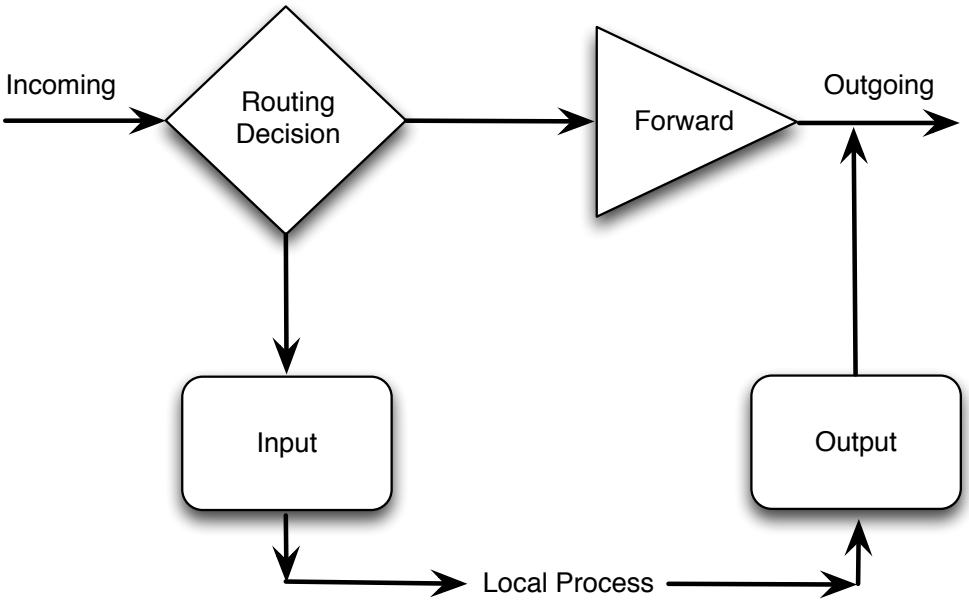
รูปที่ 1.13 ตารางที่ใช้ในการกำหนดกฎของ iptables

iptables จะถูกเตรียมตารางให้แล้ว 4 ตาราง (4 built-in tables) ได้แก่

1. Filter Table
2. NAT Table
3. Mangle Table
4. Raw Table

ส่วนประกอบของ iptables จะมีส่วนประกอบหลัก 3 ส่วนได้แก่

- INPUT คือส่วนของข้อมูลที่เข้ามาสู่เครื่องคอมพิวเตอร์
- OUTPUT คือส่วนของข้อมูลที่ออกจะเครื่องคอมพิวเตอร์
- FORWARD คือส่วนที่ส่งต่อข้อมูลจากระบบเครือข่ายภายในสู่เครือข่ายภายนอก ซึ่งการบล็อกพอร์ตก็จะใช้ส่วนนี้เป็นหลัก



รูปที่ 1.14 โฟลว์ชาร์ตแสดงการควบคุมการไหลของข้อมูล

ตัวอย่างการแสดงรายละเอียดของตารางภายใน iptables

```
$ sudo iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:ssh
DROP       all  --  anywhere             anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

โดยที่

- ACCEPT คือไฟร์วอลล์จะยอมให้แพ็กเก็ต (packet) ผ่านไปยังปลายทางได้
- DROP คือไฟร์วอลล์จะทิ้งแพ็กเก็ตทันที แต่จะไม่แจ้งผู้ส่งเกี่ยวกับข้อความที่ส่งไม่สำเร็จ
- REJECT คือไฟร์วอลล์จะทิ้งแพ็กเก็ตทันที และผู้ส่งจะได้รับข้อความผ่าน ICMP ตอบกลับถึงข้อความที่ส่ง
  ไม่สำเร็จ
- QUEUE คือไฟร์วอลล์จะส่งต่อแพ็กเก็ต ไปยังส่วนระบบบนที่ติดต่อกับผู้ใช้ (userspace)
- RETURN คือไฟร์วอลล์จะหยุดการทำงานภายใน chain แล้วกลับไปยัง Chain เดิมที่เรียกก่อนหน้านี้

ตัวอย่างแสดงการเปิดพอร์ตด้วยคำสั่ง iptables โดยหลักการสำคัญคือ ควรปิดพอร์ตทั้งหมดก่อนแล้วจึงค่อยเลือกเปิดพอร์ตที่ใช้ทีละพอร์ตจะเป็นวิธีที่มีความปลอดภัยที่สุด

```
$ IPTABLES -P FORWARD DROP      #CLOSE PORT ALL
$ IPTABLES -A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT
$ IPTABLES -A FORWARD -p tcp --dport 53 -j ACCEPT    #DNS
$ IPTABLES -A FORWARD -p udp --dport 53 -j ACCEPT #DNS
$ IPTABLES -A FORWARD -p udp --dport 67 -j ACCEPT #DHCP
$ IPTABLES -A FORWARD -p udp --dport 69 -j ACCEPT #TFTP
$ IPTABLES -A FORWARD -p udp --dport 111 -j ACCEPT   #NFS
$ IPTABLES -A FORWARD -p udp --dport 2049 -j ACCEPT  #NFS
$ IPTABLES -A FORWARD -p udp --dport 32700 -j ACCEPT #NFS
$ IPTABLES -A FORWARD -p tcp --dport 80 -j ACCEPT #HTTP
$ IPTABLES -A FORWARD -p tcp --dport 8080 -j ACCEPT  #HTTP
$ IPTABLES -A FORWARD -p tcp --dport 443 -j ACCEPT   #HTTPS
$ IPTABLES -A FORWARD -p tcp --dport 8443 -j ACCEPT  #HTTPS
$ IPTABLES -A FORWARD -p tcp --dport 20 -j ACCEPT #FTP
$ IPTABLES -A FORWARD -p udp --dport 20 -j ACCEPT #FTP
$ IPTABLES -A FORWARD -p tcp --dport 21 -j ACCEPT #FTP
$ IPTABLES -A FORWARD -p udp --dport 21 -j ACCEPT #FTP
$ IPTABLES -A FORWARD -p tcp --dport 22 -j ACCEPT #SSH
$ IPTABLES -A FORWARD -p tcp --dport 23 -j ACCEPT #TELNET
```

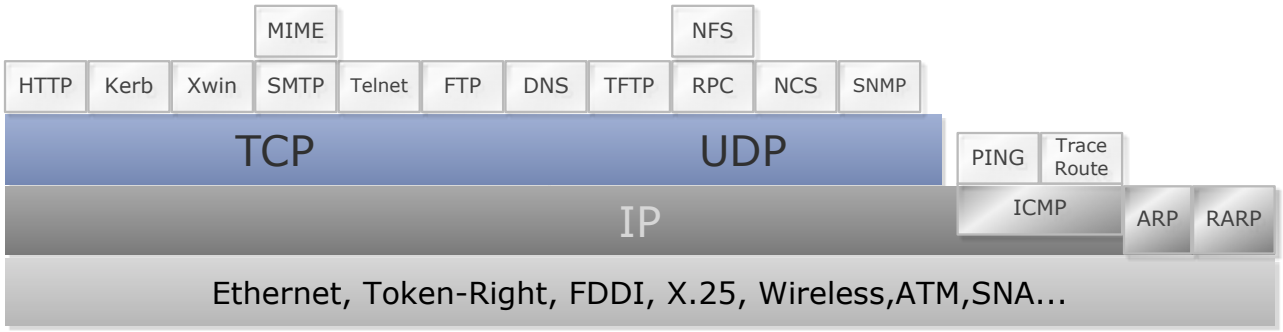
พอร์ตพื้นฐานที่จะถูกเปิดไว้สำหรับบอร์ดสมองกลฝังตัว เพื่อใช้ในการติดตั้ง bootload-er,ลีนุกซ์คอร์เนลและ เรียก root filesystem ได้นั้นจะมีด้วยกันอย่างน้อย 4 ถึง 5 พอร์ต ได้แก่

ตาราง 1.11 รายการ services ที่ถูกเปิดใช้ในระบบสมองกลฝังตัว

SERVICE	PORT NAME	PORT NO.	TCP/IP PROTOCOL
DHCP	bootps	67	UDP
TFTP	tftp	69	UDP
NFS	sunrpc	111	UDP
	nfs	2049	UDP
	mountd	32700 หรือ 32772	UDP

ในกรณีที่ต้องการปิดการทำงานของไฟร์วอลล์ เพื่อเข้าสู่โหมดบำรุงรักษาระบบสามารถใช้คำสั่งดังต่อไปนี้

```
$ sudo /etc/init.d/iptables off
```



รูปที่ 1.15 รายละเอียดของโปรโตคอลในแต่ละชั้นของ TCP/IP

TFTP

การทำงานของ Trivial File Transfer Protocol (TFTP) เป็นการทำงานที่เป็นการส่งไฟล์ระหว่างระบบ โดยทั่วไปผู้ใช้จะรู้จัก FTP ซึ่งเป็นโปรโตคอลที่มีการใช้งานอย่างกว้างขวางที่อนุญาตให้ผู้ใช้สามารถมองเห็นโครงสร้างระบบไฟล์และสามารถดาวน์โหลดหรืออัพโหลดไฟล์ได้ แต่สำหรับ TFTP นั้นขั้นตอนการทำงานจะน้อยกว่า FTP กล่าวคือไม่ต้องมีขั้นตอนการรับรองตัวตน (authentication) และเป็นการส่งข้อมูลผ่านโปรโตคอล UDP แต่อย่างไรก็ตาม TFTP ก็ยังถูกมีการใช้งานอย่างกว้างขวางในการส่งไฟล์ configuration ต่างๆบนระบบเครือข่ายคอมพิวเตอร์

ขั้นตอนการติดตั้งโปรแกรม TFTP ลงในเครื่อง host เพื่อใช้ในการเก็บไฟล์ image สำหรับเครื่องลูกข่ายที่เป็นบอร์ดสมองกลฝังตัว

ในกรณีที่ใช้ Ubuntu เวอร์ชั่น 12.04 ขึ้นไป จะใช้คำสั่งดังนี้

```
$ sudo apt-get install xinetd tftp-hpa tftpd-hpa
```

แต่ถ้าในกรณีที่ใช้ Ubuntu เวอร์ชั่น 11.10 ลงมา จะใช้คำสั่งดังนี้

```
$ sudo apt-get install xinetd tftp tftpd
$ sudo mkdir /tftpboot
```

หลังจากติดตั้งโปรแกรมและสร้างไดเรกทอรี tftpboot เรียบร้อยแล้ว จะต้องมีการตั้งค่าในไฟล์ `/etc/xinetd.conf` และแก้ไขตำแหน่งไดเรกทอรี TFTP_DIRECTORY ที่ถูกระบุอยู่ในไฟล์ `/etc/default/tftpd-hpa` (ในกรณีที่ใช้ Ubuntu เวอร์ชั่น 12.04 ขึ้นไป)

```
$ sudo vim /etc/xinetd.conf
service tftp
{
    socket_type = dgram
    protocol = udp
    wait = yes
    user = root
    server = /usr/sbin/in.tftpd
    server_args = -s /tftpboot
    disable = no
 }
```

```
$ sudo vim /etc/default/tftpd-hpa   (ในกรณีที่ใช้ Ubuntu เวอร์ชั่น 12.04 ขึ้นไป)
TFTP_USERNAME="tftp"
TFTP_ADDRESS="0.0.0.0:69"
TFTP_OPTIONS="--secure"
TFTP_DIRECTORY="/home/training/tftpboot"
```

ทำการเริ่มต้นบริการ xinetd ใหม่ด้วยคำสั่ง

```
$ sudo /etc/init.d/xinetd restart
```

ทำการสร้างไฟล์ต้นฉบับไว้ในไดเรกทอรี /tftpboot/ บนเครื่อง Host

```
$ sudo echo 'This is a data' > /tftpboot/myfile
```

ผู้ใช้สามารถทดสอบการดาวน์โหลดไฟล์โดยการทำการเชื่อมต่อไปยัง TFTP server ดังตัวอย่างคำสั่งข้างล่างนี้

```
$ tftp <Host IP>  <----- ระบุหมายเลข IP ของเครื่อง TFTP server
tftp> get myfile
Received 16 bytes in 0.1 seconds
tftp> quit
```

DHCP

บอร์ดสมองกลฝังตัวในหลายงานประยุกต์อาจจะต้องร้องขอหมายเลข IP Address ให้กับตัวมันเอง หรือในระหว่างการพัฒนาบอร์ดสมองกลฝังตัวจะถูกเชื่อมต่ออยู่ในระบบเครือข่ายเดียวกันกับเครื่อง Host ดังนั้นภายในระบบเครือข่ายจะมีเครื่องแม่ข่ายที่ทำหน้าที่จ่ายหมายเลข IP Address ผ่านโปรโตคอล DHCP ให้กับลูกข่ายหรือบอร์ดสมองกลฝังตัว (target board) ในกรณีที่เป็นเครื่อง Host นักพัฒนาจะต้องทำการติดตั้ง DHCP server เพิ่มเติมเอง ดังขั้นตอนดังนี้

```
$ sudo apt-get install isc-dhcp-server
```

แก้ไขไฟล์ /etc/dhcp/dhcpd.conf เพื่อทำการตั้งค่าให้เครื่อง Host เป็น DHCP Server โดยเพิ่มบรรทัดดังต่อไปนี้

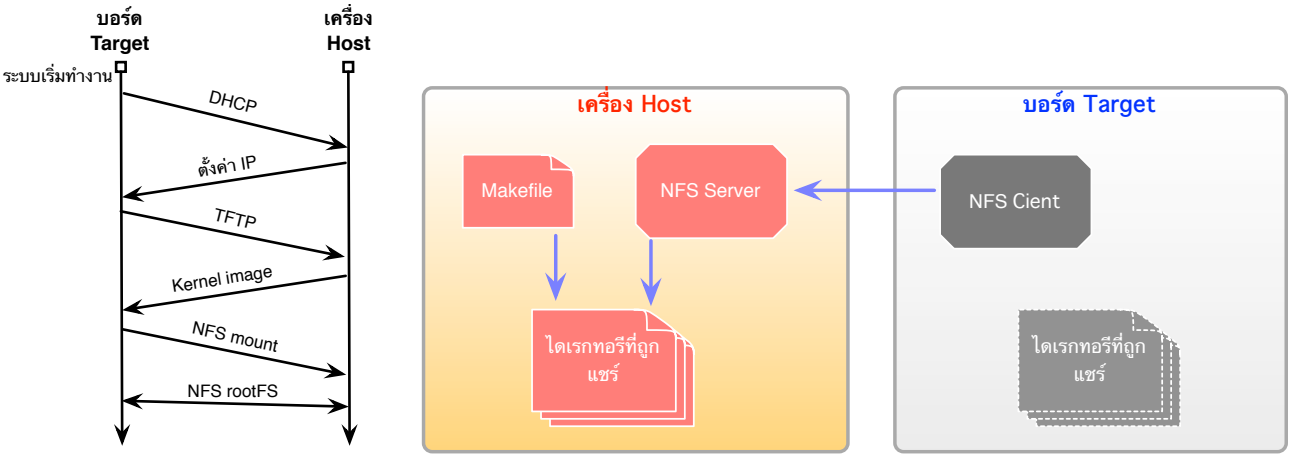
```
$ sudo vim /etc/dhcp/dhcpd.conf
subnet 192.168.0.0 netmask 255.255.255.0 {
    host targetboard {
        fixed-address 192.168.0.2;
         server-name "192.168.0.1";
        option router "192.168.0.1";
        hardware ethernet 12:34:56:78:9a:bc;
        option root-path "/rootfs";
        filename uImage;
    }
 }
```

จากตัวอย่างภายในไฟล์ dhcpd.conf ข้างต้น เครื่อง DHCP server จะจ่าย IP Address ให้กับบอร์ดสมองกลฝังตัวที่มี MAC address เป็น 12:34:56:78:9a:bc ดังนั้นผู้ใช้จะต้องปรับค่า MAC address บนบอร์ดสมองกลฝังตัว ให้ตรงตามค่า MAC Address ที่เครื่อง DHCP Server รู้จักให้ตรงกัน

NFS

nfs เป็นโปรโตคอลที่ได้รับการออกแบบมาเพื่อการเชื่อมโยงทรัพยากรฮาร์ดดิสก์จากเครื่องอื่นๆที่อยู่บนเครือข่ายคอมพิวเตอร์ที่อยู่ห่างออกไปให้เป็นเสมือนระบบไฟล์ของอุปกรณ์เอง สำหรับนักพัฒนาระบบสมองกลฝังตัวแล้วการใช้งาน nfs มีผลทำให้ในระหว่างการปรับแต่งระบบปฏิบัติการ หรือการพัฒนาโปรแกรมเพื่อใช้ในระบบสมองกลฝังตัวคล่องตัวและสะดวกมากยิ่งขึ้นและไม่มีข้อจำกัดของขนาดในการเก็บข้อมูล โดยการนำไฟล์ root file system (RFS) ไปวางไว้ในไดเรกทอรีที่ถูกตั้งค่าให้เป็นไดเรอทอรีที่ถูกแชร์ผ่านโปรโตคอล NFS

เนื่องจากโดยทั่วไปแล้วเมื่อนักพัฒนาได้ปรับแต่งระบบปฏิบัติการเสร็จ และพัฒนาโปรแกรมฝังเข้าไป root file system เรียบร้อยก็จะถูกนำไปเขียนลงในตัวเก็บข้อมูลที่อยู่ภายในบอร์ดสมองกลฝังตัว เช่น Flash Memory เป็นต้น



รูปที่ 1.16 แสดงขั้นตอนการ mount พื้นที่บนเครื่อง Host ด้วยโปรโตคอล NFS

ในระบบปฏิบัติการลีนุกซ์บนระบบสมองกลฝังตัว การเชื่อมต่อระบบเข้ากับ Network File System นั้น ในขั้นตอนการทำงานจะมีลักษณะเดียวกับการเชื่อมต่อของระบบปฏิบัติการลีนุกซ์โดยทั่วไป ขั้นตอนการติดตั้ง nfs-kernel-server และตั้งค่าในไฟล์ /etc/exports ดังนี้

```
$ sudo apt-get install rpcbind nfs-kernel-server
$ sudo mkdir /rootfs
$ sudo vim /etc/exports
/rootfs 192.168.0.2(rw,sync,no_subtree_check,no_root_squash)
/rootfs localhost(rw,sync,no_subtree_check,no_root_squash)
```

ทำการเรียก service ที่เกี่ยวข้องทั้งหมด ด้วยคำสั่ง

```
$ sudo service xinetd restart
$ sudo service tftpd-hpa restart
$ sudo service isc-dhcp-server restart
$ sudo service rpcbind-boot stop
$ sudo service nfs-kernel-server stop
$ sudo service rpcbind-boot start
$ sudo service nfs-kernel-server start
```

ไฟล์ `/etc/exports` เป็นไฟล์ที่ได้ถูกสร้างขึ้นจากเครื่องแม่ข่าย เพื่อระบุว่าจะให้ไดเรกทอรีใดในเครื่องแม่ข่ายที่จะให้เครื่องลูกข่ายสามารถทำการ mount ไดเรกทอรีได้ ด้วยคำสั่งข้างล่าง

```
$ mount –t nfs nfs_server_Address:/rootfs/ /mnt/rfs
```

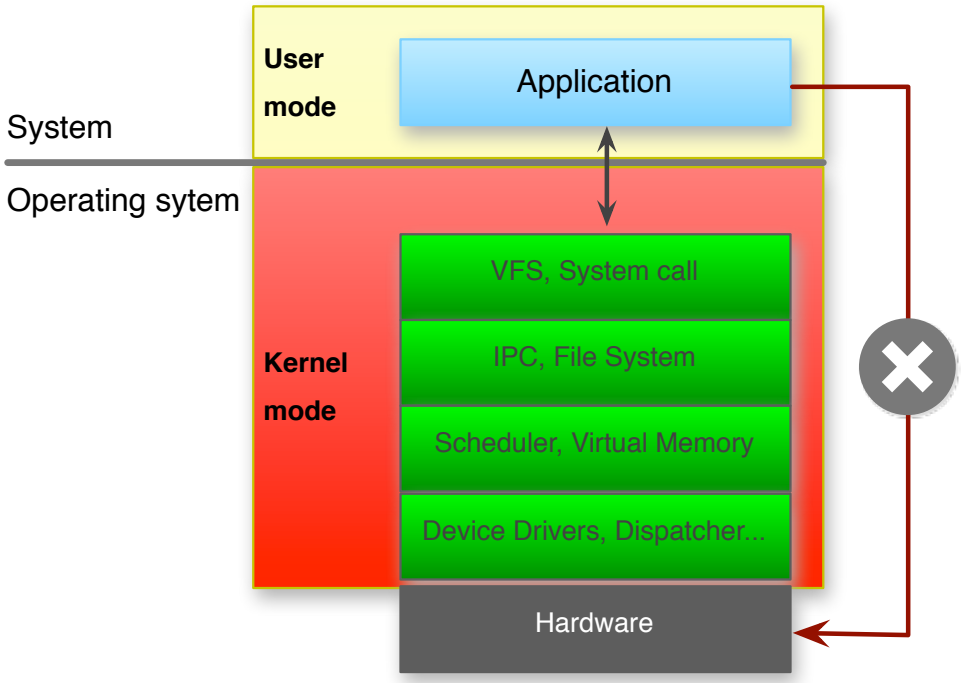
*ตัวอย่าง shell script สำหรับช่วยในการติดตั้ง tftp & nfs บนเครื่องแม่ข่าย
http://goo.gl/XuB1EZ

บทที่ 2 พื้นฐานลีนุกซ์คอร์เนลสำหรับนักพัฒนา

เคอร์เนล (Kernel) สามารถแยกออกมาได้ 3 ประเภท ได้แก่

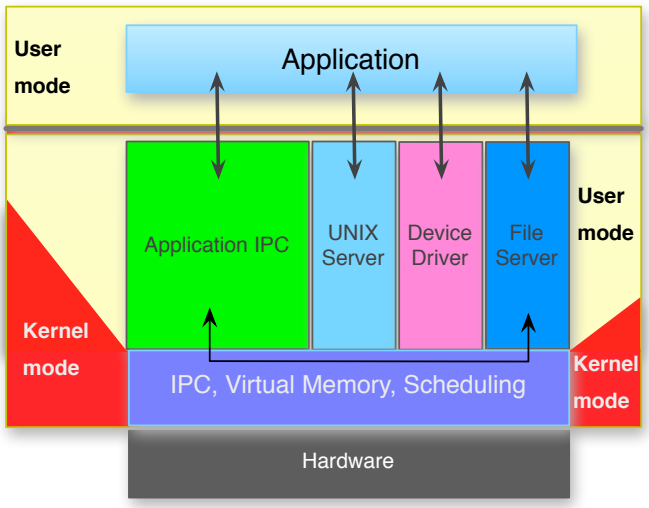
- *Monolithic kernel* เช่น Linux Kernel, MS-DOS, Microsoft Windows 9x Series
- *Micro kernel* เช่น AIX, AmigaOS, Android OS, Haiku, L4 microkernel family เป็นต้น
- *Hybrid kernel* เช่น BeOS kernel, NetWare kernel, ReactOS kernel, NT kernel Windows NT kernel (Windows 2000/Windows XP/Windows 2003/Windows Vista), 8.XNU kernel (ใช้ใน Mac OS X) เป็นต้น

และเป็นที่ทราบกันดีว่าลีนุกซ์คอร์เนลโดยส่วนใหญ่นั้นจะเป็นชนิด monolithic kernel ซึ่งหมายถึงหน้าที่หลักโดยส่วนใหญ่ของระบบปฏิบัติการจะถูกเรียกผ่าน Kernel ทั้งหมด ดังแสดงในรูปข้างล่าง



รูปที่ 2.1 แสดงการรายละเอียดการทำงานของ Monolithic Kernel

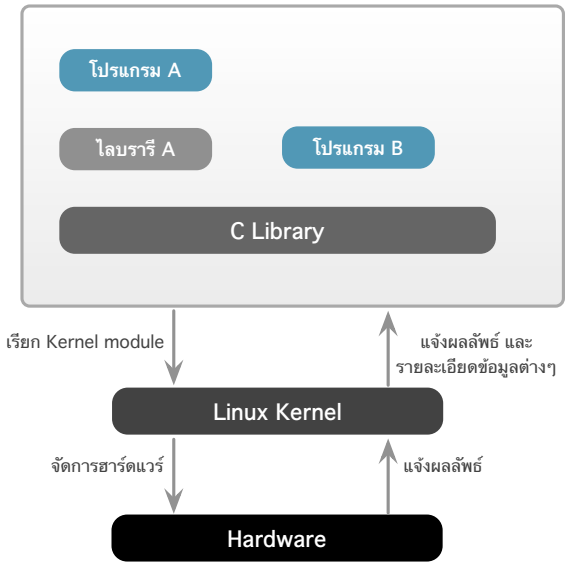
แตกต่างจาก Micro Kernel ที่บางส่วนของระบบปฏิบัติการยังคงทำใน Kernel เช่น การสื่อสารระหว่างโปรเซส (inter-process communication) การจัดลำดับงานของอุปกรณ์อินพุต/เอาท์พุต (basic input/output scheduling) การจัดการหน่วยความจำ (memory management) ส่วนหน้าที่อื่นๆจะทำภายนอก Kernel ตัวอย่างเช่น drivers, network stack, file systems



รูปที่ 2.2 แสดงการรายละเอียดการทำงานของ Micro Kernel

Linux Kernel

ลีนุกซ์คอร์เนลซึ่งเป็นตัวกลางสำคัญของระบบปฏิบัติการลีนุกซ์ในการติดต่อระหว่างฮาร์ดแวร์และซอฟต์แวร์ โดยฮาร์ดแวร์นั้นหมายถึงอุปกรณ์ต่างๆภายในและอุปกรณ์รอบข้างเครื่องคอมพิวเตอร์ ตัวอย่างเช่น หน่วยประมวลผลกลาง หน่วยความจำ การ์ดแสดงผล ฮาร์ดดิสก์ อุปกรณ์อินพุตและเอาท์พุต เมาส์ คีย์บอร์ด เป็นต้น สำหรับซอฟแวร์นั้นประกอบไปด้วยโปรแกรมของระบบปฏิบัติการและโปรแกรมประยุกต์ต่างๆ โดยภายใน Kernel จะประกอบไปด้วย 2 ส่วนสำคัญคือ Kernel Module และ Device Driver ทั้งสองจะทำหน้าที่ในการดูแลจัดการการร้องขอที่เกิดขึ้นจากฮาร์ดแวร์และซอฟต์แวร์ แล้วทำการประมวลผลข้อมูลในเบื้องต้นเพื่อส่งต่อให้ระบบปฏิบัติการต่อไป เพื่อให้บริหารจัดการการใช้งานทรัพยากรทั้งหมดได้อย่างมีระบบ



รูปที่ 2.3 แสดงการสื่อสารระหว่างโปรแกรมในระบบปฏิบัติการและอุปกรณ์

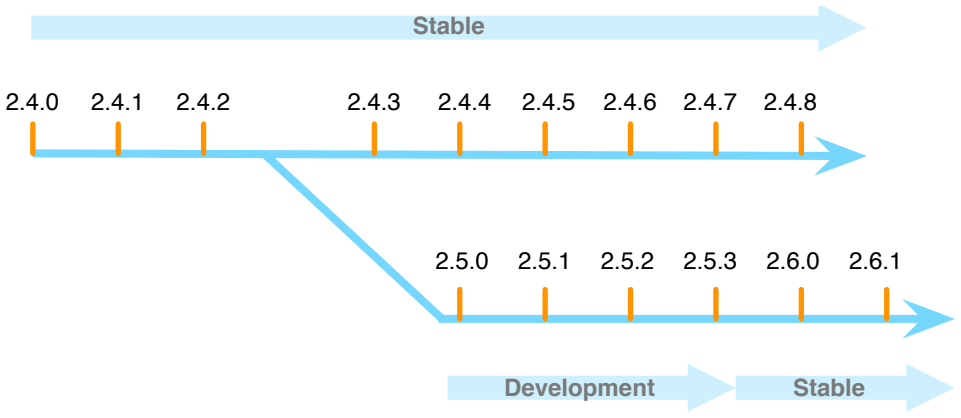
จุดเด่นสำคัญอีกจุดหนึ่งคือลีนุกซ์คอร์เนลสามารถรองรับสถาปัตยกรรมที่มีอยู่ตั้งแต่ในอดีตจนถึงปัจจุบัน ซึ่งสามารถดูรายชื่อได้จากไดเรกทอรี `/arch` โดยสามารถแยกเป็นกลุ่มสถาปัตยกรรมได้ 2 แบบคือแบบ 32 บิต เช่น arm, avr32, blackfin, m68k, microblaze, mips, score, sparc, um, x86, powerpc และกลุ่มสถาปัตยกรรมแบบ 64 บิต เช่น alpha, arm64, ia64, sparc64, tile, x86_64, powerpc

รายละเอียดของแต่ละสถาปัตยกรรมเหล่านี้นักพัฒนาสามารถเข้าไปอ่านเพิ่มเติมได้จากไดเรกทอรี `arch/<arch>/Kconfig`, `arch/<arch>/README` หรือในไดเรกทอรี `Documentation/<arch>/` นอกจากนั้นลีนุกซ์คอร์เนลยังถูกปรับปรุงโค้ดภายใน ให้ทำงานได้อย่างมีประสิทธิภาพ และมีความยืดหยุ่นสูงกับสถาปัตยกรรมที่หลากหลายอย่างต่อเนื่องและรองรับการเข้ากันได้กับมาตราฐานฮาร์ดแวร์รุ่นใหม่ๆแต่ยังคงได้รับการควบคุมจากผู้เชี่ยวชาญที่ดูแลซอร์สโค้ดจากทั่วโลกเพื่อไม่ให้ใครแอบซ่อนโค้ดที่ไม่พึงประสงค์หรือสร้างความไม่ปลอดภัยให้กับระบบโดยรวม

สำหรับนักพัฒนาระบบสมองกลฝังตัวสามารถที่จะเลือกใช้ฟังก์ชั่นบางตัวในลีนุกซ์คอร์เนลเพื่อให้เหมาะสมกับระบบฮาร์ดแวร์ที่มีอยู่บนบอร์ดสมองกลรวมทั้งสามารถเลือกโปรแกรมประยุกต์บางตัวที่ต้องการให้ทำงานอยู่ในบอร์ดสมองกลฝังตัวได้

LINUX VERSIONING

โดยปกติทุก 2-3 ปี จะมีการออกรุ่นเสถียร (stable) ของลีนุกซ์คอร์เนลที่เป็นเลขคู่ เช่น 1.0.x, 2.0.x, 2.**2**.x, 2.**4**.x, 2.**6**.x, 3.0.x เมื่อมีการพัฒนาปรับปรุงฟังก์ชั่นใหม่ๆเข้าไปและมีการเปลี่ยนแปลงโค้ดชุดใหญ่ ก็จะออกเลขรุ่นโดยใช้เป็นเลขคี่ เช่น 2.**1**.x, 2.**3**.x, 2.**5**.x แต่สำหรับการปรับปรุงเปลี่ยนแปลงในระดับเล็กลงมา (Minor release) จะใช้เปลี่ยนเลขรุ่นหลักที่สาม เช่น 2.5.**12**, 2.6.**39**



รูปที่ 2.4 แสดงการออกเลขเวอร์ชันของ Linux Kernel

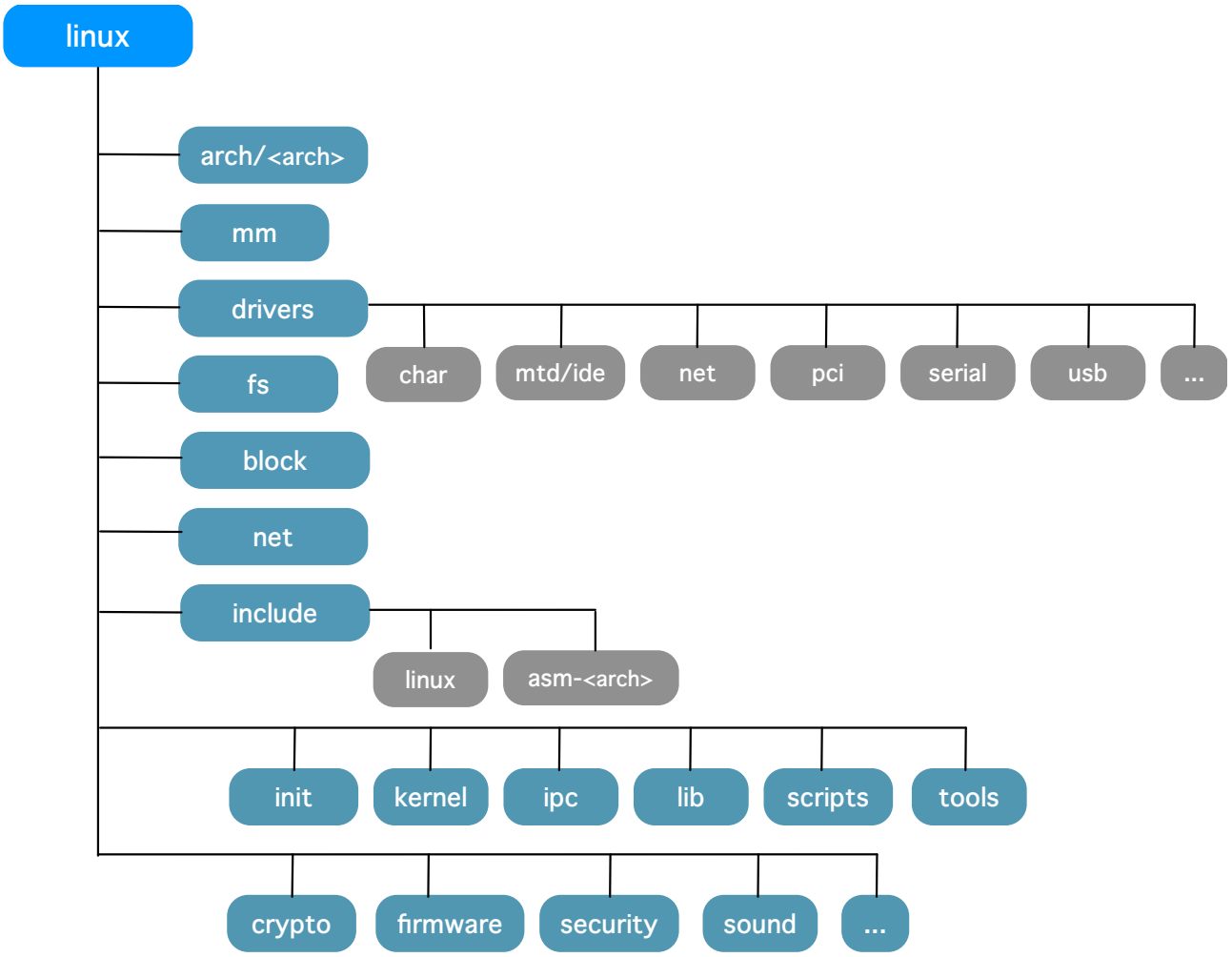
ตั้งแต่ปี ค.ศ. 2003 ถึง ค.ศ. 2011 รุ่น 2.6.x เป็นรุ่นที่มีระยะการใช้งานยาวนานมากเป็นพิเศษ อาจเนื่องมาจากช่วงนั้นเป็นยุคของการเติบโตและเปลี่ยนแปลงของคอมพิวเตอร์รวมทั้งอุปกรณ์อาร์ดแวร์ภายใน

เครื่องและอุปกรณ์ต่อพ่วงรอบข้างอย่างรุนแรงและยังเป็นการเกิดขึ้นของยุคคอมพิวเตอร์ชนิดพกพา เช่น Laptop, Netbook, Mobile Internet Device, Smart Phone, Tablet

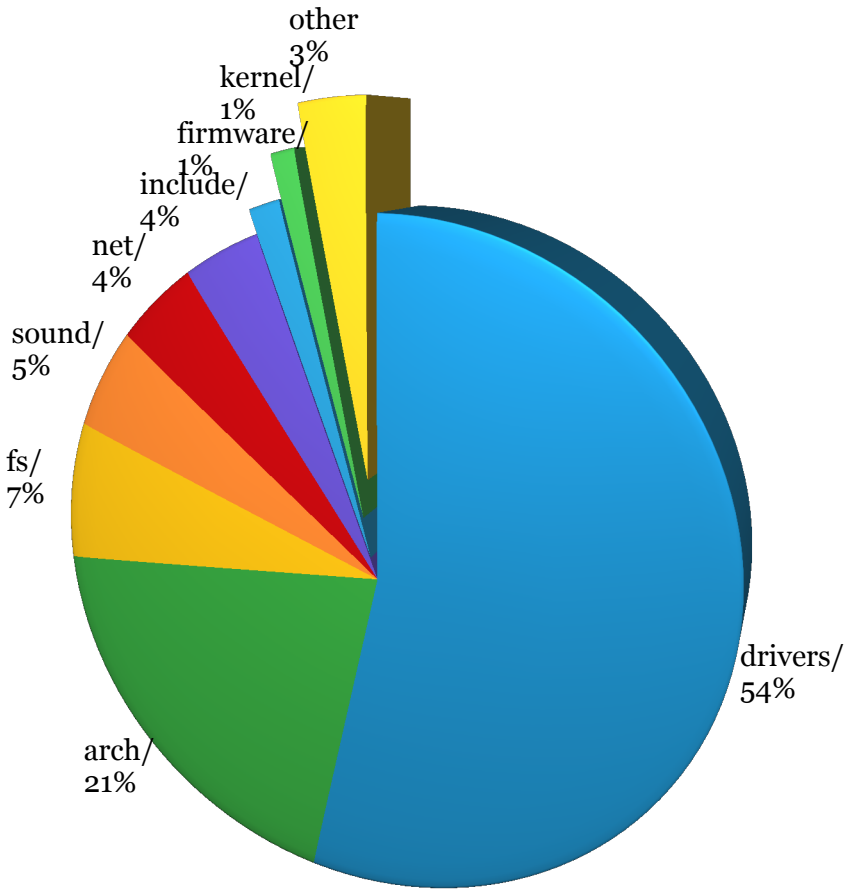
ในที่สุดลีนุกซ์คอร์เนลรุ่น 3.0 ก็เริ่มประกาศเป็นทางการในเดือน กรกฎาคม ปีค.ศ. 2011 ซึ่งเป็นการเปลี่ยนขยับตัวเลขจาก 2.6 ไปสู่ 3.0 ที่ยาวนานแต่กลับไม่ได้เป็นการแก้ไขในระดับโค้ดมากแต่อย่างใด ขนาดภายในลีนุกซ์คอร์เนล 3.x จะมีขนาดโดยรวมอยู่ประมาณ 434 MB ด้วยจำนวนไฟล์ถึง 39,400 กว่าไฟล์ (มากกว่า 14,800,000 บรรทัด) ดังนั้นถ้าต้องการจะทำการบีบอัดให้มีขนาดเล็กที่สุดควรจะเป็นนามสกุล .xz (ลดลงไปได้ประมาณ 85.7%)

กรณีการพัฒนาระบบสมองกลฝังตัวนั้นสามารถใช้ลีนุกซ์คอร์เนลเล็กที่สุดด้วยขนาดเพียง 1.3 MB เพื่อให้เหมาะสมกับข้อจำกัดของทรัพยากรภายในบอร์ดเมื่อเทียบกับเครื่องคอมพิวเตอร์ทั่วไป

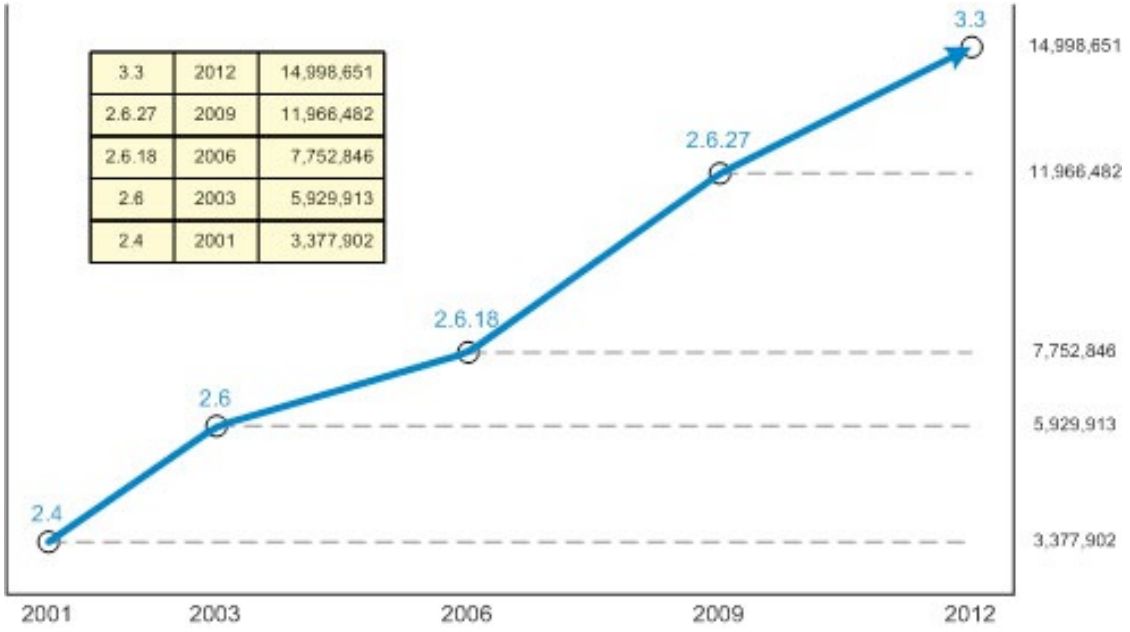
โครงสร้างไดเรกทอรี และขนาดพื้นที่ของลีนุกซ์คอร์เนล 3.2



รูปที่ 2.5 แสดงโครงสร้างไดเรกทอรีของลีนุกซ์เคอร์เนล



รูปที่ 2.6 แสดงรายละเอียดการใช้พื้นที่ของแต่ละไดเรกทอรีของลีนุกซ์เคอร์เนล



รูปที่ 2.7 แสดงการเพิ่มขนาดการใช้พื้นที่ของลีนุกซ์เคอร์เนลตั้งแต่เวอร์ชัน 2.2 ถึง เวอร์ชัน 3.3

ตาราง 2.1 แสดงรายละเอียดโครงสร้างไดเรกทอรีของระบบปฏิบัติการลีนุกซ์ Kernel code

ไดเรกทอรี	รายละเอียด
arch/<architecture>	Architecture specific code
arch/<architecture>/include/asm	Architecture and machine dependent headers
arch/<architecture>/mach-<machine>	Machine/board specific code
block	Block layer code
COPYING	Linux copyright conditions (GNU GPL)
CREDITS	Linux main contributors
crypto/	Cryptographic libraries
Documentation/	Kernel Documentation. Don't miss!
drivers/	All device drivers expect sound ones (usb, pci..)
fs/	Filesystems (fs/ext3/, etc.)
include/	Kernel headers
include/linux	Linux kernel core headers
init/	Linux initialization (including main.c)
ipc/	Code used for process communication
Kbuild	Part of the kernel build system
Kernel/	Linux kernel core (very small!)
lib/	Misc library routines
MAINTAINERS	Maintain of each kernel part. Very Useful!
Makefile	Top Linux Makefile (Set arch and version)
mm/	Memory Management code
net/	Network support codes (not drivers)
README	Overview and Build Instructions
REPORTING-BUGS	Bug report instruction
samples/	Sample codes (markers, kprobes, kobjects)
scripts/	Scripts for internal or external uses
security/	Security Model Implementation (SELinux...)
sound/	Sound support codes and drivers
usr/	Code to generate an initramfs cpio archive.

พื้นฐานการปรับแต่งและสร้าง Custom Kernel

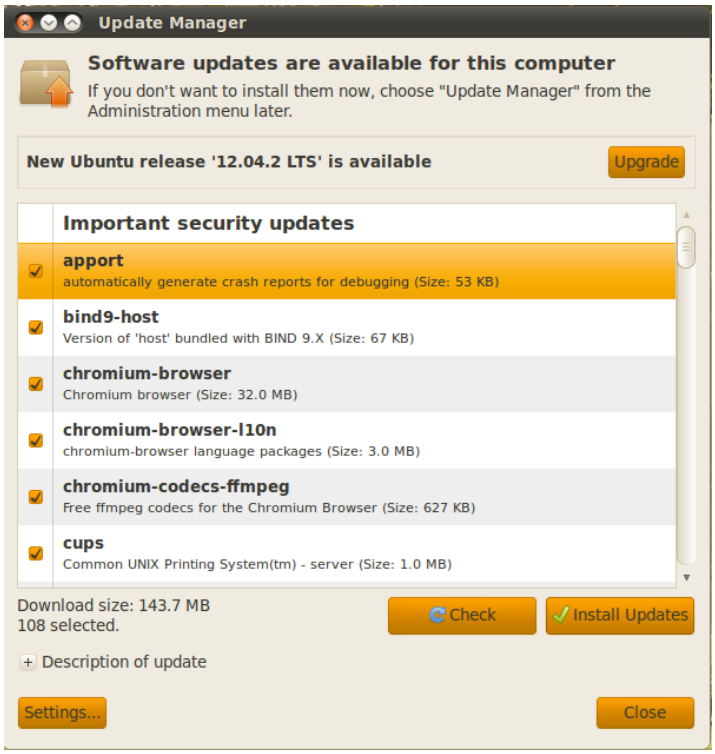
ในการปรับแต่งลีนุกซ์คอร์เนลหรืออัพเดทเป็นรุ่นใหม่เพื่อให้เหมาะสมกับเครื่องที่ใช้งานได้นั้น นักพัฒนาควรตรวจสอบรายละเอียดสถาปัตยกรรมและรุ่นของลีนุกซ์คอร์เนลที่ใช้งานอยู่ภายในเครื่องก่อน โดยใช้คำสั่ง uname ดังตัวอย่างข้างล่าง

```
$ uname -a
Linux EE-Burapha 2.6.32-42-server #96-Ubuntu SMP Wed Aug 15 19:52:20
UTC 2012 x86_64 GNU/Linux
```

รายละเอียดของผลลัพธ์จากคำสั่งข้างต้น:

- ระบบปฏิบัติการที่ใช้อยู่คือ Linux ใช้ Ubuntu Linux distro
- รุ่นลีนุกซ์คอร์เนลคือ 2.6.32
- สถาปัตยกรรม x86_64 (64 บิต)
- ส่วนที่เหลือ ก็จะเป็นครั้งที่และเวลาที่ Kernel ถูกคอมไพล์ตามลำดับ

เมื่อมีการติดตั้งระบบปฏิบัติการลีนุกซ์ไม่ว่าจะเป็นยี่ห้อ Ubuntu, Fedora หรือ FreeBSD ผู้ใช้ไม่จำเป็นจะต้องติดตั้งโปรแกรมหรือปรับแต่งลีนุกซ์คอร์เนลอะไรเพิ่มเติม แต่ในกรณีที่มีลีนุกซ์คอร์เนลรุ่นใหม่ที่จำเป็นต้องทำการอัพเดทหรือมีระบบปฏิบัติการรุ่นใหม่ที่ถูกแนะนำให้อัพเกรด ตัวระบบปฏิบัติการจะมีการแจ้งเตือนให้ผู้ใช้อยู่ภายในอัตโนมัติอยู่แล้ว ดังแสดงในรูปข้างล่าง



รูปที่ 2.8 หน้าต่างโปรแกรม Update Manager

ในกรณีที่ผู้ใช้หรือนักพัฒนาต้องการปรับแต่งลีนุกซ์คอร์เนลก็จำเป็นจะต้องใช้ซอร์สลีนุกซ์คอร์เนลเพื่อมาตั้งค่าต่างๆ ก่อนจะทำการคอมไพล์ใหม่ทั้งหมดด้วยเหตุผลดังนี้

- ต้องการทดสอบความสามารถบางอย่าง ที่ยังไม่มีอยู่ในลีนุกซ์คอร์เนลตั้งแต่แรก หรือเพื่อทดสอบการทำงานของ Linux Module ที่ได้พัฒนาขึ้นมาเอง
- ต้องการเพิ่ม device driver สำหรับอุปกรณ์ฮาร์ดแวร์ตัวใหม่ ที่ยังไม่มีอยู่ในลีนุกซ์คอร์เนลเดิม
- ต้องการเรียนรู้หลักการทำงาน หรือ ดีบักการทำงานของลีนุกซ์คอร์เนลอย่างละเอียด

5 ขั้นตอนพื้นฐานการคอมไพล์ Linux Kernel

ก่อนที่นักพัฒนาจะสามารถปรับแต่งหรือพัฒนาโปรแกรมในระดับล่างภายในลีนุกซ์คอร์เนลได้นั้นผู้เขียนจะขออธิบาย 5 ขั้นตอนพื้นฐานในการเตรียมการคอมไพล์ซอร์สโค้ดลีนุกซ์คอร์เนลที่เป็นพื้นฐานที่จะเชื่อมโยงต่อไปยังบทถัดๆไปของหนังสือเล่มนี้ รวมทั้งจะทำให้ผู้อ่านเข้าใจโครงสร้างที่สำคัญของระบบปฏิบัติการลีนุกซ์ที่จะนำไปสู่ระบบปฏิบัติการลีนุกซ์สำหรับระบบสมองกลฝังตัวต่อไปได้

ขั้นตอนที่ 1 - ดาวน์โหลดลีนุกซ์คอร์เนลรุ่นเสถียรตัวล่าสุด (Latest Stable Linux Kernel)

สามารถตรวจสอบรุ่นของ Kernel และความสามารถใหม่ๆที่ถูกเพิ่มเติมหรือถูกปรับปรุงให้ดีขึ้น ได้จากเว็บไซต์หลัก http://www.kernel.org โดยมีรายละเอียดคำสั่งดังนี้

```
$ sudo su
# cd /usr/src/
# wget https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.9.3.tar.xz
# tar -xvJf linux-3.9.3.tar.xz
```

ขั้นตอนที่ 2 - การตั้งค่าลีนุกซ์คอร์เนล (Kernel Configuration)

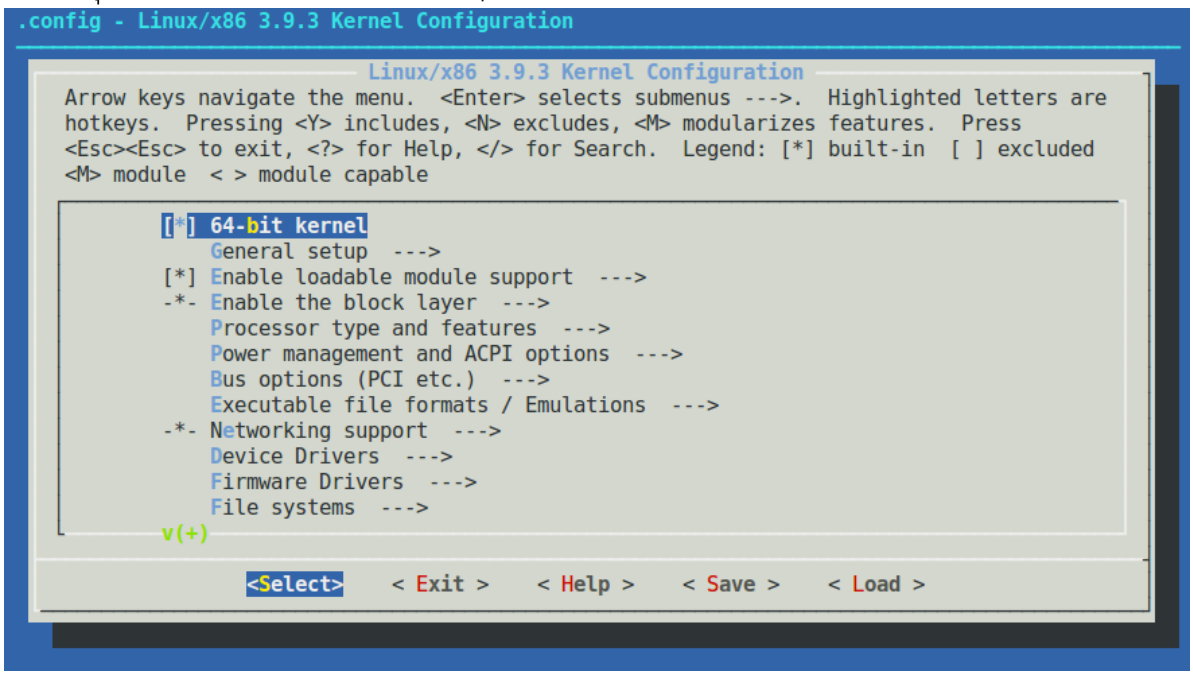
ภายในลีนุกซ์คอร์เนลมีรายละเอียดให้ตั้งค่าได้มากกว่า 3,000 รายการเนื่องจากซอร์สโค้ดลีนุกซ์คอร์เนลนั้นจะบรรจุโปรแกรมต่างๆมากมายเพื่อรองรับผู้ใช้ในแต่ละระดับให้ได้กว้างที่สุดและรองรับอุปกรณ์ฮาร์ดแวร์ต่างๆทั้งในอดีตและปัจจุบันให้ได้มากที่สุดเช่นกัน นอกจากนั้นนักพัฒนาเองก็สามารถพัฒนาโค้ดโปรแกรมใหม่ๆของตนเพิ่มเข้าไปในลีนุกซ์คอร์เนลได้เช่นกัน ก่อนที่จะทำการตั้งค่าในซอร์สโค้ดลีนุกซ์คอร์เนลผู้ใช้จำเป็นต้องติดตั้งไลบรารีพื้นฐานซึ่งได้แก่ ไลบรารี `libncurses` และ `libncurses-devel` ลงในระบบปฏิบัติการลีนุกซ์ที่ตนใช้อยู่ให้เรียบร้อยเสียก่อนด้วยคำสั่ง `sudo apt-get install libncurses libncurses-devel` หลังจากนั้นก็สามารถเข้าสู่การตั้งค่าต่างๆผ่านหน้าต่างในลักษณะเมนู แล้วจึงบันทึกค่า configure ต่างๆลงในไฟล์ชื่อว่า “.config” เพื่อใช้ในการคอมไพล์ต่อไป

วิธีการตั้งค่ามีอยู่หลายแบบเช่นใช้คำสั่ง make config, make menuconfig, make xconfig และ make oldconfig ซึ่งแต่ละคำสั่งมีวัตถุประสงค์และรายละเอียดที่แตกต่างกันไปดังตัวอย่างการใช้งานต่อไปนี้

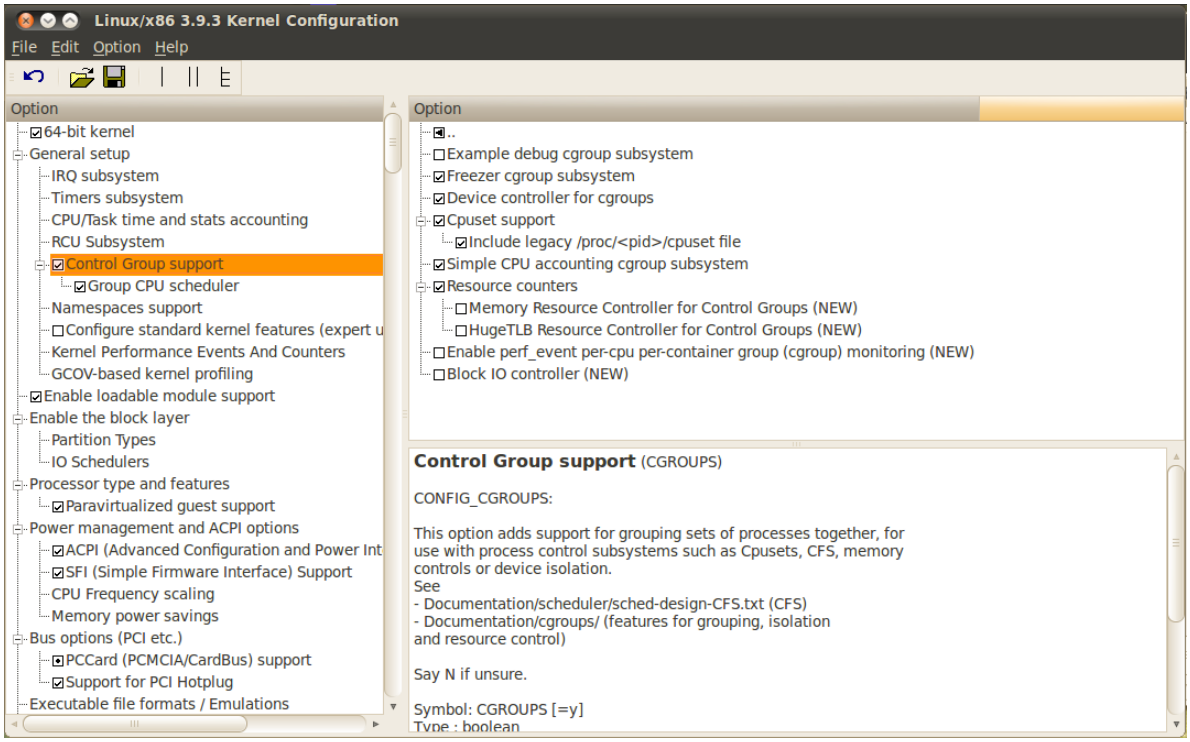
- **make config** คือคำสั่งพื้นฐาน ที่รองรับรุ่นเคอร์เนลได้ตั้งแต่ลีนุกซ์รุ่นแรกๆ ที่ใช้ในการเลือกการตั้งค่า option ต่างๆ โดยผู้ใช้จะต้องตอบคำถามรายละเอียดการตั้งค่าต่างในลักษณะ (y/n)

```
root@EE-Burapha:/usr/src/linux-3.9.3# make config
  HOSTLD  scripts/kconfig/conf
scripts/kconfig/conf --oldaskconfig Kconfig
#
# using defaults found in /boot/config-2.6.32-42-server
#
/boot/config-2.6.32-42-server:532:warning: symbol value 'm' invalid for PCCARD_NONSTATIC
/boot/config-2.6.32-42-server:2854:warning: symbol value 'm' invalid for MFD_WM8400
/boot/config-2.6.32-42-server:2855:warning: symbol value 'm' invalid for MFD_WM831X
/boot/config-2.6.32-42-server:2856:warning: symbol value 'm' invalid for MFD_WM8350
/boot/config-2.6.32-42-server:2857:warning: symbol value 'm' invalid for MFD_WM8350_I2C
/boot/config-2.6.32-42-server:2862:warning: symbol value 'm' invalid for AB3100_CORE
/boot/config-2.6.32-42-server:3314:warning: symbol value 'm' invalid for FB_VESA
/boot/config-2.6.32-42-server:3938:warning: symbol value 'm' invalid for MMC_RICOH_MMC
/boot/config-2.6.32-42-server:4173:warning: symbol value 'm' invalid for COMEDI_PCI_DRIVERS
/boot/config-2.6.32-42-server:4174:warning: symbol value 'm' invalid for COMEDI_PCMCIA_DRIVERS
/boot/config-2.6.32-42-server:4175:warning: symbol value 'm' invalid for COMEDI_USB_DRIVERS
*
* Linux/x86 3.9.3 Kernel Configuration
*
64-bit kernel (64BIT) [Y/n/?]
*
* General setup
*
Cross-compiler tool prefix (CROSS_COMPILE) [] (NEW)
```

- **make menuconfig** คำสั่งนี้เริ่มมีให้ใช้ตั้งแต่ลีนุกซ์คอร์เนลรุ่น 1.3 และรุ่น 2.0 เป็นต้นไป ซึ่งเป็นคำสั่งที่นิยม เนื่องจากใช้งานง่าย มีการแสดงผลคล้ายหน้าต่างในรูปแบบ Text mode และมีการจัดกลุ่มการตั้งค่าออกเป็นประเภทๆ



- **make xconfig** คำสั่งนี้เริ่มมีให้ใช้ตั้งแต่ลีนุกซ์คอร์เนลรุ่น 1.3 และรุ่น 2.0 เป็นต้นไปเช่นเดียวกันคำสั่ง make menuconfig สามารถใช้งานได้ง่ายเนื่องจากมีการแสดงผลเมนูในรูปแบบ Graphic mode บน x-window แต่จะต้องทำการติดตั้งไลบรารีของ qt เพิ่มเติมเข้าไปในระบบเสียก่อน



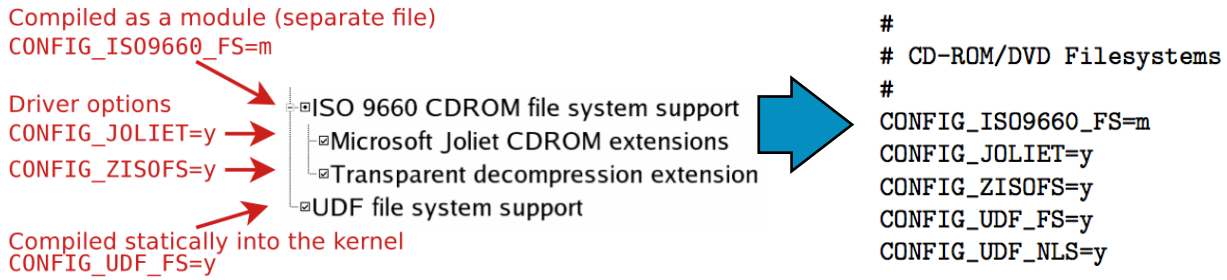
- **make oldconfig** คำสั่งนี้จะอ้างอิงจากไฟล์ .config ที่มีอยู่แล้ว หรือนำมาจากไฟล์ที่บันทึกการตั้งค่า default ของสถาปัตยกรรมที่ต้องการเช่น สถาปัตยกรรม ARM ที่มีให้เลือก System-on-Chip (SoC) จากหลายๆค่ายที่ถูกบรรจุไว้ในไดเรกทอรี **<linux>/arch/x86/configs/** ตัวอย่างเช่น ต้องการใช้กับสถาปัตยกรรม x86 รุ่น i386 ของบริษัท Intel ก็จะใช้คำสั่ง **make i386_defconfig**

สามารถค้นหาไฟล์ defconfig ที่ลีนุกซ์ได้เตรียมไว้ให้เพื่อใช้เป็นไฟล์ .config ต้นแบบสำหรับแต่ละสถาปัตยกรรม ด้วยคำสั่งข้างล่างนี้

```
# find ./arch -name "*_defconfig" -type f
./arch/x86/configs/x86_64_defconfig
./arch/x86/configs/i386_defconfig
./arch/x86/configs/goldfish_defconfig
./arch/xtensa/configs/iss_defconfig
./arch/xtensa/configs/s6105_defconfig
./arch/xtensa/configs/common_defconfig
./arch/cris/configs/etrax-100lx_defconfig
...
# cd linux-3.9.3
# make menuconfig
```

เมื่อได้ตั้งค่าต่างๆเรียบร้อยแล้วจะได้ไฟล์ชื่อ “.config” ออกมาซึ่งบรรจุรายละเอียดของลีนุกซ์คอร์เนลว่าจะมีการใช้หรือไม่ใช้ฟังก์ชันใดบ้าง เพื่อให้เข้ากับสถาปัตยกรรมนั้นๆ

```
root@EE-Burapha:/usr/src/linux-3.9.3# make i386_defconfig
#
# configuration written to .config
#
root@EE-Burapha:/usr/src/linux-3.9.3# cat .config
#
# Automatically generated file; DO NOT EDIT.
# Linux/x86 3.9.3 Kernel Configuration
#
# CONFIG_64BIT is not set
CONFIG_X86_32=y
CONFIG_X86=y
CONFIG_INSTRUCTION_DECODER=y
CONFIG_OUTPUT_FORMAT="elf32-i386"
CONFIG_ARCH_DEFCONFIG="arch/x86/configs/i386_defconfig"
CONFIG_LOCKDEP_SUPPORT=y
CONFIG_STACKTRACE_SUPPORT=y
CONFIG_HAVE_LATENCYTOP_SUPPORT=y
CONFIG_MMU=y
CONFIG_NEED_SG_DMA_LENGTH=y
CONFIG_GENERIC_ISA_DMA=y
CONFIG_GENERIC_BUG=y
CONFIG_GENERIC_HWEIGHT=y
CONFIG_ARCH_MAY_HAVE_PC_FDC=y
CONFIG_RWSEM_XCHGADD_ALGORITHM=y
CONFIG_GENERIC_CALIBRATE_DELAY=y
CONFIG_ARCH_HAS_CPU_RELAX=y
CONFIG_ARCH_HAS_DEFAULT_IDLE=y
```



ขั้นตอนที่ 3 - การคอมไพล์ Linux Kernel

ในปัจจุบันนี้เครื่องคอมพิวเตอร์ส่วนใหญ่จะใช้หน่วยประมวลผลกลาง (CPU) แบบหลายแกน ที่นิยมเรียกกันว่า Multi-core CPU ดังนั้นเพื่อให้การคอมไพล์ลีนุกซ์คอร์เนลเป็นไปได้อย่างมีประสิทธิภาพและใช้เวลาที่เหมาะสมตามทรัพยากรของเครื่องคอมพิวเตอร์ ผู้ใช้สามารถเพิ่มตัวเลือก -j ตามด้วยจำนวนแกนคูณสอง (cores x 2) เช่น **`make -j8`** โดยที่ตัวเลข 8 หมายถึงจำนวนแกนของหน่วยประมวลผลกลางที่อยู่ในเครื่องนั้น (4-core) คูณสอง ซึ่งมี 3 ขั้นตอนย่อยดังนี้

```
# make -j8
```

เป็นการสร้าง Kernel Image เอาไว้สำหรับการบูตเข้าระบบปฏิบัติการซึ่งจะได้ไฟล์มาเป็น bzImage (สำหรับ x86) หรือ zImage (สำหรับ ARM) ซึ่งจะเก็บอยู่ในไดเรกทอรี **`arch/<ชื่อสถาปัตกรรม>/boot/*Image`**

```
# make modules
```

คือการสร้าง Kernel modules ที่ต้องการจะโหลด device drivers ต่างๆ และโมดูลสำคัญของระบบต่างๆ ซึ่งจะมีนามสกุลไฟล์เป็น .ko

```
# make modules_install
```

เป็นคำสั่งในการติดตั้ง Kernel modules ที่ถูกคอมไพล์ออกมา (.ko) เพื่อไปเก็บไว้ในไดเรกทอรี `/lib/modules/3.9.3/`

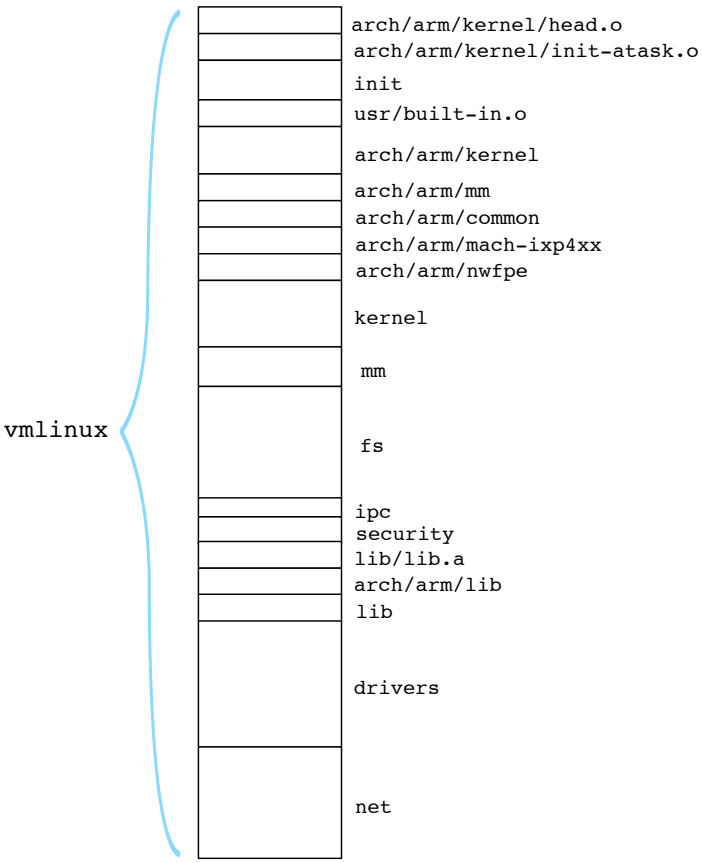
ขั้นตอนที่ 4 - การติดตั้งลีนุกซ์คอร์เนลตัวใหม่

ด้วยคำสั่งข้างล่างนี้จะเป็นการสร้างไฟล์ที่จะใช้ในการบูทเครื่องเพื่อเข้าสู่ระบบปฏิบัติการลีนุกซ์ที่ใช้ลีนุกซ์คอร์เนลรุ่นใหม่ที่ได้มาจากการคอมไพล์ก่อนหน้านี้ ซึ่งไฟล์เหล่านี้จะถูกเก็บไว้ในไดเรกทอรี /boot หลังจากใช้คำสั่งข้างล่าง

```
# make install
```

รายชื่อไฟล์ต่างๆที่ถูกสร้างขึ้นภายในไดเรกทอรี /boot มีดังนี้

- **vmlinuz-3.9.3** (actual kernel)
- **System.map-3.9.3** (symbols exported by the kernel)
- **initrd.img-3.9.3** (initrd image is temporary RFS used during boot process)
- **config-3.9.3** (kernel configuration file)
- **vmlinuz-3.9.3** (actual kernel)



รูปที่ 2.9 รูปแสดงรายละเอียดภายในไฟล์ vmlinux

ตาราง 2.2 แสดงรายละเอียดของไฟล์ไบนารีภายในไฟล์ vmlinux

Component	Description
arch/arm/kernel/head.o	Kernel architecture-specific startup code.
init_task.o	Initial thread and task structs required by kernel.
init/built-in.o	Main kernel-initialization code.
usr/built-in.o	Built-in initramfs image.
arch/arm/kernel/built-in.o	Architecture-specific kernel code.
arch/arm/mm/built-in.o	Architecture-specific memory-management code.
arch/arm/common/built-in.o	Architecture-specific generic code. Varies by architecture.
arch/arm/mach-ixp4xx/built-in.o	Machine-specific code, usually initialization.
arch/arm/nwfpe/built-in.o	Architecture-specific floating point-emulation code.
kernel/built-in.o	Common components of the kernel itself.
mm/built-in.o	Common components of memory-management code.
ipc/built-in.o	Interprocess communications, such as SysV IPC.
security/built-in.o	Linux security components.
lib/lib.a	Archive of miscellaneous helper functions.
arch/arm/lib/lib.a	Architecture-specific common facilities. Varies by architecture.
lib/built-in.o	Common kernel helper functions.
drivers/built-in.o	All the built-in drivers not loadable modules.
sound/built-in.o	Sound drivers.
net/built-in.o	Linux networking.
.tmp_kallsyms2.o	Symbol table.

ขั้นตอนที่ 5 - การบูทระบบปฏิบัติการลีนุกซ์สู่ตัวใหม่

เมื่อมีการติดตั้งลีนุกซ์คอร์เนลตัวใหม่ลงภายในไดเรกทอรี /boot รวมทั้งการเพิ่มเมนูเข้าไปในไฟล์ grub.cfg เป็นที่เรียบร้อยแล้ว ขั้นตอนนี้ก็เพียงบูทเครื่องคอมพิวเตอร์ใหม่เข้าสู่ระบบปฏิบัติการลีนุกซ์อีกครั้งโดยการพิมพ์คำสั่งดังนี้

```
# reboot หรือ shutdown -r now
```

เมื่อเข้าสู่ระบบปฏิบัติการลีนุกซ์แล้ว ผู้ใช้สามารถตรวจสอบรุ่นของ Kernel ตัวใหม่ได้โดยพิมพ์คำสั่ง

```
$ uname -r
3.9.3
```

หมายเหตุ:

ในกรณีที่ต้องการลบไฟล์ต่างๆที่ถูกสร้างในระหว่างที่กำลังคอมไพล์ลีนุกซ์คอร์เนลก็สามารถใช้คำสั่งดังข้างล่างนี้เพื่อลบส่วนที่ไม่ต้องการออก



- **make clean** คือการลบไฟล์ที่ได้เกิดจากการตั้งค่า (./configure) และถูกคอมไพล์ (make) เป็นไบนารีเกือบทั้งหมด
- **make mrproper** คือการลบไฟล์ที่ได้ถูกคอมไพล์เป็นไบนารีทั้งหมด รวมทั้งลบไฟล์ .config
- **make distclean** คือการลบไฟล์ที่ได้เกิดจากการตั้งค่า (./configure) และถูกคอมไพล์ (make) เป็นไบนารีทั้งหมด

คอมไพล์ลีนุกซ์คอร์เนล 3.x สำหรับ UBUNTU ที่ใช้อยู่

ในกรณีที่ผู้ใช้ได้ติดตั้ง Linux distro อย่างเช่น Ubuntu โดยปกติแล้วตัว Ubuntu ตั้งแต่รุ่น 10.x ขึ้นไปจะใช้ลีนุกซ์คอร์เนลรุ่น 2.6.32 เป็นพื้นฐานหลักดังนั้นถ้าต้องการอัพเกรดลีนุกซ์คอร์เนลเป็นรุ่นใหม่ ไม่ว่าจะเป็นรุ่น 3.9.3, 3.8, 3.7, 3.4.6, 3.3, 3.2, 3.1 หรือ 3.0.x แนะนำให้เริ่มต้นที่รุ่น 3.0 เพื่อให้เข้าใจขั้นตอนการปรับแต่ง Kernel อย่างมีหลักการที่ดีพอเพราะถ้าเกิดความผิดพลาดในการตั้งค่าอาจจะทำให้ไม่สามารถเข้าสู่ระบบปฏิบัติการได้อีก เมื่อนักพัฒนาเริ่มมีความเชี่ยวชาญมากขึ้นในระดับนึงแล้วก็ค่อยทำการ patch ให้เป็นรุ่นสูงขึ้นตามที่ต้องการได้

ตัวอย่างข้างล่างจะแสดงการอัพเกรดลีนุกซ์คอร์เนลเป็นรุ่น 3.0.0 เริ่มต้นด้วยการติดตั้งไลบรารี และเครื่องมือที่เกี่ยวข้องในการคอมไพล์ Kernel โดยใช้คำสั่งดังต่อไปนี้ (ในกรณีที่เป็น ubuntu 12.04)

```
$ sudo apt-get install git-core libncurses5 libncurses5-dev libelf-dev
asciidoc binutils-dev linux-source qt3-dev-tools libqt3-mt-dev
libncurses5 libncurses5-dev fakeroot build-essential crash kexec-tools
makedumpfile kernel-wedge kernel-package
```

ดาวน์โหลดลีนุกซ์คอร์เนลจาก kernel.org แล้วแตกไฟล์ไว้ในไดเรกทอรี /usr/src/

```
$ sudo su
# cd /usr/src/
# wget https://www.kernel.org/pub/linux/kernel/v3.0/linux-3.0.tar.xz
# tar -xvJf linux-3.0.0.tar.xz
```

เนื่องจากลีนุกซ์คอร์เนลนั้นมีโค้ดที่รองรับหลากหลายสถาปัตยกรรมและมีรายละเอียดของการตั้งค่าอยู่พอสมควร ดังนั้นเพื่อให้ง่ายและเข้ากันได้กับสถาปัตยกรรมและระบบฮาร์ดแวร์ที่มีอยู่ภายในเครื่อง นักพัฒนาสามารถนำไฟล์ .config เดิมที่อยู่ในเครื่องมาใช้ได้ทันที

```
# cp -vi /boot/config-`uname -r` .config
```

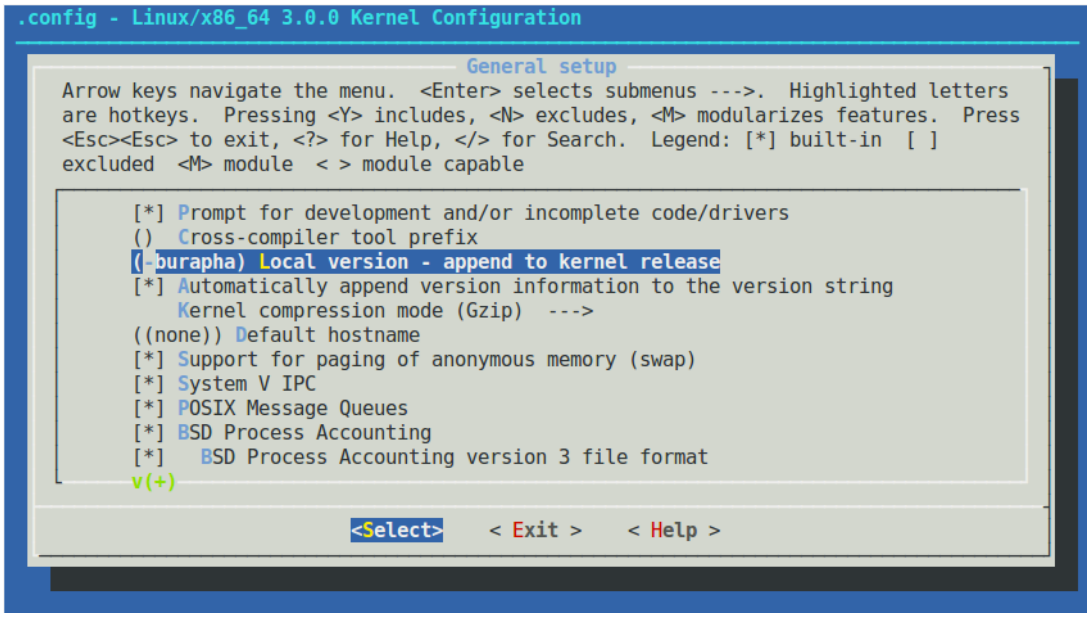
ในกรณีที่มีไฟล์ .config เดิมอยู่แล้วซึ่งได้ถูกตั้งค่าต่างๆให้รองรับระบบฮาร์ดแวร์พื้นฐานของเครื่องเดิม หากต้องการให้รองรับ Linux Kernel Modules เดิมที่อยู่ในรุ่นปัจจุบันด้วย สามารถใช้คำสั่งดังนี้

```
# make oldconfig          ให้กด Enter ไปเรื่อยๆเพื่อใช้ค่า defaults ของ Kernel รุ่นใหม่
# make localmodconfig
```

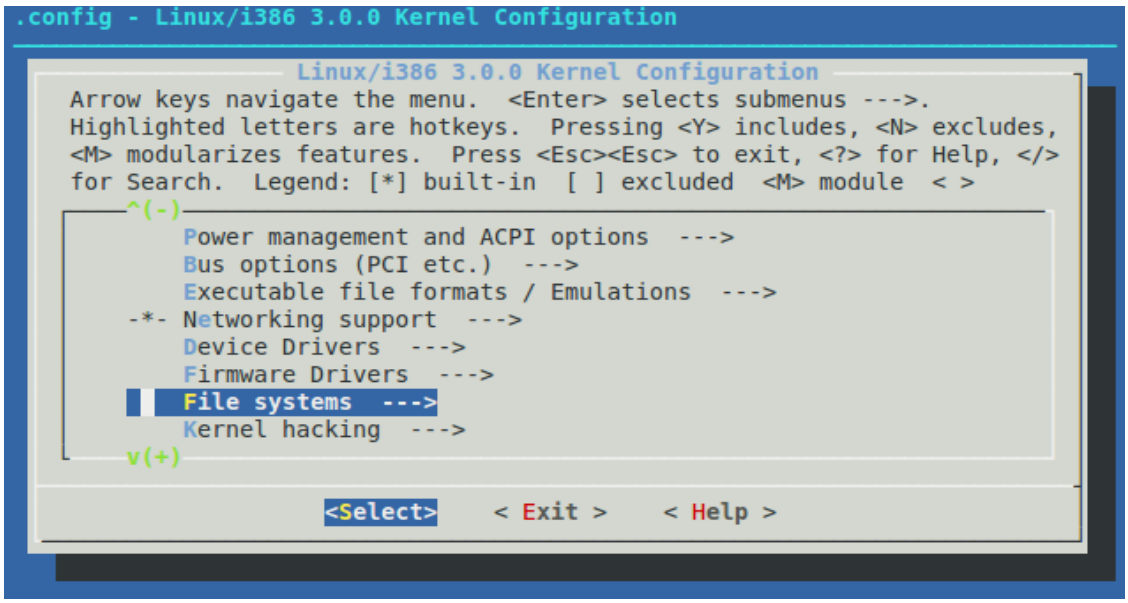
ขั้นตอนถัดไปเป็นการตั้งค่าที่ต้องการเพิ่มเติมของ Kernel รุ่นใหม่ (3.0.0) โดยใช้คำสั่งดังนี้

```
# make menuconfig หรือ make xconfig
```

เพื่อให้ตัวเคอร์เนลตัวใหม่นี้สามารถถูกแยกแยะออกได้จากตัวหลักที่มีในระบบเดิม สามารถระบุ local version ต่อท้ายจากเลขเวอร์ชันเช่น คำว่า “-burapha” โดยกำหนดเพิ่มเติมในเมนู General Setup แล้วเข้าไปใน “Local version” เพื่อใส่ชื่อที่ต้องการ โดยเลือก “Automatically append version info”



และให้เลือกใช้ file system เป็น ext4 โดยเลือกเมนู “File systems”



เริ่มต้นการคอมไพล์ลีนุกซ์คอร์เนลรวมถึง Kernel Modules ที่ได้เลือกไว้ด้วยชุดคำสั่งดังนี้

```
# make -j4
```

```
# make modules_install          <----- ติดตั้ง modules ลงในไดเรกทอรี /lib/modules/
# update-initramfs -u -k 3.0.0-burapha
# make install                  <----- ติดตั้งไฟล์หลักลงในไดเรกทอรี /boot/
```

ผลลัพธ์จากการคำสั่งข้างต้น จะได้ไฟล์หลักดังนี้

- vmlinuz-3.0.0-burapha
- System.map-3.0.0-burapha
- initrd.img-3.0.0-burapha
- config-3.0.0-burapha

ขั้นตอนก่อนที่จะรีบูทเข้าสู่ระบบด้วย Kernel รุ่นใหม่ จะต้องทำการอัพเดทไฟล์ grub.cfg ซึ่งอยู่ภายใต้ไดเรกทอรี /boot/grub/ ด้วยคำสั่ง

```
# update-grub
Generating grub.cfg ...
Found linux image: /boot/vmlinuz-3.0.0-burapha
Found initrd image: /boot/initrd.img-3.0.0-burapha
Found linux image: /boot/vmlinuz-3.0.0-burapha.old
Found initrd image: /boot/initrd.img-3.0.0-burapha
Found linux image: /boot/vmlinuz-2.6.32-42-server
Found initrd image: /boot/initrd.img-2.6.32-42-server
Found memtest86+ image: /boot/memtest86+.bin
done
```

เมื่อเข้าสู่ระบบปฏิบัติการตัวใหม่แล้ว ผู้ใช้สามารถตรวจสอบรุ่นของลีนุกซ์คอร์เนลที่ได้อัพเกรดด้วยคำสั่งข้างล่างนี้

```
$ uname -r
3.0.0-burapha

$ uname -a
Linux EE-Burapha 3.0.0-burapha #2 SMP Sun Aug 4 03:32:15 PDT 2013 x86_64 GNU/
Linux
```

การพัฒนา Linux Kernel Module

ลีนุกซ์คอร์เนลบนเครื่องคอมพิวเตอร์นั้นโดยส่วนใหญ่จะเป็นชนิด monolithic kernel การทำงานภายใน Kernel ของระบบปฏิบัติการลีนุกซ์จะมีการเรียกใช้โมดูลต่างๆที่เรียกกันว่า Linux kernel modules (LKMs) ซึ่งจะถูกนำเข้า-ออกจากหน่วยความจำอยู่บ่อยครั้งตามการใช้งานจริง โดยโครงสร้างไฟล์ LKM (สำหรับ kernel 2.6) ดังแสดงในรูปข้างล่าง

```c
#include <linux/module.h>
#include <linux/kernel.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("EE - Burapha University");
MODULE_DESCRIPTION("hello world test driver");

static int __init module_start(void)
{
    printk(KERN_INFO "Hello world !  module is loaded\n");
    return 0;
}

static void __exit module_unload(void)
{
    printk(KERN_INFO "Goodbye world ! module is removed\n");
}

module_init(module_start);
module_exit(module_unload);
```

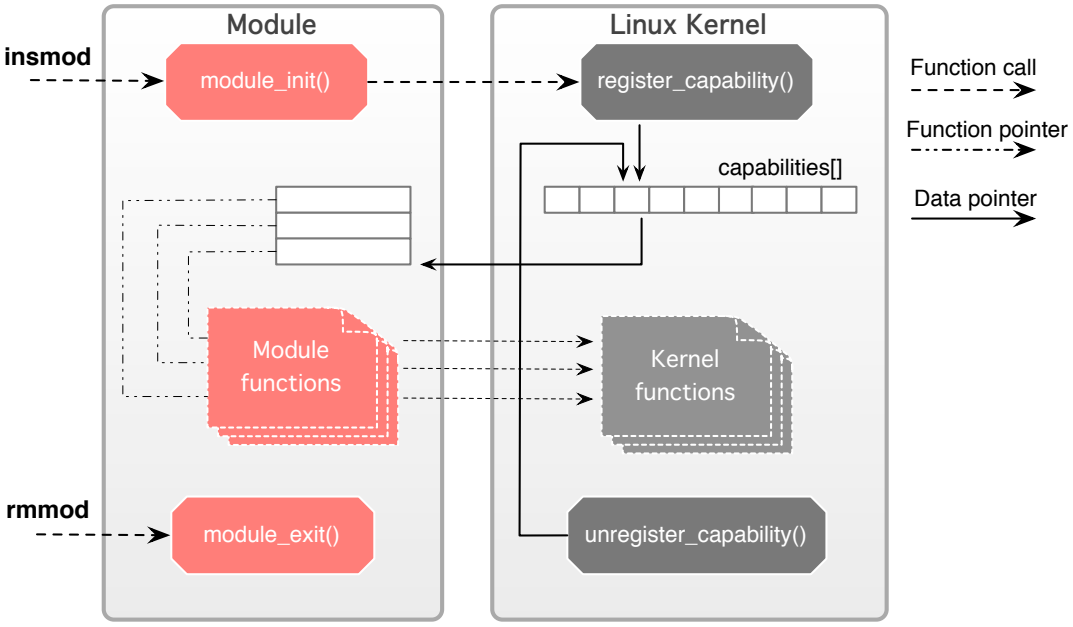
Module macros

Module Constructor/ Destructor

Exit macros

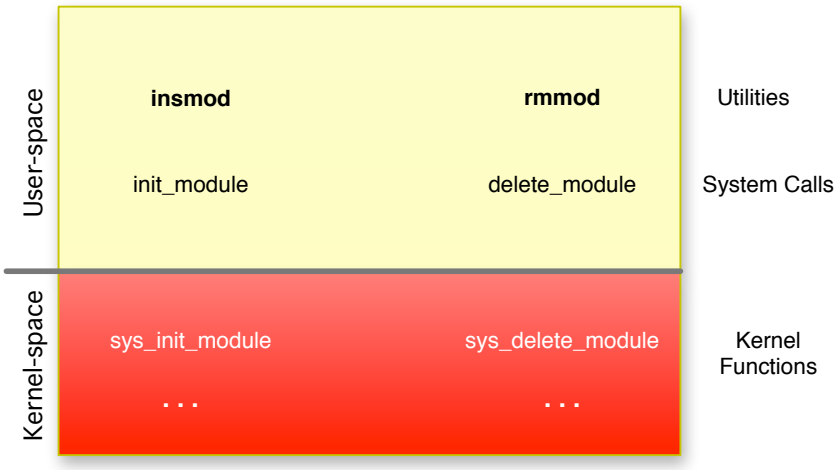
รูปที่ 2.10 ส่วนประกอบภายในไฟล์ Linux Kernel Module

LKM เป็นส่วนประกอบที่สำคัญของลีนุกซ์คอร์เนลซึ่งทำหน้าที่ในการดูแลการร้องขอที่เกิดขึ้นจากฮาร์ดแวร์และซอฟต์แวร์ แล้วทำการประมวลผลกับข้อมูลในเบื้องต้นเพื่อส่งต่อให้ระบบปฏิบัติการสามารถนำรายละเอียดเหล่านั้นไปบริหารจัดการการใช้งานทรัพยากรทั้งหมดได้อย่างมีระบบ ไม่ว่าจะเป็นด้านการจัดการหน่วยความจำ (Memory Management) การติดต่อสื่อสารระหว่างโปรเซส (Inter-processing Communication - IPC) และการจัดลำดับโปรเซส (Scheduling) โดยลีนุกซ์คอร์เนลจะทำการโหลดตัว LKM ขึ้นมาในขณะที่ระบบปฏิบัติการกำลังทำงานอยู่ด้วยคำสั่ง insmod หรือ mod-probe



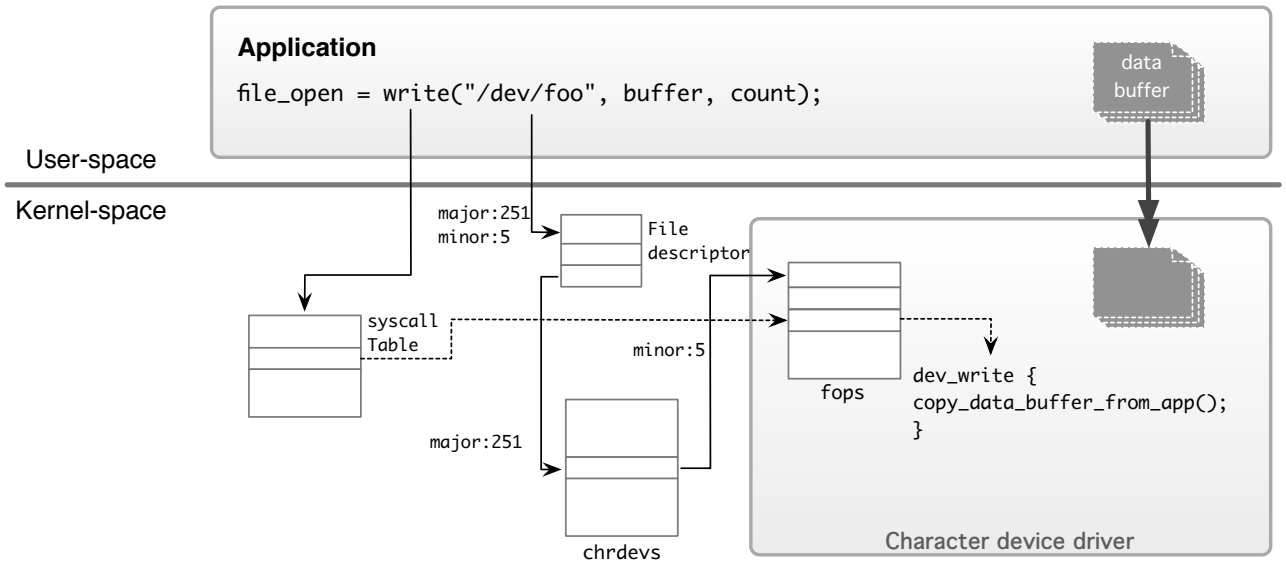
รูปที่ 2.11 รายละเอียดการทำงานภายใน Kernel เมื่อมีการเรียกใช้โมดูล

จากรูปข้างบนเป็นขบวนการการทำงานของ LKM ภายในลีนุกซ์คอร์เนลเวอร์ชั่น 2.6.x โดยมีขั้นตอนดังนี้ เมื่อมีการโหลด LKM จากระดับส่วนของผู้ใช้ (user-space) ด้วยคำสั่ง insmod (**in**sert **mod**ule) ฟังก์ชัน module_init() ก็จะทำการลงทะเบียน (register) กับตัวลีนุกซ์คอร์เนลผ่านฟังก์ชั่นชื่อว่า register_capability() แล้วทำการส่งค่าตัวชี้ (pointer) เพื่อชี้ไปยังตัวแปรอาเรย์ชื่อ “capabilities” หลังจากนั้นระบบปฏิบัติการก็จะทำการกำหนดรูปแบบที่จะให้โปรแกรมประยุกต์นั้น สามารถเข้าถึงข้อมูลภายในของตัวแปร capabilities เพื่อเรียกใช้งานฟังก์ชันภายใน Kernel ต่างๆผ่านทาง system calls ได้ต่อไปจนกระทั่งเมื่อมีการเรียกคำสั่ง rmmod (**rem**ove **mod**ule) จากระดับส่วนของผู้ใช้ (user-space) ฟังก์ชัน module_exit() ก็จะสั่งสิ้นสุดการทำงานแล้วถอนตัว LKM นั้นออกจากลีนุกซ์เคอร์เนล จากรูปข้างล่างเป็นการแสดงการวางลำดับชั้นของฟังก์ชันต่างๆ ที่อยู่ภายในส่วนของผู้ใช้ (user-space) และภายในส่วนของเคอร์เนล (kernel-space)



รูปที่ 2.12 องค์ประกอบของฟังก์ชันที่เกี่ยวข้องกับ Linux Kernel Module

ตัวโปรแกรมไดร์เวอร์ (driver) อาจจะถูกคอมไพล์รวมอยู่ในไฟล์เคอร์เนล โดยเมื่อเริ่มบูทเข้าสู่ระบบปฏิบัติการพวกมันก็จะถูกโหลดขึ้นลีนุกซ์คอร์เนลในระหว่างขั้นตอนของการบูททันที (Boot Processes) เราจะเรียกไดรเวอร์เหล่านั้นว่า Static Module ในขณะที่ไดร์เวอร์บางตัวจะถูกคอมไพล์ให้ทำงานแบบ LKM เพื่อที่จะถูกโหลดหรือถูกนำออกจากลีนุกซ์คอร์เนลเมื่อใดก็ได้ ซึ่งจะเรียกไดร์เวอร์เหล่านี้ว่า Dynamic Module



รูปที่ 2.13 แสดงขั้นตอนการทำงานของโปรแกรมไดร์เวอร์

จากรูปข้างบนแสดงขบวนการทำงานของโปรแกรมไดร์เวอร์ในส่วนของการเขียนข้อมูล (write operation) โดยเมื่อโปรแกรมเรียกฟังก์ชัน write() ก็จะมีการส่งผ่านค่าอินเตอร์รัพท์ (INT instruction) ที่อ้างอิงที่อยู่ของการเขียนข้อมูล (write operation) ที่อยู่ภายในตาราง system call (syscall[ ] table) ในขณะที่ตัวแปร fieldes จะบรรจุค่าหมายเลขอุปกรณ์หลัก (major device number) เพื่อที่จะให้ฟังก์ชัน write ภายในลีนุกซ์คอร์เนลสามารถเข้าถึงภายในโครงสร้างของระบบไฟล์นั้นได้ หลังจากนั้นลีนุกซ์คอร์เนลก็จะคัดลอกข้อมูลที่อยู่ในตัวแปร buffer จากส่วนของระดับผู้ใช้ (user-space) มายังส่วนของระดับเคอร์เนล (kernel-space) เพื่อเตรียมบันทึกลงในตัวเก็บข้อมูลต่อไป

ในทางปฏิบัติโดยทั่วไปแล้วการพัฒนาโปรแกรมไดร์เวอร์จะเป็นลักษณะแบบ Dynamic Kernel module จะมีความยืดหยุ่นสูงและจะใช้หน่วยความจำได้อย่างมีประสิทธิภาพซึ่งไฟล์ LKM ทั้งหมดจะถูกเก็บอยู่ภายใต้ไดเรกทอรี **`/lib/modules/$(uname -r)`** ดังตัวอย่างไฟล์ภายใน ที่แสดงอยู่ด้านล่าง

```
$ ls /lib/modules/$(uname -r)
build                 modules.dep          modules.pcimap
kernel                modules.dep.bin      modules.seriomap
modules.alias         modules.ieee1394map  modules.symbols
modules.alias.bin     modules.inputmap     modules.symbols.bin
```

```
modules.builtin      modules.isapnpmap    modules.usbmap
modules.builtin.bin  modules.ofmap        source
modules.ccwmap       modules.order
```

ส่วนไฟล์ไดร์เวอร์ทั้งหมดก็จะถูกเก็บไว้อยู่ภายใต้ไดเรกทอรี `/lib/modules/$(uname -r)/kernel/drivers/` ลงไปอีกชั้น ดังตัวอย่างไฟล์ภายใน ที่แสดงอยู่ด้านล่าง

```
$ ls /lib/modules/$(uname -r)/kernel/drivers/
acpi         edac      infiniband  mmc         rtc         video
ata          firewire  input       mtd         scsi        virtio
atm          firmware  isdn        net         serial      w1
auxdisplay   gpio      leds        parport     spi         watchdog
block        gpu       md          pci         ssb         xen
bluetooth    hid       media       pcmcia      staging
char         hwmon     memstick    platform    telephony
crypto       i2c       message     power       uio
dca          idle      mfd         pps         usb
dma          ieee1394  misc        regulator   uwb
```

ตัวอย่างการใช้คำสั่ง modprobe และ modinfo ในการดูรายละเอียด และจัดการกับ kernel module

```
$ sudo su
# modinfo lp.ko
filename:       lp.ko
license:        GPL
alias:          char-major-6-*
srcversion:     84EA21D13BD2C67171AC994
depends:        parport
vermagic:       3.0.0-burapha SMP mod_unload modversions
parm:           parport:array of charp
parm:           reset:bool
```

นอกจากนั้นยังมีอีกหลายคำสั่งที่น่าสนใจ ตัวอย่างเช่น คำสั่ง lsmod ที่ใช้ในการแสดงรายการ LKM ที่กำลังถูกโหลดใช้งานอยู่ ณ ขณะนี้

```
$ lsmod
Module                  Size  Used by
binfmt_misc             7568  1
acpiphp                17958  0
snd_ens1371            22663  2
gameport                9869  1 snd_ens1371
snd_ac97_codec        125146  1 snd_ens1371
btusb                  13027  0
ac97_bus                1450  1 snd_ac97_codec
snd_pcm_oss            39673  0
snd_mixer_oss          15604  1 snd_pcm_oss
snd_pcm                87151  3 snd_ens1371,snd_ac97_codec,snd_pcm_oss
snd_seq_dummy           1750  0
snd_seq_device          6594  5
snd_seq_dummy,snd_seq_oss,snd_seq_midi,snd_rawmidi,snd_seq
```

```
bluetooth              94536  1 btusb
snd                    70476  14
snd_ens1371,snd_ac97_codec,snd_pcm_oss,snd_mixer_oss,snd_pcm,snd_seq_oss,snd_
rawmidi,snd_seq,snd_timer,snd_seq_device
intel_agp              11978  1
ppdev                   6504  0
lp                      9893  0
parport_pc             29880  1
serio_raw               4720  0
intel_gtt              16812  1 intel_agp
i2c_piix4               8879  0
soundcore               7860  1 snd
snd_page_alloc          8500  1 snd_pcm
joydev                 10705  0
shpchp                 28186  0
parport                37333  3 ppdev,lp,parport_pc
usbhid                 39022  0
hid                    87524  1 usbhid
floppy                 63772  0
e1000                 103861  0
vmw_pvscsi             15397  0
mptspi                 16441  2
mptscsih               34985  1 mptspi
mptbase                93915  2 mptspi,mptscsih
vmxnet3                39295  0
```

คำสั่ง modprobe (หรือ insmod) ที่ใช้ในการโหลด LKM เข้าสู่ระบบปฏิบัติการ และ modprobe -r (หรือ rmmod) ที่ใช้ในการถอน LKM ออกจากระบบปฏิบัติการ ตามลำดับ

```
$ modprobe ppdev       หรือ   insmod ppdev.ko
$ modprobe -r ppdev    หรือ   rmmod ppdev.ko
```

พื้นฐานการเขียน LINUX MODULE

การพัฒนาโปรแกรมในระดับลีนุกซ์คอร์เนลที่เรียกว่า Linux Module นั้นนักพัฒนาจะต้องเตรียมระบบเพิ่มเติมนอกเหนือจากตัวพื้นฐานที่ติดมากับเครื่อง ตัวอย่างเช่นตัว Ubuntu จะจัดเตรียมโปรแกรมระบบ โปรแกรมประยุกต์ และคอมไพล์เลอร์สำหรับการเขียนโปรแกรมพื้นฐานเท่านั้น แต่จะไม่ได้จัดเตรียมไว้ให้สำหรับการพัฒนาในระดับลีนุกซ์คอร์เนลดังนั้นนักพัฒนาจะต้องทำการติดตั้งซอร์สโค้ดของลีนุกซ์คอร์เนลและติดตั้งไฟล์เฮดเดอร์ (linux header) เพิ่มเติมเข้าไปก่อน เพื่อให้สามารถเรียกใช้ API (Application Programming Interface) ที่เป็นไลบรารีพื้นฐานได้ โดยทำตามคำสั่งข้างล่างนี้

ค้นหาซอร์สโค้ดที่ตรงกับรุ่นของ Ubuntu ที่ใช้

```
$ apt-cache search linux-source*
linux-source - Linux kernel source with Ubuntu patches
linux-source-2.6.32 - Linux kernel source for version 2.6.32 with Ubuntu
patches
$ apt-cache search linux-headers-$(uname -r)
```

```
linux-headers-2.6.32-42-server - Linux kernel headers for version 2.6.32 on
x86_64
```

แล้วทำการติดตั้งด้วยคำสั่งดังนี้

```
$ sudo apt-get install linux-headers-$(uname -r) linux-source-2.6.37
$ cd /usr/src/
$ ls
linux-headers-2.6.32-42-server.tar.bz2  linux-source-2.6.32.tar.bz2
$ tar -xjvf linux-headers-2.6.32-42-server.tar.bz2 linux-
source-2.6.32.tar.bz2
```

ตัวอย่างการสร้างโปรแกรม Kernel Module “Hello world”

เริ่มต้นการพัฒนาโปรแกรม kernel module อย่างง่าย โดยการสร้างไฟล์ชื่อ hello-driver.c แล้วเก็บลงในไดเรกทอรี ~/kernel_module/

```
$ mkdir ~/kernel_module
$ vim hello-driver.c

/*
 *  hello-driver.c - The simplest kernel module.
 */
#include <linux/module.h>     /* Needed by all modules */
#include <linux/kernel.h>     /* Needed for KERN_INFO */
static int __init module_start(void)
{
     printk(KERN_INFO "Hello world !  module is loaded\n");

     /*
      * A non 0 return means init_module failed; module can't be loaded.
      */
     return 0;
}


static void __exit module_unload(void)
{
     printk(KERN_INFO "Goodbye world ! module is removed\n");
}

module_init(module_start);
module_exit(module_unload);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("EE - Burapha University");   /* Who wrote this module? */
MODULE_DESCRIPTION("hello world test driver"); /* What does this module do */
```

ทำการสร้างไฟล์ Makefile สำหรับใช้กำหนดค่าการคอมไพล์ให้กับโปรแกรม hello-driver.c

```
$ vim Makefile

obj-m  = hello-driver.o

all:
      make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
      make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

ใช้คำสั่ง make ในฐานะสิทธิ์ root เพื่อทำการสร้างโมดูล

```
$sudo su
# make
make -C /lib/modules/2.6.32-42-server/build M=/home/student/kernel_module
modules
make[1]: Entering directory `/usr/src/linux-headers-2.6.32-42-server'
  CC [M]  /home/student/kernel_module/hello-driver.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/student/kernel_module/hello-driver.mod.o
  LD [M]  /home/student/kernel_module/hello-driver.ko
make[1]: Leaving directory `/usr/src/linux-headers-2.6.32-42-server'

# ls
hello-driver.c   hello-driver.mod.c  hello-driver.o  modules.order
hello-driver.ko  hello-driver.mod.o  Makefile        Module.symvers
```

ทดสอบโหลดตัวโมดูลด้วยคำสั่ง insmod และสามารถตรวจสอบผลลัพธ์การทำงานด้วยคำสั่ง dmesg (debug kernel message) ดังตัวอย่างข้างล่าง

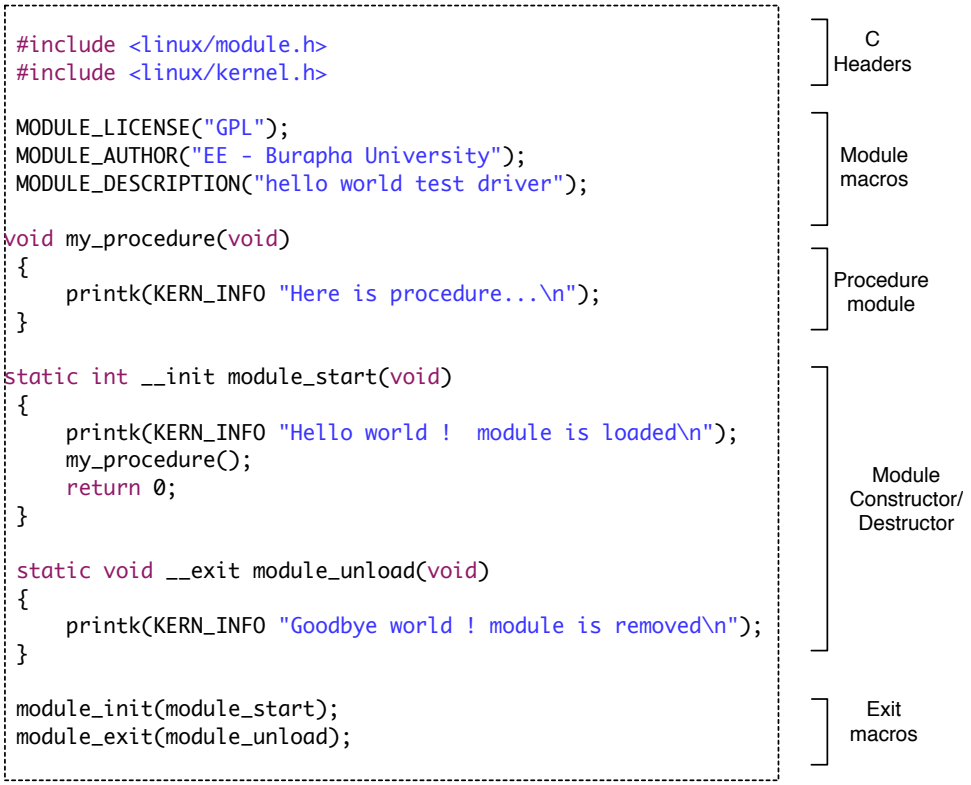
```
# insmod hello-driver.ko
# dmesg | tail -n1
[68587.257467] Hello world !  module is loaded
```

ถอนโมดูลออกจากลีนุกซ์คอร์เนลด้วยคำสั่ง rrmod

```
# rmmod hello-driver.ko
# dmesg | tail -n1
[68621.602044] Goodbye world ! module is removed
```

โครงสร้างภาพรวมของ Linux kernel module

จากรูปโครงสร้างไฟล์ข้างล่างแบ่งได้เป็น 5 ส่วน โดยส่วนที่เป็นโค้ดโปรแกรมหลักจะอยู่ในฟังก์ชั่น init และ exit และจะทำงานก็ต่อเมื่อมีการโหลดหรือถอน LKM ภายในลีนุกซ์เคอร์เนล



รูปที่ 2.14 โครงสร้างแบบเต็มของ Linux Kernel Module

จากโครงสร้างข้างต้นส่วนบนสุดคือส่วนเรียกใช้ไลบรารี (C Headers) โดยจะอ้างอิงไลบรารีหลักๆอย่างน้อยสองตัวคือ `linux/module.h` และ `linux/kernel.h` ส่วนถัดมาคือโปรแกรมย่อย (Procedure Program) ซึ่งเป็นส่วนฟังก์ชันย่อยที่จะถูกเรียกจากโปรแกรมหลัก และส่วนสุดท้ายคือส่วนรายละเอียดของโมดูล (modules description) ที่ใช้อธิบายรายละเอียดของโมดูลที่ออกแบบซึ่งจะถูกแสดงออกมาเมื่อมีการเรียกใช้คำสั่ง **modinfo** ดังตัวอย่างข้างล่าง

```
MODULE_LICENSE("GPL");
MODULE_AUTHOR("EE - Burapha University");   /* Who wrote this module? */
MODULE_DESCRIPTION("hello world test driver"); /* What does this module do */
```

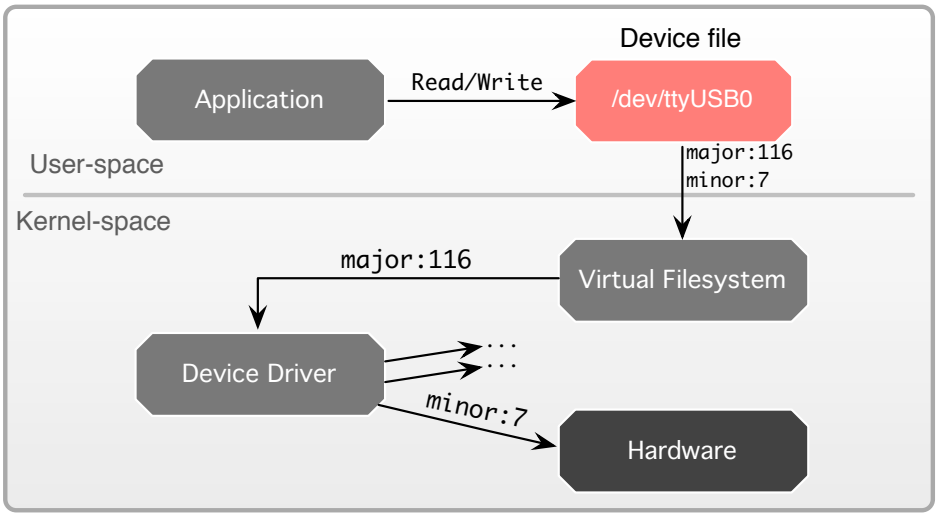
```
$ modinfo hello-driver.ko
filename:       hello-driver.ko
description:    hello world test driver
author:         EE - Burapha University
license:        GPL
srcversion:     E36F5E4D3F232F0A1C18AEF
depends:
vermagic:       2.6.32-42-server SMP mod_unload modversions
```

พื้นฐานการเขียนโปรแกรมไดร์เวอร์สำหรับ CHARACTER DEVICE

พื้นฐานในการเขียนโปรแกรมไดร์เวอร์จะเกี่ยวกับไฟล์ภายในระบบปฏิบัติการลีนุกซ์ตามคุณลักษณะในการส่งข้อมูลของฮาร์ดแวร์นั้นๆ ซึ่งระบบปฏิบัติการลีนุกซ์แบ่งไฟล์ชนิดที่เป็นอุปกรณ์ (device) ออกเป็น 2 ประเภท ได้แก่

1. Character device ซึ่งเป็นการส่งข้อมูลแบบสตรีม (stream) สามารถเข้าถึงได้ทีละเพียง 1 โปรเซสเสมือนโปรแกรมไดร์เวอร์ปกติทั่วไป

2. Block device เป็นการส่งข้อมูลแบบก้อน (block) ใช้กรณีหน่วยประมวลผลกลางต้องการเคลื่อนย้ายข้อมูลขนาดใหญ่ระหว่างหน่วยความจำหรือระหว่างตัวเก็บข้อมูล เช่น ฮาร์ดดิส เป็นต้น



รูปที่ 2.15 แสดงการอ้างอิงค่า major และ minor

จากรูปข้างบนเป็นตัวอย่างการทำงานของโปรแกรมที่ต้องการเข้าถึงไฟล์ /dev/ttyUSB0 ที่ถูกเชื่อมต่ออยู่กับพอร์ต USB ที่มีการส่งข้อมูลในรูปแบบอนุกรม ซึ่งจะสังเกตว่าตัวโปรแกรมจะต้องทำการอ้างอิงหมายเลข (major และ minor) เพื่อระบุประเภทฮาร์ดแวร์หลักและตัวย่อยลงไปที่โปรแกรมต้องการจะติดต่อสื่อสารข้อมูลด้วย

ตัวอย่างโปรแกรม chardev.c จะแสดงจำนวนครั้งที่มีการอ่านข้อมูลจากไฟล์ device ชื่อ /dev/chardev ที่ถูกสร้างขึ้นมา

```
/*
 *  chardev.c: Creates a read-only char device that says how many times
 *  you've read from the dev file
 */

#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <asm/uaccess.h>     /* for put_user */
```

```
/*
 *  Prototypes - this would normally go in a .h file
 */
int init_module(void);
void cleanup_module(void);
static int device_open(struct inode *, struct file *);
static int device_release(struct inode *, struct file *);
static ssize_t device_read(struct file *, char *, size_t, loff_t *);
static ssize_t device_write(struct file *, const char *, size_t, loff_t *);

#define SUCCESS 0
#define DEVICE_NAME "chardev"  /* Dev name as it appears in /proc/devices   */
#define BUF_LEN 80          /* Max length of the message from the device */

/*
 * Global variables are declared as static, so are global within the file.
 */

static int Major;          /* Major number assigned to our device driver */
static int Device_Open = 0;  /* Is device open?
                              * Used to prevent multiple access to device */
static char msg[BUF_LEN];     /* The msg the device will give when asked */
static char *msg_Ptr;
static char msg_data[BUF_LEN] ="NONE \n" ; /*initial msg for write mode */

static struct file_operations fops = {
     .read = device_read,
     .write = device_write,
     .open = device_open,
     .release = device_release
};

/*
 * This function is called when the module is loaded
 */
int init_module(void)
{
        Major = register_chrdev(0, DEVICE_NAME, &fops);

     if (Major < 0) {
       printk(KERN_ALERT "Registering char device failed with %d\n", Major);
       return Major;
     }

     printk(KERN_INFO "I was assigned major number %d. To talk to\n", Major);
     printk(KERN_INFO "the driver, create a dev file with\n");
     printk(KERN_INFO "'mknod /dev/%s c %d 0'.\n", DEVICE_NAME, Major);
     printk(KERN_INFO "Try various minor numbers. Try to cat and echo to\n");
     printk(KERN_INFO "the device file.\n");
     printk(KERN_INFO "Remove the device file and module when done.\n");

     return SUCCESS;
```

```
}

/*
 * This function is called when the module is unloaded
 */
void cleanup_module(void)
{
     /*
      * Unregister the device
      */
      unregister_chrdev(Major, DEVICE_NAME);
        printk(KERN_ALERT "Error in unregister_chrdev:\n");
}

/*
 * Methods
 */

/*
 * Called when a process tries to open the device file, like
 * "cat /dev/mycharfile"
 */
static int device_open(struct inode *inode, struct file *file)
{
     static int counter = 0;

     if (Device_Open)
          return -EBUSY;

     Device_Open++;
     sprintf(msg, "I already told you %d times . Data in device is %s \n", coun-
ter++,msg_data);
     msg_Ptr = msg;
     try_module_get(THIS_MODULE);
     return SUCCESS;
}

/*
 * Called when a process closes the device file.
 */
static int device_release(struct inode *inode, struct file *file)
{
     Device_Open--;          /* We're now ready for our next caller */

     /*
      * Decrement the usage count, or else once you opened the file, you'll
      * never get get rid of the module.
      */
     module_put(THIS_MODULE);

     return 0;
}
```

```
/*
 * Called when a process, which already opened the dev file, attempts to
 * read from it.
 */
static ssize_t device_read(struct file *filp,     /* see include/linux/fs.h   */
                   char *buffer,     /* buffer to fill with data */
                   size_t length,     /* length of the buffer     */
                   loff_t * offset)
{
     /*
      * Number of bytes actually written to the buffer
      */
     int bytes_read = 0;
     /*
      * If we're at the end of the message,
      * return 0 signifying end of file
      */
     if (*msg_Ptr == 0)
          return 0;
     /*
      * Actually put the data into the buffer
      */
     while (length && *msg_Ptr) {
          /*
           * The buffer is in the user data segment, not the kernel
           * segment so "*" assignment won't work.  We have to use
           * put_user which copies data from the kernel data segment to
           * the user data segment.
           */
          put_user(*(msg_Ptr++), buffer++);

          length--;
          bytes_read++;
     }
     /*
      * Most read functions return the number of bytes put into the buffer
      */
     return bytes_read;
}
/*
 * Called when a process writes to dev file: echo "hi" > /dev/hello
 */
static ssize_t device_write(struct file *filp, const char *buff, size_t length,
loff_t * off){

     int bytes_write  ;

     for (bytes_write = 0; bytes_write < length && bytes_write < BUF_LEN;
bytes_write++){
          get_user(msg_data[bytes_write], buff + bytes_write);
      }
```

```
return bytes_write ;

}
```

ทำการสร้างไฟล์ Makefile สำหรับช่วยในการคอมไพล์โปรแกรม chardev.c

```
# cat Makefile
obj-m  = chardev.o

all:
     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

คอมไพล์โปรแกรมเพื่อให้ได้ไฟล์ chardev.ko ออกมาด้วยคำสั่งข้างล่างนี้

```
# make
make -C /lib/modules/2.6.32-42-server/build M=/home/student/kernel_module
modules
make[1]: Entering directory `/usr/src/linux-headers-2.6.32-42-server'
  CC [M]  /home/student/kernel_module/chardev.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/student/kernel_module/chardev.mod.o
  LD [M]  /home/student/kernel_module/chardev.ko
make[1]: Leaving directory `/usr/src/linux-headers-2.6.32-42-server'
# ls
chardev.c      chardev.mod.o   hello-driver.ko     hello-driver.o
modules.order
chardev.ko     chardev.o       hello-driver.mod.c  Makefile
Module.symvers
chardev.mod.c  hello-driver.c  hello-driver.mod.o  Makefile_hello
```

สุดท้ายทำการทดสอบโหลดโมดูลเข้าสู่ลีนุกซ์คอร์เนลด้วยคำสั่ง insmod และครวจสอบผลลัพธ์ด้วยคำสั่ง dmesg ดังข้างล่างนี้

```
# insmod chardev.ko
# dmesg| tail -n6
[70707.121856] I was assigned major number 250. To talk to
[70707.121858] the driver, create a dev file with
[70707.121860] 'mknod /dev/chardev c 250 0'.
[70707.121861] Try various minor numbers. Try to cat and echo to
[70707.121862] the device file.
[70707.121863] Remove the device file and module when done.
```

ทำการทดสอบติดต่อไดร์เวอรตัวนี้อีกครั้ง โดยเริ่มต้นให้สร้างไฟล์แทนไฟล์เดิม (/dev/chardev) เนื่องจากโปรแกรมไดร์เวอร์ chardev.ko ได้สร้างและลบออกไปเรียบร้อยแล้ว ด้วยการสร้างไฟล์อุปกรณ์ตัวใหม่ชื่อ /dev/my_chardev แต่ยังคงให้ใช้หมายเลข major และ minor เดิม (major:250 และ minor:0) ด้วยคำสั่งข้างล่าง

```
# mknod /dev/my_chardev c 250 0
# ls -al /dev/my_chardev
crw-r--r-- 1 root root 250, 0 2013-09-23 22:22 /dev/my_chardev
```

ใช้คำสั่ง cat เพื่อแสดงข้อมูลที่ได้มาจาก /dev/my_chardev

```
# cat /dev/my_chardev
I already told you 0 times . Data in device is NONE
```

จากผลลัพธ์จะสังเกตเห็นว่ายังไม่มีข้อมูลอยู่ใน device เลย ดังนั้นให้ลองทดสอบส่งข้อความไปยังไฟล์ /dev/my_chardev ด้วยคำสั่ง echo ดังตัวอย่างข้างล่าง

```
# echo "Hi...My device" > /dev/my_chardev
# cat /dev/my_chardev
I already told you 2 times . Data in device is Hi...My device
```

การเพิ่ม Linux Module ใหม่เข้าไปยัง Linux Source Tree

เมื่อนักพัฒนาได้พัฒนาโปรแกรมไดรเวอร์หรือ LKM เสร็จเรียบร้อยแล้ว แต่ต้องการรวมโค้ดโปรแกรมของตนเข้าไปอยู่ในโครงสร้างหลักเดียวกับลีนุกซ์คอร์เนลเพื่อให้ผู้ใช้หรือนักพัฒนาคนอื่นๆ สามารถเข้าถึงการตั้งค่าจากคำสั่ง make menuconfig เพื่อเลือกใช้โปรแกรมไดรเวอร์ของเราได้สะดวกขึ้น

ทำได้โดยการนำโปรแกรม LKM ที่ได้พัฒนาขึ้นมานี้ไปเก็บไว้ภายใต้ไดเรกทอรี drivers/misc ของซอร์สลีนุกซ์คอร์เนลในที่นี้ขอยกตัวอย่างไฟล์ hello-driver.c

```
$ cp ~/kernel_module/hello-driver.c /usr/src/linux-source-$(uname -r)
```

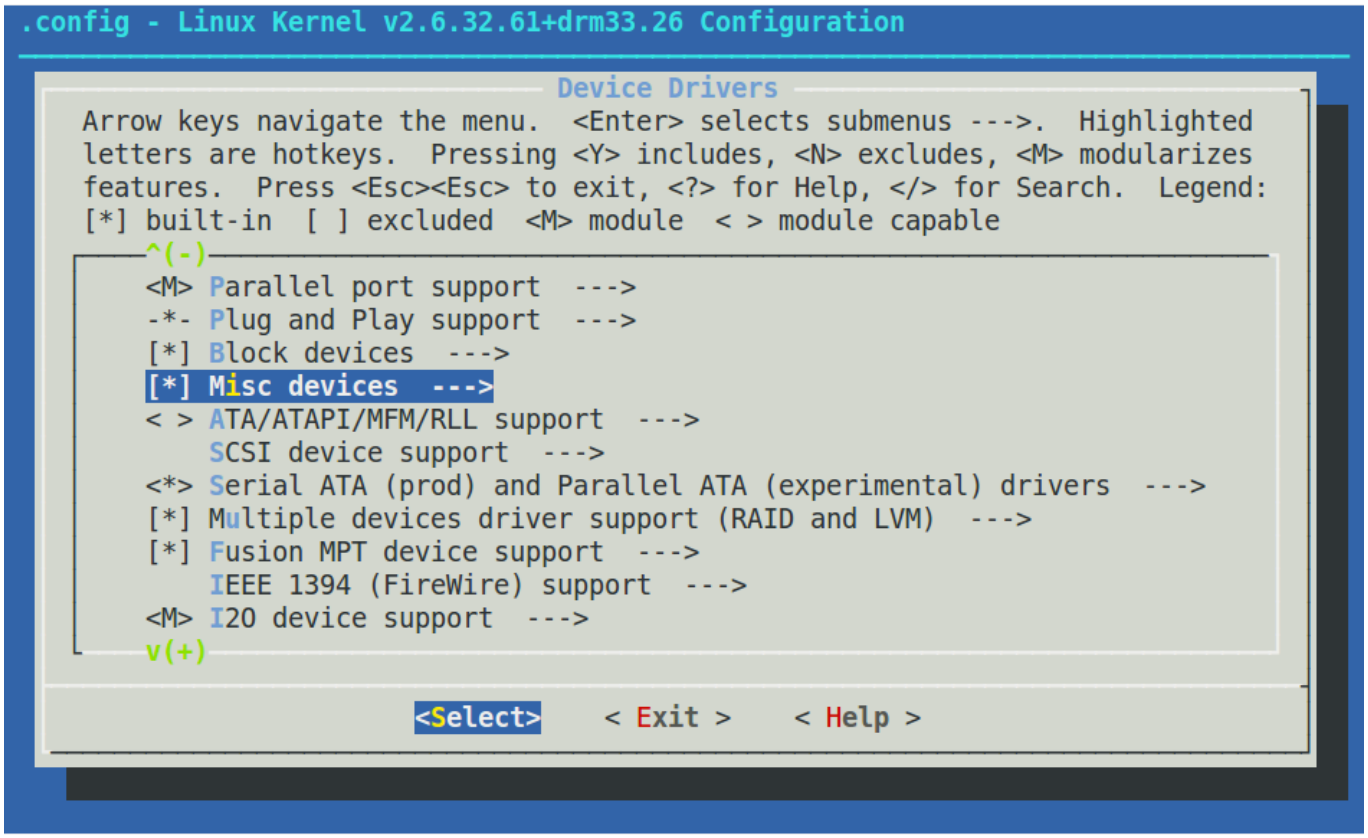
ขั้นตอนถัดไป ให้เปิดไฟล์ drivers/misc/Kconfig เพื่อเพิ่มเมนูเข้าไปในลีนุกซ์คอร์เนลดังตัวอย่างข้างล่างนี้

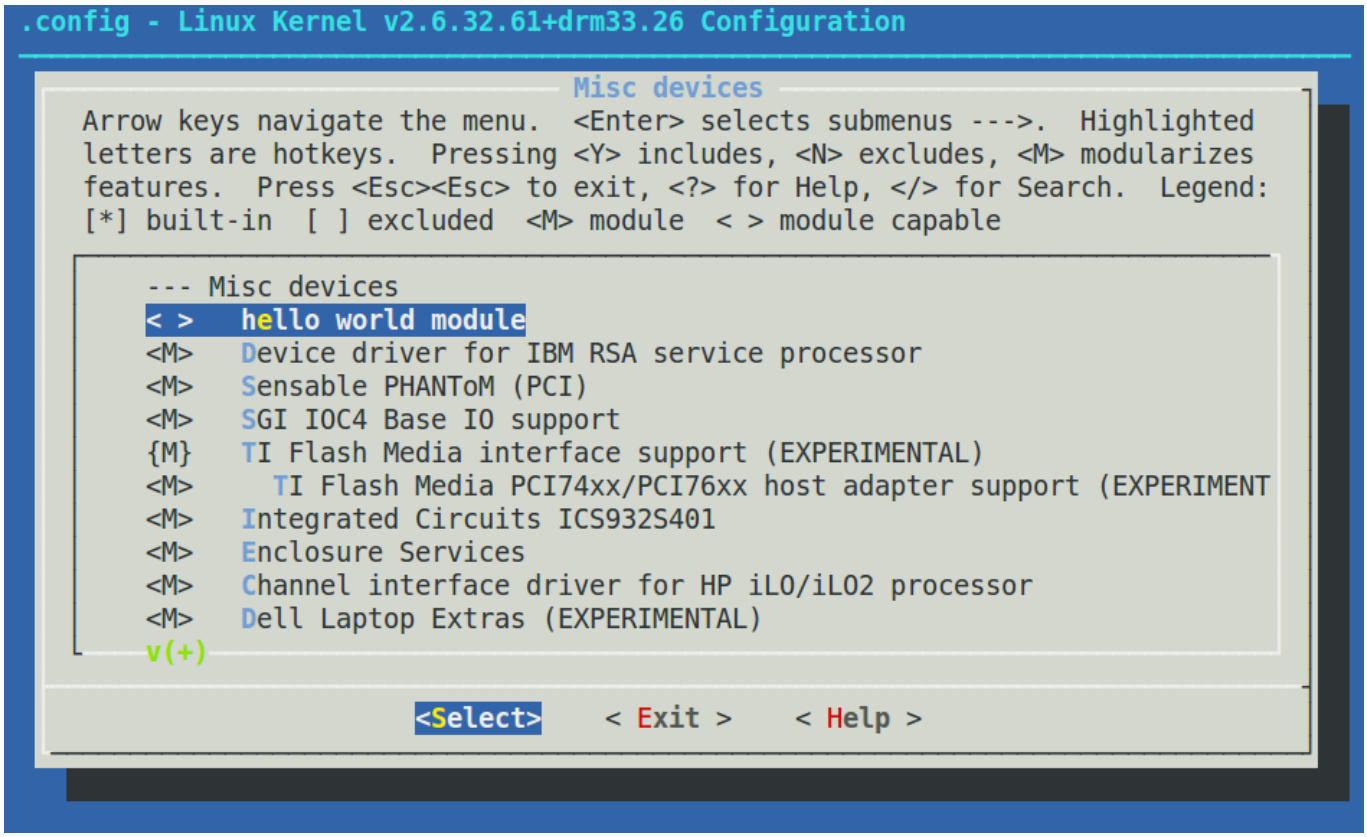
```
config HELLO_WORLD_MODULE
    tristate "hello world module"
    depends on X86  <----- สามารถระบุสถาปัตยกรรมที่โมดูลตัวนี้ทำงานได้เพิ่ม เช่น ARM
    default m if X86  <----- สามารถระบุให้เป็น m หรือ y ได้ และเพิ่มสถาปัตยกรรมได้
    help
```

```
    hello world module.
```

แก้ไขไฟล์ drivers/misc/Makefile โดยการเพิ่มบรรทัดข้างล่างนี้ เพื่อให้มีการคอมไพล์ LKM เพิ่มเติม
`obj-$(CONFIG_HELLO_WORLD_MODULE)+= hello-driver.o`

และให้ทำการเริ่มต้นการคอมไพล์เพื่อสร้างลีนุกซ์คอร์เนลตัวใหม่โดยการเข้าสู่เมนูการตั้งค่าด้วยคำสั่ง make menuconfig แล้วเข้าไปในเมนู Device Drivers เลือกโมดูลที่ต้องการดังตัวอย่างการตั้งค่าข้างล่างนี้





ขั้นตอนสุดท้ายทำการคอมไพล์โมดูลและติดตั้งโมดูลเพิ่มเข้าไปใน root file system ตัวใหม่ ดังคำสั่งข้างล่างนี้

```
$ make modules
$ make modules_install INSTALL_MOD_PATH=$ROOTFS
```

บทที่ 3 EMBEDDED LINUX DEVELOPMENT

ความเป็นมาของระบบสมองกลฝังตัว

ยุคเริ่มต้นของคอมพิวเตอร์ตั้งแต่ปี ค.ศ. 1930 ซึ่งเป็นคอมพิวเตอร์ขนาดใหญ่และสามารถทำการประมวลผลได้เพียงทีละงาน (single task) เท่านั้นแต่เมื่อเวลาผ่านไปการพัฒนาและวิวัฒนาการของเทคโนโลยีสูงมากขึ้น ทำให้ชิฟและอุปกรณ์อิเล็กทรอนิกส์ภายในเล็กลงอย่างต่อเนื่อง จนกระทั่งในปี ค.ศ. 1940 ได้เริ่มมีการนำระบบสมองกลฝังตัวมาควบคุมขีปนาวุธนำวิถี (Missile) สำหรับกองทัพทหาร ซึ่งหนึ่งในระบบสมองกลฝังตัวที่ทันสมัยที่สุดในยุคนั้นก็คือเครื่อง "Apollo Guidance Computer" ถูกสร้างขึ้นโดย Charles Stark Draper ณ แล็ป MIT Instrumentation แต่โปรเจคตัวนี้ยังถูกมองว่ามีความเสี่ยงในการใช้งานเนื่องจากต้องนำวงจรที่เคยมีขนาดใหญ่มากมาย่อให้เล็กลงรวมทั้งน้ำหนักจะต้องเบาลงเพื่อให้สามารถนำไปฝังไว้ภายในผลิตภัณฑ์ที่ต้องการได้ แต่ในที่สุดผลงานชิ้นแรกจากแล๊ป MIT ก็ถูกพัฒนาปรับปรุงขึ้นมาได้จนสำเร็จโดยตั้งชื่อว่า Autonetics D-17 guidance computer และได้ถูกนำไปใช้ในการทหารของโครงการชื่อ Minute missile

ตั้งแต่ช่วงปี ค.ศ. 1961 จนถึงปี ค.ศ. 1966 ได้มีการปรับปรุงการออกแบบของวงจรจากเดิมที่ใช้ตัวทรานซิสเตอร์ขนาดใหญ่และใช้ฮาร์ดดิสภายนอกในการเก็บข้อมูล มาเป็นการนำชิฟประมวลผลแบบใหม่เข้ามาใช้งานแทนที่ตัวทรานซิสเตอร์เดิมจนทำให้ได้วงจรมีขนาดเล็กลงและมีประสิทธิภาพสูงมากขึ้น จนสามารถออกมาเป็นผลิตภัณฑ์ระบบสมองกลฝังตัวรุ่นใหม่ได้ โดยใช้ชื่อว่า Minuteman II

เทคโนโลยีด้านระบบสมองกลฝังตัวได้ถูกพัฒนาอย่างต่อเนื่องเพื่อให้มีขนาดและราคาที่ลดลงแต่ยังคงประสิทธิภาพสูงขึ้น มาจนกระทั่งในปี ค.ศ. 1971 ก็ได้เริ่มเข้าสู่ยุคไมโครโปรเซสเซอร์ยุคแรกอย่างเต็มตัวด้วยการเปิดตัวของหน่วยประมวลผล Intel 4004 ที่ถูกออกแบบขึ้นมาสำหรับเป็นหน่วยประมวลผลกลางในเครื่องคิดเลข แต่ก็ยังคงใช้ฮาร์ดดิสภายนอกในการเก็บหน่วยความจำต่างอยู่ จนมาถึงในปี ค.ศ. 1980 ก็ได้มีการผนวกหน่วยความจำกับตัวชิปประมวลผลเข้าด้วยกันได้สำเร็จซึ่งถือได้ว่าเป็นจุดเริ่มต้นที่สำคัญของวงการไมโครคอลโทรเลอร์ตั้งแต่นั้นเป็นต้นมา โดยตัวไมโครคอนโทรเลอร์ตัวแรกของโลกคือ Intel 8048 และต่อมาในปี ค.ศ. 1980 ก็เป็นจุดเปลี่ยนของวงการไมโครคอลโทรเลอร์อีกครั้งหนึ่ง เมื่อ Intel ได้เปิดตัว Intel 8051 ที่มาพร้อมกับเครื่องมือช่วยในการพัฒนาโปรแกรม (software developments tools) จนได้รับการตอบรับการผู้ใช้อย่างกว้างขวางและเป็นที่นิยมทั้งในระดับมหาวิทยาลัยบริษัทพัฒนาทางด้านระบบสมองกลฝังตัว

แต่ก็ยังมีข้อจำกัดในการพัฒนาโปรแกรมเพราะสามารถเขียนโปรแกรมลงไปในไมโครคอนโทรเลอร์ได้เพียงครั้งเดียว (PROMs) แต่หลังจากนั้นไม่นานก็ได้มีการพัฒนาเพิ่มมาอีกขั้นเพื่อทำให้สามารถลบโปรแกรมภายในแล้วบันทึกได้ใหม่โดยใช้เทคโนโลยี EPROM ที่เป็นการลบด้วยแสงยูวี (UV) และต่อมาภายหลังก็ได้มีการสร้างเครื่องลบโปรแกรมชนิดกล่องสร้างแสง UV ที่สามารถลบข้อมูลใน EPROM โดยใช้เวลาประมาณ 5-10 นาทีจนกระทั่งปี ค.ศ. 1993 ได้มีการพัฒนาจาก EPROM มาเป็น EEPROM ที่

สามารถลบหรือแก้ไขโปรแกรมได้โดยที่ไม่ต้องเอาออกจากคอมพิวเตอร์เหมือน EPROM แต่อย่างไรก็ตาม EEPROM จะต้องลบข้อมูลทั้งหมดก่อนที่จะใส่ข้อมูลใกม่เข้าไปทุกครั้ง ซึ่งยังมีข้อจำกัดอยู่ที่จำนวนครั้งการลบหรือแก้ไข ต่อมาตั้งแต่ปี ค.ศ. 1992 ทั้ง 12 บริษัทยักษ์ใหญ่ทางด้านคอมพิวเตอร์ก็ได้ร่วมกันจัดตั้งกลุ่มสมาคม PC/104 Consortium เพื่อกำหนดแนวทางในการนำเทคโนโลยีเครื่องคอมพิวเตอร์ทั่วไปมาปรับให้เหมาะสมกับการประยุกต์ใช้ในระบบสมองกลฝังตัว โดยได้ออกแบบเมนบอร์ดและปรับขนาดของกล่องให้มีขนาดเล็กกระทัดรัดเพื่อให้สามารถวางซ้อนกันเป็นชั้นได้ด้วยขนาดกว้างยาวเพียง 4 นิ้ว และมีความหนาเพียง 1 นิ้ว ด้วยการนำเสนอของกลุ่มสมาคม PC/104 ทำให้ได้รับการตอบรับจากภาครัฐและภาคเอกชนอย่างคาดไม่ถึง ดังนั้นกลุ่มตลาดทางด้านการทหารและการแพทย์จึงเป็นกลุ่มแรกที่ได้นำแนวคิดและเทคโนโลยีสมองกลฝังตัวไปประยุกต์ใช้งานจริง

จนกระทั่งเทคโนโลยีสมองกลฝังตัวเริ่มได้รับความนิยมอย่างกว้างขวาง และประสิทธิภาพของตัวประมวลผลภายในเองก็เพิ่มขึ้นอย่างรวดเร็วจึงเริ่มนำไปสู่การพัฒนาในอุตสาหกรรมสื่อมัลติมิเดียโดยเริ่มต้นจากการได้สร้างตู้ PC/104-based Kiosk ขึ้นมาเป็นตัวแรก จนถึงวันนี้ได้มีบริษัทมากกว่า 100 แห่งที่เข้าร่วมกลุ่มสมาคม PC/104 Embedded Consortium เพื่อร่วมกำหนดมาตราฐานของผลิตภัณฑ์ตัวอย่างเช่น Ethernet card, FireWire, hard drives, RAM drives, video cards, audio cards, general I/O, flash cards, modems, GPS, cellular telephone, wireless Internet และอีกมากมาย นอกจากนั้นก็ยังมีอีกปัจจัยหนึ่งที่กระตุ้นกลุ่มนักพัฒนาและผู้สนใจเทคโนโลยีสมองกลฝังตัวคือการออกนิตยสารทางด้านระบบสมองกลฝังตัว ซึ่งเริ่มออกสู่สายตานักพัฒนามาตั้งแต่ปี ค.ศ. 1988 ในชื่อว่า “Embedded Systems Programming” และเพิ่งมาเปลี่ยนชื่อเป็น “Embedded Systems Design” เมื่อปี ค.ศ. 2005 จนกระทั่งถึงปัจจุบันนี้



รูปที่ 3.1 นิตยสาร Embedded System Design

ดังนั้นสามารถสรุปนิยามของระบบสมองกลฝังตัวได้ว่า เป็นระบบที่ถูกออกแบบสำหรับใช้งานในวัตถุประสงค์เฉพาะด้านที่ไม่เหมือนกับคอมพิวเตอร์ที่ใช้งานทั่วไป เพราะระบบสมองกลฝังตัวจะถูกกำหนดการทำงานตามความต้องการอย่างชัดเจนไว้ล่วงหน้าเรียบร้อยแล้วซึ่งคุณลักษณะของระบบสมองกลฝังตัวที่สำคัญได้แก่

- เป็นการออกแบบเพื่อทำงานอย่างใดอย่างหนึ่ง โดยจะทำงานตามโปรแกรมที่ตั้งไว้
- เป็นการประมวลผลแบบครั้งละงาน (Single Task)

- สามารถเรียกใช้ได้หลายฟังก์ชันแต่ได้ทีละฟังก์ชันจนสำเร็จเท่านั้น
- ราคาถูก
- ใช้งานง่าย
- มีจุดเชื่อมต่อน้อย ใช้งานง่าย
- มีฟังก์ชันการทำงานที่เพียงพอและรวดเร็ว
- ใช้พลังงานต่ำ
- รองรับการสั่งงานจากภายนอก
- สามารถคำนวณได้แม่นยำและแสดงผลแบบ real-time

ซึ่งในงานที่ไม่ซับซ้อนมากนักระบบสมองกลฝังตัวก็จะใช้เพียงไมโครโปรเซสเซอร์เดี่ยวและโปรแกรมควบคุมก็จะถูกเก็บอยู่ภายในหน่วยความจำชนิดอ่านได้อย่างเดียวหรือเรียกสั้นๆว่า ROM (Read-only Memory) แต่ในกรณีที่ต้องมีการประมวลผลที่ซับซ้อนหลายส่วนพร้อมกันภายในตัวระบบสมองกลฝังตัวจำเป็นต้องมีระบบปฏิบัติการ (Operating System) ฝังอยู่ภายในเพื่อคอยจัดการงานและหน่วยความจำอย่างมีระบบและมีประสิทธิภาพ ตัวระบบปฏิบัติการลีนุกซ์จึงเป็นตัวเลือกที่น่าสนใจที่สุดในกลุ่มนักพัฒนาทางด้านระบบสมองกลฝังตัว ด้วยเหตุผลที่เป็นระบบปฏิบัติการที่มีความยืดหยุ่นและมีประสิทธิภาพสูง สามารถรองรับสถาปัตยกรรมได้ไม่น้อยกว่า 30 สถาปัตยกรรมและเป็นที่รู้กันดีว่าลีนุกซ์คอร์เนลนั้นส่วนใหญ่ถูกเขียนด้วยภาษาซีและแอสเซมบลี้บางส่วน จึงทำให้ง่ายต่อการปรับแต่งและนำไปใช้ในสถาปัตยกรรมได้หลากหลาย นอกจากนั้นในมุมมองการเติบโตและความได้เปรียบของการแข่งขันในตลาดก็ไม่ทำธรรมดาเพราะตั้งแต่ปี ค.ศ. 2005 จำนวนนักพัฒนาลีนุกซ์คอร์เนลก็เพิ่มขึ้นจากเดิมเกือบสามเท่าตัวโดยทุกๆวันจะมีโค้ดใหม่เพิ่มโดยเฉลี่ย 10,000 บรรทัด และทุกๆชั่วโมงจะมีประมาณ 5.45 patch files ที่ถูกอัพโหลดเข้าสู่เว็ปไซต์ลีนุกซ์คอร์เนล (*อ้างอิง: Linux Foundation Analysis*) รวมทั้งผู้ผลิตอุปกรณ์รายใหญ่ๆและผู้จำหน่ายซอฟแวร์ที่ได้รับการรองรับมาตราฐาน (ISV: Independent Software Vendors) จำนวนมากก็ยังหันมาสนับสนุนระบบปฏิบัติการลีนุกซ์กันอย่างจริงจัง และที่สำคัญ GNU GPL (General Public License) ยังให้สิทธิ์แก่ผู้ใช้อิสระในการใช้โปรแกรมการแก้ไขภายในโปรแกรมและคัดลอกแจกจ่ายให้ผู้อื่นอย่างไม่มีจำนวนจำกัดอีกด้วย

ระบบปฏิบัติการลีนุกซ์มีการเติบโตในระดับสูงต่อปีอย่างต่อเนื่องและเริ่มเข้ามามีบทบาทสำคัญในตลาดทางด้านอุตสาหกรรมเทคโนโลยีระบบสมองกลฝังตัวตั้งแต่ปี ค.ศ. 2002 เป็นต้นมา โดยในช่วงแรกตั้งแต่ปี ค.ศ. 2001 ถึง ค.ศ. 2002 ผลการสำรวจระบบปฏิบัติการที่ถูกใช้ใน smart gadgets มากที่สุดคือระบบปฏิบัติการลีนุกซ์ (15.5%) ซึ่งเอาชนะระบบปฏิบัติการ WinCE/WinCE.Net (6%) และ XPe (5%) จากค่าย Microsoft และ VxWorks (10.3%) จากค่าย Wind River ในช่วงเวลาต่อมาตั้งแต่ปี ค.ศ. 2003 ถึง ค.ศ.2005 ผลการสำรวจพบว่าจำนวนผู้ที่ใช้ระบบปฏิบัติการลีนุกซ์ภายในบริษัทเพื่อสร้างผลิตภัณฑ์อย่างต่อเนื่องอยู่ที่ประมาณ 49-52% และหลังจากปี ค.ศ. 2006 เป็นต้นมาถือได้ว่าเป็นช่วงเริ่มต้นของการขยายตัวของระบบสมองกลฝังตัวไม่ว่าจะเป็นการแข่งขันของชิพประมวลผลกลาง (CPU) เช่น ARM, x86, PowerPC, MIPS, Coldfire, DSP เป็นต้น การแข่งขันด้านระบบปฏิบัติการเช่น OpenZaurus,

Ångström distribution, Familiar Linux, webOS from Palm, Inc., Maemo based on Debian เป็นต้น นอกจากนั้นก็ยังมีระบบปฏิบัติการที่คุ้นหูกันดีในวงการนักพัฒนาไทย ที่ฝังอยู่ในอุปกรณ์เครือข่าย อุปกรณ์ชนิดพกพาไม่ว่าจะเป็น PDA, Pocket PC, Smart Phone, Netbook ต่างๆ ตัวอย่างเช่น AirOS (Ubiquiti Network), CatOS/Cisco IOS/IOS-XR (Cisco Systems), DD-WRT (NewMedia-NET), JunOS (Juniper Networks), OpenWRT (LinkSys), Palm OS (Palm Inc.), Symbian OS (Nokia), Windows CE/Mobile (Microsoft), iPhone (Apple), OpenZaurus (Sharp), Maemo (Nokia), Qtopia/Qt Embedded (TrollTech), Openmoko (based on Ångström), Mobilinux (Monta-vista), Android (Google), Moblin (Intel), Ubuntu Mobile และ Meego (Maemo+Moblin) เป็นต้น โดยผู้จัดจำหน่ายที่เน้นขายระบบปฏิบัติการลีนุกซ์สำหรับระบบสมองกลฝังตัวได้แก่ Monta-Vista, Koan, Sysgo, Denx, Metrowerks, FSMLabs, LynuxWorks, Wind River, และ TimeSys เป็นต้น จึงทำให้เทคโนโลยีสมองกลฝังตัวกลายเป็นองค์ประกอบสำคัญในตลาดทางด้านอุปกรณ์อิเล็กทรอนิกส์และมีการใช้ระบบปฏิบัติการลีนุกซ์เป็นระบบปฏิบัติการหลักเช่นเดียวกัน โดยสามารถแบ่งกลุ่มตลาดหลักๆได้ดังนี้

- กลุ่มอุตสาหกรรมยานยนต์ (Automotive) ได้แก่ ระบบควบคุมภายในยานพาหนะ (ECU-Electronic Control Unit) ระบบความปลอดภัยภายใน
- เครื่องใช้ไฟฟ้ารวมถึงอุปกรณ์ Set-top-Box อุปกรณ์สำหรับการสื่อสารและใช้งานผ่านระบบอินเทอร์เน็ต
- ระบบควบคุมอุตสาหกรรม (Industrial Automation) ได้แก่ ระบบควบคุมขบวนการผลิตทั้งหมด ระบบควบคุมหุ่นยนต์ในสายการผลิต รวมไปถึงระบบควบคุมไฟฟ้ากำลังสูง (HVAC System) เป็นต้น
- เครื่องมือแพทย์ ได้แก่ ระบบเฝ้าติดตามอาการผู้ป่วยอัตโนมัติ ระบบบำบัดอาการป่วย ระบบตรวจสอบและวิเคราะห์ผล ระบบช่วยในการผ่าตัด และระบบตรวจวิเคราะห์ด้วยเสียงและภาพ เป็นต้น
- โทรศัพท์เคลื่อนที่ และคอมพิวเตอร์ชนิดพกพา
- เครื่องมือทางทหารและอวกาศ ได้แก่ ระบบนำร่องและควบคุมการบิน ระบบสื่อสารผ่านดาวเทียม
- อุปกรณ์สำนักงาน ได้แก่ เครื่องถ่ายเอกสาร แฟกซ์ เครื่องพิมพ์ เครื่องแสกน โทรศัพท์ VoIP เครื่องสำรองข้อมูลขนาดใหญ่ เป็นต้น
- ระบบสื่อสารแบบสายและไร้สาย ได้แก่ อุปกรณ์เครือข่ายตั้งแต่ระดับ LAN, MAN และ WAN

จากการขยายตัวอย่างสูงของเทคโนโลยีสมองกลฝังตัวไปยังตลาดที่ได้กล่าวมาข้างต้น เกิดจากปัจจัยสำคัญได้แก่

- การเพิ่มจำนวนของนักพัฒนาทางด้านฮาร์ดแวร์และซอฟแวร์ระบบสมองกลฝังตัวมีแนวโน้มสูงขึ้นโดยเฉพาะนักพัฒนาซอฟแวร์

- การขยายตัวของอุปกรณ์สมองกลฝังตัวที่มีอยู่เดิมประมาณสองพันล้านตัวและมีแนวโน้มจะเพิ่มได้ถึงสองแสนล้านตัวภายในปี ค.ศ. 2013
- ระบบปฏิบัติการลีนุกซ์ได้รับการตอบรับและสนับสนุนเป็นอย่างดีจากบริษัทยักษ์ใหญ่ต่างๆ ตัวอย่างเช่น Sony, Motorola, Philips, Panasonic, Google, Samsung และ Siemens เป็นต้น
- ภาษาซียังคงถูกใช้เป็นภาษาหลักในการเขียนโปรแกรมทางด้านฮาร์ดแวร์ (Device Programming) ตามมาด้วยภาษา Assembly, C++ และ embedded C++

อ้างอิงส่วนหนึ่งมาจาก: http://www.ecti-thailand.org/emagazine/views/66

สถาปัตยกรรมในระบบสมองกลฝังตัว



รูปที่ 3.2 แสดงตัวอย่างผลิตภัณฑ์ทางด้านสมองกลฝังตัว *Photo from: http://free-electrons.com*

สถาปัตยกรรมไมโครโปรเซสเซอร์และอุปกรณ์ฮาร์ดแวร์สำคัญของระบบสมองกลฝังตัว

ระบบปฏิบัติการลีนุกซ์ได้ถูกพัฒนาปรับปรุงอย่างต่อเนื่องเพื่อให้สามารถรองรับการสถาปัตยกรรมต่างๆของหน่วยประมวลผลกลางที่ออกมาในตลาดให้ได้มากที่สุดซึ่งรายละเอียดทางด้านเทคนิคภายในโค้ดของลีนุกซ์คอร์เนลสำหรับแต่ละสถาปัตยกรรมจะถูกบรรจุอยู่ภายใต้ไดเรกทอรี /arch ของลีนุกซ์คอร์เนล โดยในปัจจุบันระบบปฏิบัติการลีนุกซ์สามารถรองรับสถาปัตยกรรมต่างๆได้ไม่ต่ำกว่า 30 สถาปัตยกรรม เช่น สถาปัตยกรรม ARM, AVR32, Intel x86, Intel x86_64, MIPS, Motorola 68000 (M68K) , PowerPC, และ Super-H เป็นต้น โดยแต่ละสถาปัตยกรรมจะมีวัตถุประสงค์ในการนำไปที่แตกต่างกันตัวอย่างเช่น

- สถาปัตยกรรม x86 and x86-64 สำหรับเครื่องคอมพิวเตอร์ทั่วไป
- สถาปัตยกรรม ARM จะถูกนำไปใช้ในหลากหลายงานประยุกต์ไม่ว่าจะเป็นอุปกรณ์เครื่องใช้ไฟฟ้า อุปกรณ์โทรศัพท์ (Mobile Internet Device) อุปกรณ์สำหรับงานมัลติมีเดีย (Set-top-Box, DVD Player) หรือ อุปกรณ์ควบคุมในอุตสาหกรรมการผลิต เป็นต้น
- สถาปัตยกรรม PowerPC จะถูกนำไปใช้มากอยู่ในอุปกรณ์ทางด้านโทรคมนาคม ระบบเครือข่าย โดยเฉพาะที่ต้องทำงานแบบเวลาจริง (real-time)
- สถาปัตยกรรม MIPS ยังคงมีใช้อยู่หลากหลายในอุปกรณ์ทางด้านระบบเครือข่าย
- สถาปัตยกรรม SuperH ถูกพัฒนาโดยบริษัท Hitachi (1990s) ในสมัยก่อนจะพบเห็นบ่อยในอุปกรณ์ทางด้านมัลติมีเดีย เช่น อุปกรณ์ Set-top-box จนกระทั่งในปัจจุบันนี้ทางบริษัท Renesus ก็ยังคงนำมาใช้ในระบบนำทาง อุปกรณ์ทางด้านเครือข่าย เครื่องพิมพ์ และกล้องถ่ายรูป
- สถาปัตยกรรม Blackfin ถูกพัฒนาและทำตลาดโดยบริษัท Analog Devices ซึ่งจะถูกเน้นไปในงานด้านประมวลสัญญาณดิจิตอล (Signal Processing) ที่เน้นทำให้ตัว Blackfin CPU เองมีราคาถูก
- สถาปัตยกรรม Motorola 68000 จะได้รับความนิยมในช่วงปี 1980s - 1990s ในอุปกรณ์คอมพิวเตอร์ส่วนบุคคล เช่น Apple Macintosh, Commodore Amiga, Atari ST ซึ่งจะเป็นคู่แข่งสำคัญกับบริษัท Intel

ตาราง 3.1 การรองรับมาตราฐานการเชื่อมต่อของแต่ละสถาปัตยกรรม

OPTION	X86	ARM	PPC	MIPS	SH	M68K
Parallel Port support	X	X		X		
IEEE 1394 support	X	X	X		X	
IrDA support	X	X	X	X		
USB support	X	X	X	X		
Bluetooth support	X	X	X			

หน่วยความจำและตัวเก็บข้อมูล

สำหรับระบบปฏิบัติการลีนุกซ์พื้นฐานที่มีขนาดเล็กที่สุดจะสามารถทำงานได้ภายใต้หน่วยความจำขนาดเพียง 8 MB แต่อย่างไรก็ตามถ้าจะต้องให้สามารถใช้งานประยุกต์ได้ดีในระดับหนึ่งที่ไม่ช้ามากเกินไปก็ควรจะต้องมีขนาดหน่วยความจำให้อยู่ที่ประมาณ 32 MB ส่วนขนาดของตัวเก็บข้อมูล (Storage) นั้นสามารถใช้ขนาดเล็กสุดอยู่ที่ขนาด 4 MB ในกรณีที่ต้องการเก็บหรือสำรองข้อมูลเพิ่มขึ้น ก็อาจจะเพิ่มตัวเก็บข้อมูลประเภทแฟลช (Flash storage) เช่น NAND/NOR Flash หรือ ตัวเก็บข้อมูลประเภทบล็อค (Block storage) เช่น SD/MMC card และ eMMC เป็นต้น

การติดต่อสื่อสารกับอุปกรณ์รอบข้าง

ส่วนของเคอร์เนลในระบบปฏิบัติการลีนุกซ์สามารถรองรับระบบการสื่อสารในมาตราฐานต่างๆได้ ไม่ว่าจะเป็นระบบบัสพื้นฐานในการสื่อสารภายในอุปกรณ์ระหว่างกันเองเช่น I2C, SPI, CAN, 1-wire, SDIO, UART และ USB ระบบการสื่อสารระดับสูงเช่น Ethernet, Wifi, Bluetooth, และ CAN รวมถึงโปรโตคอลมาตราฐานในระดับระบบเครือข่าย (TCP/IP) ก็มีโปรแกรมไดร์เวอร์พร้อมอยู่ภายในตัวระบบปฏิบัติการลีนุกซ์เอง

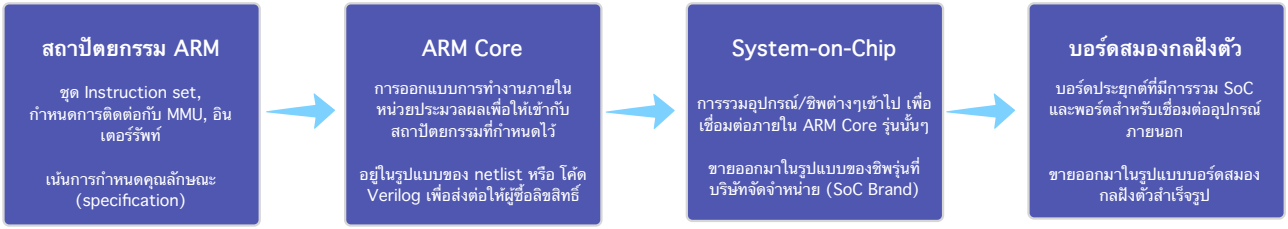
ก่อนจะเป็นบอร์ดสมองกลฝังตัว

ในปัจจุบันนี้บอร์ดสมองกลฝังตัวมีออกมาหลากหลายรุ่นหลายยี่ห้อ ทั้งในราคาที่นักพัฒนาหรือผู้ใช้ทั่วไปสามารถหาซื้อได้ไม่ว่าจะเป็น BeagleBoard, PandaBoard, BeagleBone, ODROID, Raspberry Pi, Freescale i.MX53, AM335x Sitara, Friendly ARM เป็นต้น โดยใช้หน่วยประมวลผลกลาง (CPU) จากผู้ผลิตต่างๆ เช่น Texas Instrument, Samsung, Qualcomm, Nvidia ซึ่งส่วนใหญ่แล้วจะใช้สถาปัตยกรรม ARM อยู่ภายใน



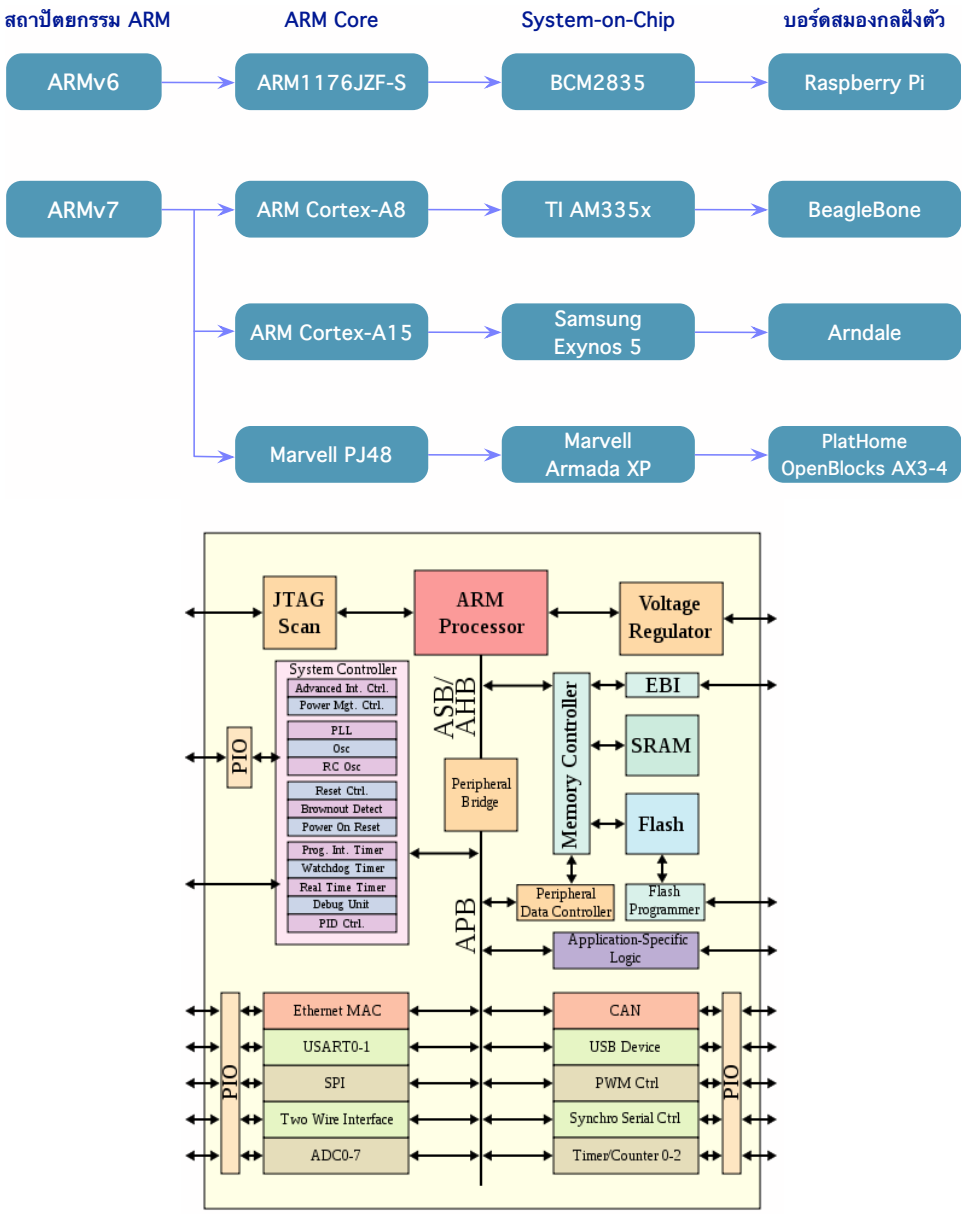
รูปที่ 3.3 แสดงตัวอย่างบอร์ดพัฒนาระบบสมองกลฝังตัว

ซึ่งกว่าจะกลายมาเป็นบอร์ดสมองกลฝังตัวได้นั้นไม่เพียงแต่ต้องหาชิฟต่างๆมาเพื่อประกอบเป็นบอร์ดเพื่อนำไปใช้ในงานประยุกต์แต่นักพัฒนาควรจะต้องเข้าใจฟังก์ชันการทำงานและความสามารถพิเศษที่บอร์ดสมองกลนั้นมีอยู่



รูปที่ 3.4 แสดงลำดับตั้งแต่สถาปัตยกรรมจนมาเป็นบอร์ดสมองกล

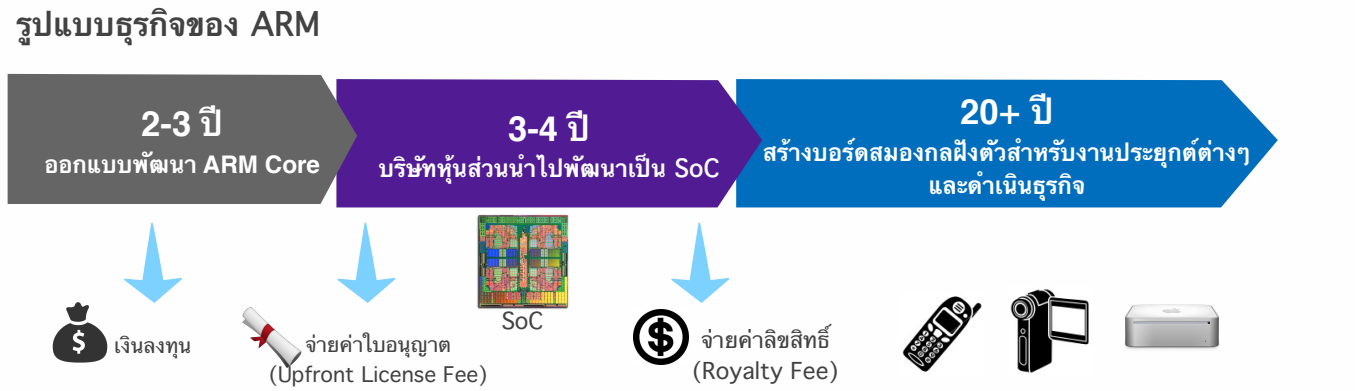
หัวใจสำคัญของส่วนนี้จะเริ่มต้นจากตัว System-on-Chip (SoC) ซีพียูซึ่งมี ARM Core อยู่ภายในที่ถูกเชื่อมโยงด้วยชิพเซ็ทต่างๆ (Chipset) ที่ทำให้บอร์ดสมองกลตัวนั้นสามารถรองรับการสื่อสารได้หลากหลายรูปแบบ ตัวอย่างเช่น Ethernet, AUX, USB, SATA, LCD, HDMI จากรูปข้างล่างแสดงขบวนการตั้งแต่สถาปัตยกรรม ARM จนกลายมาเป็น System-on-Chip ที่บริษัทและเหล่านักพัฒนาสามารถนำมาออกแบบเป็นบอร์ดสมองกลฝังตัวเพื่อนำไปใช้ในงานประยุกต์ต่างๆได้ในที่สุด



รูปที่ 3.5 แสดงตัวอย่าง SoC ที่นิยมอยู่ในปัจจุบัน และตัวอย่างองค์ประกอบภายใน SoC

บริษัท ARM ไม่ได้ทำตัวเองเป็นผู้ผลิตไมโครโปรเซสเซอร์ แต่จะเป็นผู้ที่ออกแบบและกำหนดการทำงานภายในสถาปัตยกรรม เช่นชุดคำสั่ง Instruction Set, การเชื่อมต่อกับตัวจัดการหน่วยความจำ (MMU) หรือการควบคุมสัญญาณอินเตอร์รัพต่างๆ (Interrupts) เป็นต้น เพื่อต้องการขายลิขสิทธิ์ (License fee) การใช้สถาปัตยกรรมแก่บริษัทร่วมทางธุรกิจ (Business Partner) ที่จะร่วมออกแบบกับทางบริษัท ARM จนมาเป็น ARM Core แต่ละรุ่น และจะทำการออกแบบการเชื่อมต่อกับ Chipset ภายใน

ต่างๆ จนในที่สุดกลายมาเป็น System-on-Chip ตัวอย่างเช่น TI OMAP, TI AM335x, Samsung Exynos, Rockchip, FreeScale i.Mx53 โดยบริษัทร่วมธุรกิจนี้จะต้องจ่ายค่า Royalty fee ให้กับบริษัท ARM เมื่อใดที่มีการนำไปสร้างเป็นบอร์ดสมองกลฝังตัวสำหรับสินค้าทางด้านอิเล็กทรอนิกส์และสินค้าเทคโนโลยีต่างๆ เช่น โทรศัพท์มือถือ (Smart Phone), แท็ปเล็ต (Tablet), อุปกรณ์เครื่องใช้ไฟฟ้าหรือระบบอิเล็กทรอนิกส์ภายในยานพาหนะ เป็นต้น ซึ่งจากรูปข้างล่างแสดงโมเดลทางธุรกิจของบริษัท ARM และระยะเวลาในการออกแบบวิจัยพัฒนาจนกว่าจะมาเป็นผลิตภัณฑ์ในตลาดในที่สุด



รูปที่ 3.6 รูปแบบการดำเนินการธุรกิจของ ARM

เริ่มต้นสู่การพัฒนาบนระบบปฏิบัติการลีนุกซ์แบบฝังตัว

ระบบปฏิบัติการลีนุกซ์ที่ถูกฝังเข้าไปอยู่ในบอร์ดสมองกลจะถูกเรียกว่า ระบบปฏิบัติการลีนุกซ์แบบฝังตัว (Embedding Linux) ซึ่งเป็นระบบที่ถูกปรับแต่งเคอร์เนลและโปรแกรมระบบให้สามารถทำงานได้บนสถาปัตยกรรมของหน่วยประมวลผลกลางของบอร์ดนั้นๆ ซึ่งเรียกวิธีการนี้เป็นทับศัพท์ภาษาอังกฤษว่าการ “Porting” นอกจากนั้นในปัจจุบันก็มีบริษัทจำนวนมากที่ขายผลิตภัณฑ์ที่มีระบบปฏิบัติการลีนุกซ์แบบฝังตัวอยู่ภายใน และก็ยังมีอีกหลายกลุ่มบริษัทที่ทำธุรกิจเกี่ยวกับการให้บริการแบบองค์รวมทางด้านการพัฒนาระบบปฏิบัติการลีนุกซ์แบบฝังตัว ไม่ว่าจะเป็นลีนุกซ์คอร์เนลที่ถูกปรับแต่งพิเศษ เครื่องมือการคอมไพล์สำหรับสถาปัตยกรรมต่างๆ (Cross-development tools) และ ไลบรารีทางด้านการทำงานแบบเวลาจริง (Real time extensions) เป็นต้น ถือได้ว่าระบบปฏิบัติการลีนุกซ์นั้นเป็นระบบปฏิบัติการที่เหมาะสำหรับการทำงานทางด้านระบบสมองกลฝังตัวมากที่สุด เนื่องจากมีโปรแกรมไลบรารีต่างๆ มีชุดเครื่องมือในการพัฒนาที่ครบถ้วนเพียงพอสำหรับระบบสมองกลฝังตัว

องค์ประกอบการเตรียมสภาพแวดล้อมสำหรับ EMBEDDED LINUX

องค์ประกอบที่นักพัฒนาหรือผู้สนใจด้านเทคโนโลยีระบบสมองกลฝังตัวควรทำความเข้าใจดังแสดงในตารางข้างล่าง

ตาราง 3.2 องค์ประกอบสำคัญพื้นฐานในการเตรียมสำหรับบอร์ดสมองกลฝังตัว

องค์ประกอบ	รายละเอียด
BSP (Board Support Package)	ภายในบรรจุตัว bootloader และ Kernel ที่รองรับและเข้ากันได้กับรุ่นของฮาร์ดแวร์ที่ใช้ในบอร์ดสมองกลฝังตัวที่เลือกใช้
Cross Toolchains	เครื่องมือพัฒนาที่สามารถแปลงรหัสเพื่อให้ทำงานข้ามแพลตฟอร์มได้
Third-party components	ไลบรารีต่างๆ และโปรแกรมพิเศษจากนักพัฒนาภายนอก ซึ่งอาจจะมีลิขสิทธิ์การใช้งาน เช่น GPLv3, Apache, MIT เป็นต้น

องค์ประกอบ	รายละเอียด
Hardware และ Accessories	- เครื่อง Host คือ เครื่องคอมพิวเตอร์ที่ติดตั้งระบบปฏิบัติการลีนุกซ์ สำหรับปรับแต่งลีนุกซ์คอร์เนลและคอมไพล์โปรแกรมต่างๆสำหรับบอร์ด target - เครื่อง Target คือ บอร์ดสมองกลฝังตัว เช่น Beagleboard, Versatile Express, Pandaboard, ODROID, FriendlyARM เป็นต้น - สาย Cable ต่างๆ เช่น LAN Cable, Serial Cable, USB-to-Serial เป็นต้น - MMC/SD Card

สำหรับขั้นตอนสำคัญของการพัฒนาระบบสมองกลฝังตัวนั้น สามารถแบ่งออกได้เป็น 4 ส่วนหลักคือ

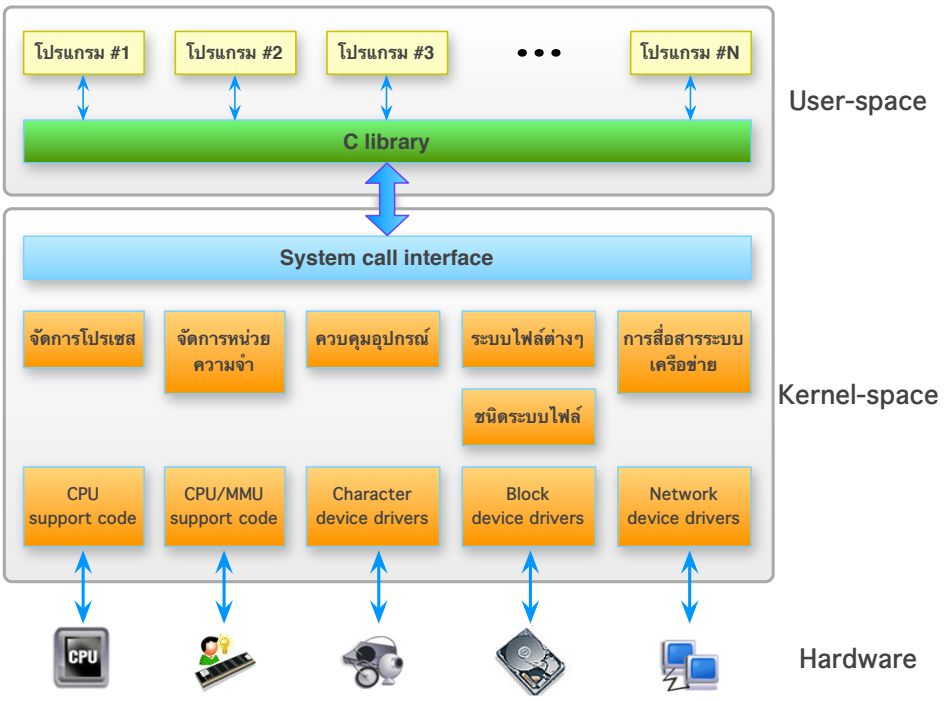
การเริ่มทำงานของบอร์ดสมองกลฝังตัว: เนื่องจากบอร์ดสมองกลฝังตัวจำเป็นต้องมี bootloader ที่มีการตั้งค่าเกี่ยวกับตำแหน่งของ Kernel Image, root file system และค่าพารามิเตอร์ที่จำเป็น เพื่อให้บอร์ดสามารถบูทเข้าสู่ระบบปฏิบัติการลีนุกซ์แบบฝังตัวได้สมบูรณ์แบบ ซึ่งถ้าบอร์ดเชื่อมต่ออยู่กับ Ethernet LAN ก็สามารถเข้าสู่ระบบผ่าน telnet หรือ ssh เข้าไปได้ แต่อย่างไรก็ตามในระหว่างการพัฒนาควรใช้ serial console เพื่อใช้สำรองการเชื่อมต่อในกรณีไม่สามารถต่อ LAN ได้

การตั้งค่าและสร้าง Linux kernel: เนื่องจากบอร์ดสมองกลฝังตัวส่วนใหญ่ถึงแม้จะมีประสิทธิภาพสูงแต่ก็ไม่สามารถใช้คอมไพล์ Kernel ได้ด้วยข้อจำกัดของทรัพยากรภายใน เช่น หน่วยความจำ พลังงานที่จำกัด ดังนั้นงานเหล่านี้ทั้งหมดจะทำอยู่บนเครื่อง Host ก่อนจะ Porting เข้าบอร์ดสมองกลฝังตัว

การตั้งค่าและสร้าง Root File System: ภายใน root file system จะบรรจุโปรแกรมของระบบและโปรแกรมสำหรับผู้ใช้ที่สามารถทำงานกับเคอร์เนลและสถาปัตยกรรมนั้นๆได้ ถ้าปราศจาก root file system แล้วจะเกิดเหตุการณ์แสดงข้อความดังนี้ “Kernel panic” และหยุดทำงานในที่สุด ซึ่งตำแหน่งการวาง root file system นั้นมีอยู่อย่างน้อย 2 แบบคือเก็บไว้ใน Flash Storage เช่น MMC, SDCard และเก็บไว้ในระบบเครือข่ายเช่น NFS

การคอมไพล์และดีบักโปรแกรม: เนื่องจากบอร์ดสมองกลฝังตัวไม่สามารถคอมไพล์ Kernel ด้วยตัวเองได้ จึงจำเป็นต้องทำอยู่บนเครื่อง Host ไม่ว่าจะเป็นการคอมไพล์โปรแกรมด้วยเครื่องมือในการคอมไพล์ที่เรียกกันว่า Cross Toolchain หรือการดีบักโปรแกรมที่กำลังทำงานบนบอร์ดสมองกลฝังตัวด้วย gdb เป็นต้น

โครงสร้างสถาปัตยกรรมของระบบปฏิบัติการลีนุกซ์แบบฝังตัว จะไม่แตกต่างมากเมื่อเทียบกับระบบปฏิบัติการลีนุกซ์ทั่วไป ดังแสดงในรูปข้างล่าง



รูปที่ 3.7 สถาปัตยกรรมภายในระบบปฏิบัติการลีนุกซ์

จากโครงสร้างข้างต้น แต่ละชั้นมีรายละเอียดดังนี้

1. Hardware Layer - อุปกรณ์ฮาร์ดแวร์ภายในบอร์ด ตัวอย่างเช่น หน่วยประมวลผลขนาด 32 บิต ขึ้นไป หน่วยจัดการหน่วยความจำ (MMU - Memory Management Unit) หน่วยความจำขนาดเล็ก (RAM) หน่วยความจำแบบอ่านได้อย่างเดียว (ROM) ตัวบันทึกข้อมูลประเภทแฟลช เช่น MMC/SD Card เป็นต้น
2. Linux Kernel Layer (Kernel-space) - ทำหน้าที่ควบคุมจัดการอุปกรณ์ฮาร์ดแวร์ทั้งหมด จัดการการจัดคิวของโปรเซสต่างๆ จัดเตรียมฟังก์ชันสำคัญ (System Calls) ให้กับโปรแกรมประยุกต์ เป็นต้น ซึ่งจะมีองค์ประกอบภายในที่มีหน้าที่สำคัญๆ ได้แก่
   - *Low-level interfaces*:
     - จัดเตรียมชุด APIs (Application Programming Interface) สำหรับการติดต่อกับฮาร์ดแวร์แบบที่ไม่ขึ้นอยู่กับรุ่นของฮาร์ดแวร์
     - เป็นส่วนดูแลและจัดการฮาร์ดแวร์ต่างๆเช่น การทำงานของหน่วยประมวลผลกลาง การทำงานของหน่วยความจำ ติดต่อกับพอร์ตสื่อสาร และ I/O Devices เป็นต้น
   - *High-level abstractions:*
     - จะจัดเตรียมฟังก์ชันให้โปรแกรมประยุกต์สามารถสื่อสารระหว่างกันได้ และทำงานแบบ multi-tasking ได้ เช่น IPC, Threading, Socket, Signalling เป็นต้น
     - โค้ดโปรแกรมส่วนใหญ่สามารถนำไปใช้ต่อได้เกือบทุกสถาปัตยกรรม
   - *Filesystems:*
     - สามารถรองรับระบบ filesystems ต่างๆที่มีอยู่ในปัจจุบันได้ไม่น้อยกว่า 40 ชนิด

- จะมี Virtual Filesystem layer ที่ทำให้การเรียกใช้ filesystems ที่แตกต่างเหล่านั้นง่ายและสะดวกขึ้นโดยที่ไม่ต้องเขียนโค้ดใหม่

• *Networking protocols*

- รองรับโปรโตคอลมาตราฐานต่างๆที่ใช้กันอยู่ในปัจจุบันได้ไม่น้อยกว่า 50 ชนิด
- มีฟังก์ชัน Socket APIs ที่ใช้ในการจัดการและกำหนดการเชื่อมต่อในแต่ละแบบของระบบเครือข่ายนั้นๆ

3. Libraries (User-space) - ไลบรารีที่อยู่ในระดับของ user space ที่มีความหลากหลายมากกว่าที่มีอยู่ในเคอร์เนล ไม่ว่าจะเป็นไลบรารีทางด้านการคำนวณ การแสดงผลกราฟฟิก การประมวลผลข้อมูลดิจิตอล การสื่อสารข้อมูลผ่านระบบเครือข่าย การเข้ารหัส/ถอดรหัส และ การติดต่อกับอุปกรณ์ฮาร์ดแวร์ที่มาเชื่อมต่อกับบอร์ด เป็นต้น ดังนั้นถ้าโปรแกรมประยุกต์ต้องการจะติดต่อกับเคอร์เนลจะเรียกผ่านไลบรารีชื่อว่า glibc (ซึ่งประกอบไปด้วยตัวย่อยคือ uClibc และ eglibc)

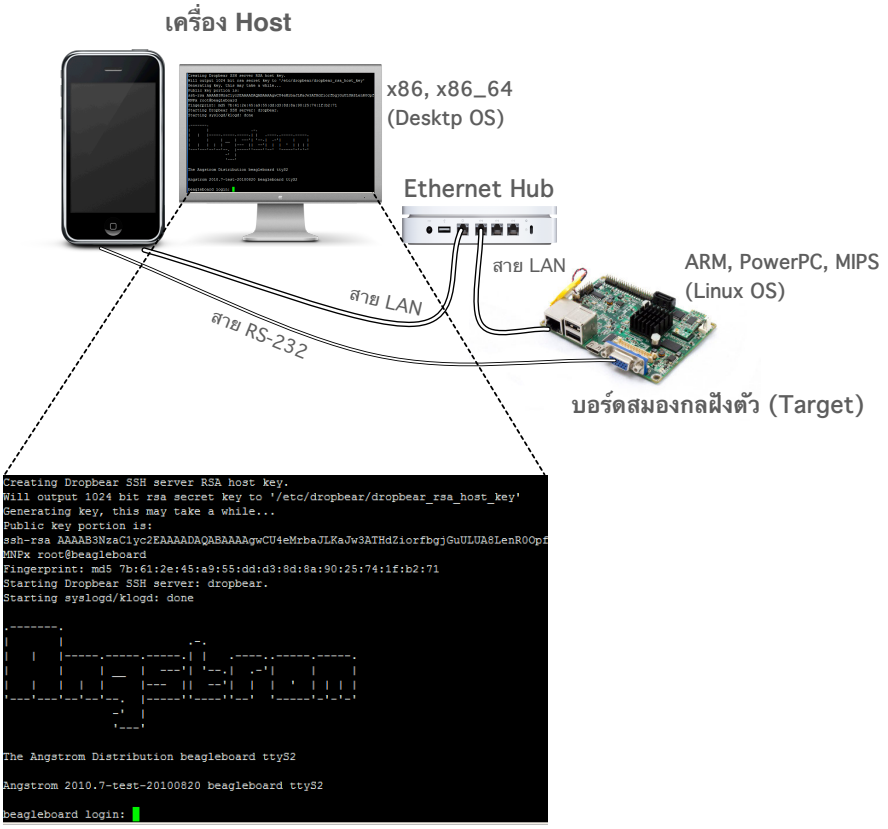
การเตรียมระบบและการเชื่อมต่อระหว่างเครื่อง HOST และ บอร์ด TARGET

การเตรียมระบบสำหรับการพัฒนาโปรแกรมหรือระบบปฏิบัติการสำหรับระบบสมองกลฝังตัวนั้นจะต้องมีการเตรียมระบบฮาร์ดแวร์ ซึ่งประกอบไปด้วยฮาร์ดแวร์ 2 ส่วนหลักๆได้แก่

1. เครื่องคอมพิวเตอร์สำหรับพัฒนาหรือเรียกว่าเครื่อง “Host” (ซึ่งก็เป็นเครื่องคอมพิวเตอร์ส่วนบุคคลทั่วไป)
2. บอร์ดสมองกลฝังตัวหรือเรียกว่าบอร์ด “Target”

รูปแบบการเชื่อมต่อกันระหว่างเครื่อง Host และบอร์ด Target นั้นมีอย่างน้อย 3 แบบคือ

- สายอนุกรม RS-232 (RS-232 Serial Cable) เพื่อให้นักพัฒนาสามารถดูรายละเอียดการบูทของบอร์ด ในขณะที่เริ่มตรวจสอบค่าในระดับ low level การโหลดตัว Kernel และการโหลดตัว root file system เรียกหน้าจอแสดงผลนี้ว่า “Console”
- สาย LAN (Ethernet LAN Cable) เพื่อให้นักพัฒนาสามารถวาง Kernel Image และ Root file system ไว้บนเครื่องภายในระบบเครือข่าย ผ่านโปรโตคอล bootp, tftp หรือ NFS เป็นต้น ในกรณีที่บูทเข้าสู่ระบบปฏิบัติการลีนุกซ์แบบฝังตัว (Embedded Linux) ภายในบอร์ดได้แล้ว ถ้าไม่ต้องการดูผ่านหน้าจอ Console ก็สามารถเข้าสู่ระบบแบบระยะไกล (Remote Login) ผ่านโปรแกรม telnet หรือ ssh ได้เช่นกัน
- สาย JTAG (JTAG Cable) ใช้ในกรณีที่นักพัฒนาต้องการดีบักการทำงานของโปรแกรม ค่ารีจิสเตอร์และรายละเอียดภายในหน่วยความจำของบอร์ดสมองกล เป็นต้น



รูปที่ 3.8 แสดงการต่อสำหรับการพัฒนาระหว่างเครื่อง Host และบอร์ด Target

จากรูปด้านบนแสดงการเชื่อมต่อระหว่างเครื่อง Host และบอร์ด Target โดยภายในเครื่อง Host จะมีระบบปฏิบัติการลีนุกซ์ (เช่น Ubuntu, Red Hat, Fedora, SuSE หรือ Debian เป็นต้น) และเครื่องมือหรือโปร แกรมอัตถประโยชน์ (เช่น Serial Console Software, Cross Toolchain, Network sharing Software) สำหรับการพัฒนาระบบสมองกลฝังตัวอยู่ภายในโดยถูกเชื่อมต่อผ่านสายอนุกรม (RS-232 Cable) เรียบร้อยแล้ว ซึ่งในปัจจุบันเครื่องคอมพิวเตอร์หรือโน๊ตบุ๊คส่วนใหญ่จะไม่มีพอร์ตอนุกรมให้อีกแล้ว แต่จะมีการเตรียมพอร์ต USB มาให้แทน ดังนั้นนักพัฒนาจะต้องจัดหาสาย USB-to-Serial มาเพิ่มเติมเพื่อให้สามารถต่อเข้ากับบอร์ด Target รวมทั้งจะต้องเชื่อมต่อระบบเครือข่ายภายในเข้ากับอุปกรณ์เครือข่ายเช่น Ethernet Hub หรือ Switch Hub

ก่อนเริ่มบูทบอร์ดสมองกลฝังตัวและเห็นการแสดงผลต่างๆผ่านหน้า Serial Console บนเครื่อง Host ได้นั้นนักพัฒนาจะต้องมีโปรแกรม Serial Console ตัวอย่างเช่นโปรแกรม Minicom, Picocom, Gtkterm, Putty, Cutecom ที่สามารถติดต่อกับพอร์ตอนุกรม (Serial Port) หรือพอร์ต USB โดยระบบปฏิบัติการลีนุกซ์จะมองเห็นเป็น /dev/ttyS*x* หรือ /dev/ttyUSB*x* ยกตัวอย่างการใช้โปรแกรม minicom และ Gtkterm ดังข้างล่างนี้

```
$ minicom -b 115200 /dev/ttyS0        หรือ
$ minicom -b 115200 /dev/ttyUSB0
```

```
File  Edit  View  Terminal  Help

OPTIONS: I18n
Compiled on Jan 25 2010, 06:49:09.
Port /dev/ttyUSB0

Press CTRL-A Z for help on special keys

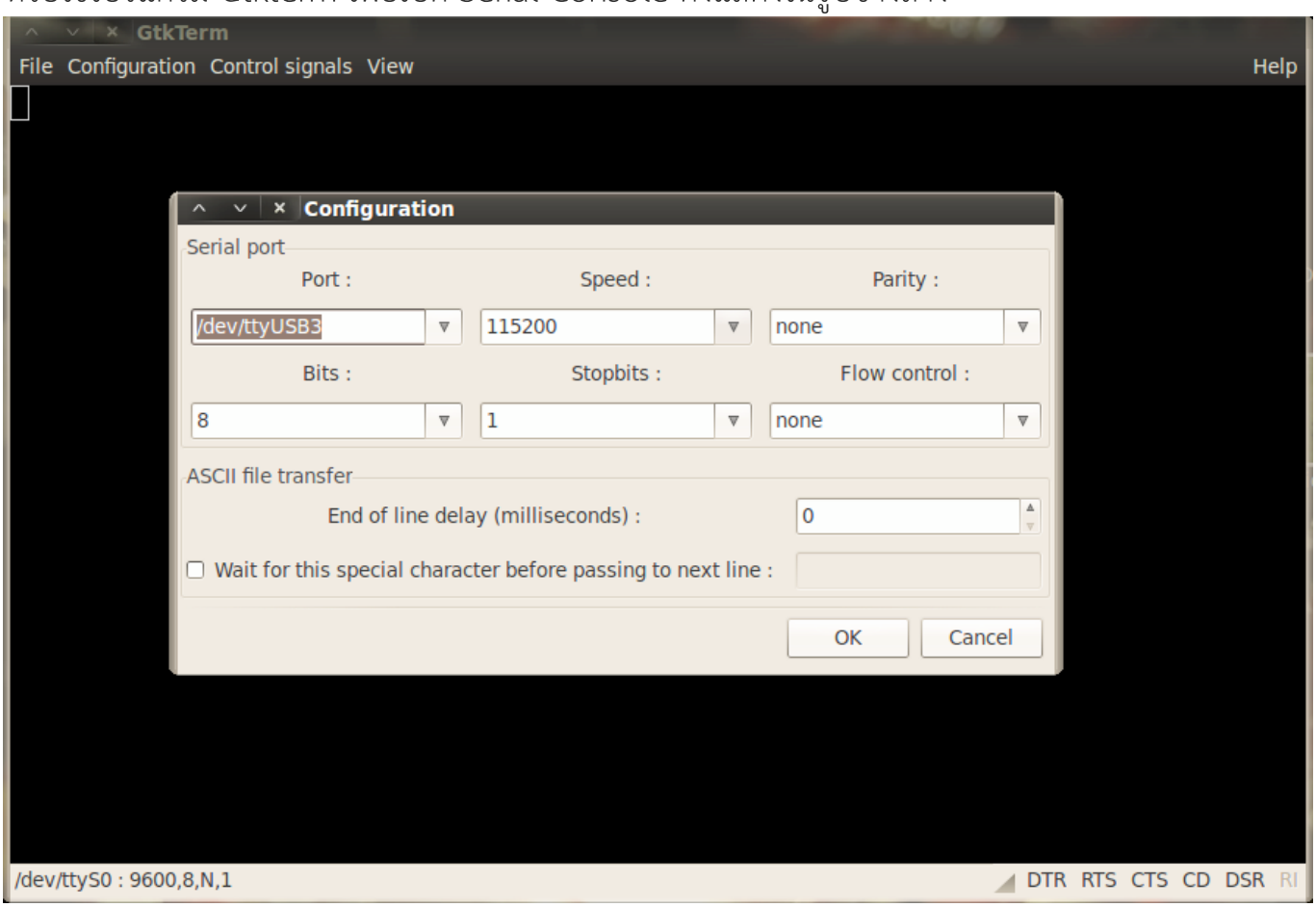


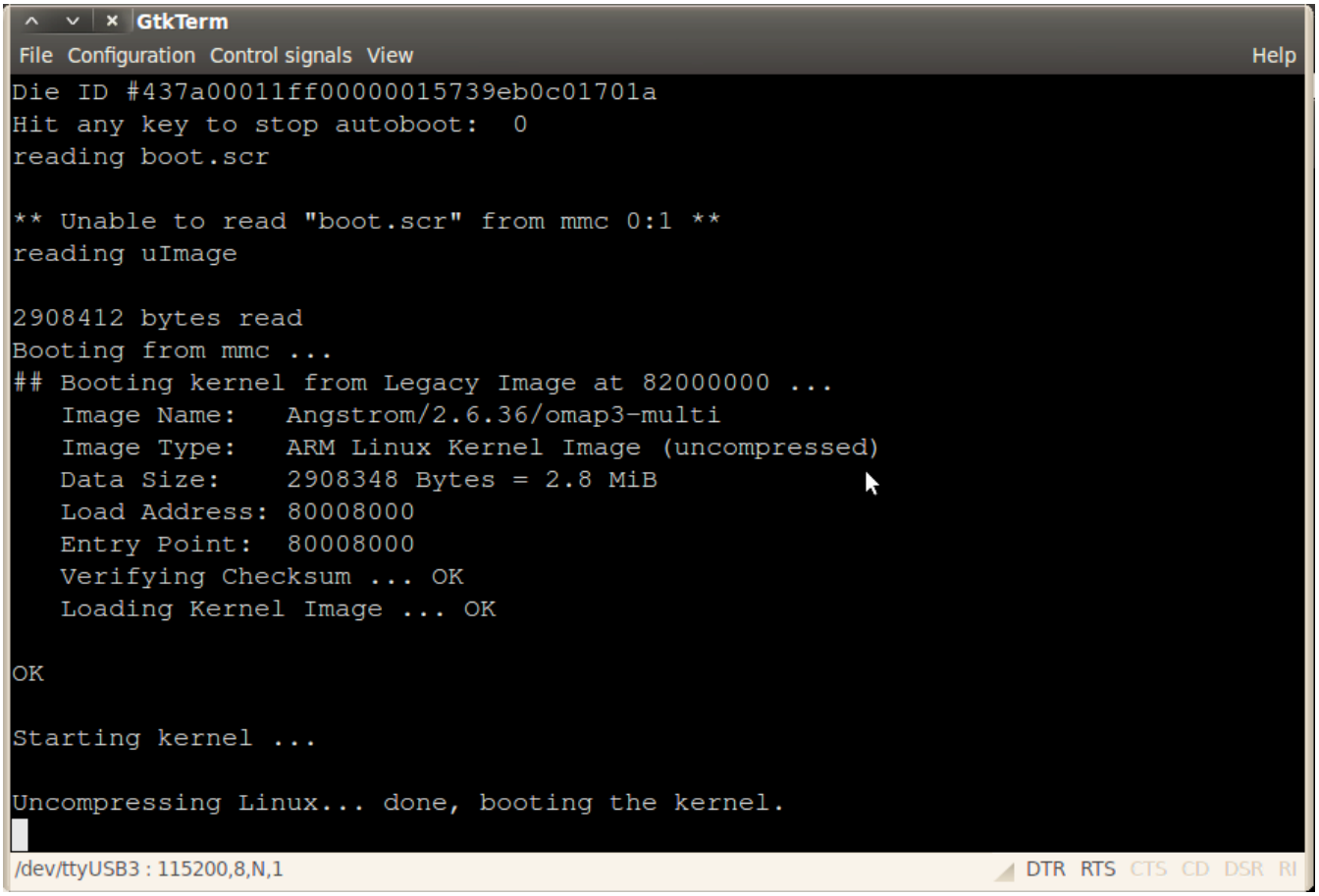
Texas Instruments X-Loader 1.4.2 (Apr 23 2009 - 18:44:10)
Reading boot sector
Loading u-boot.bin from mmc

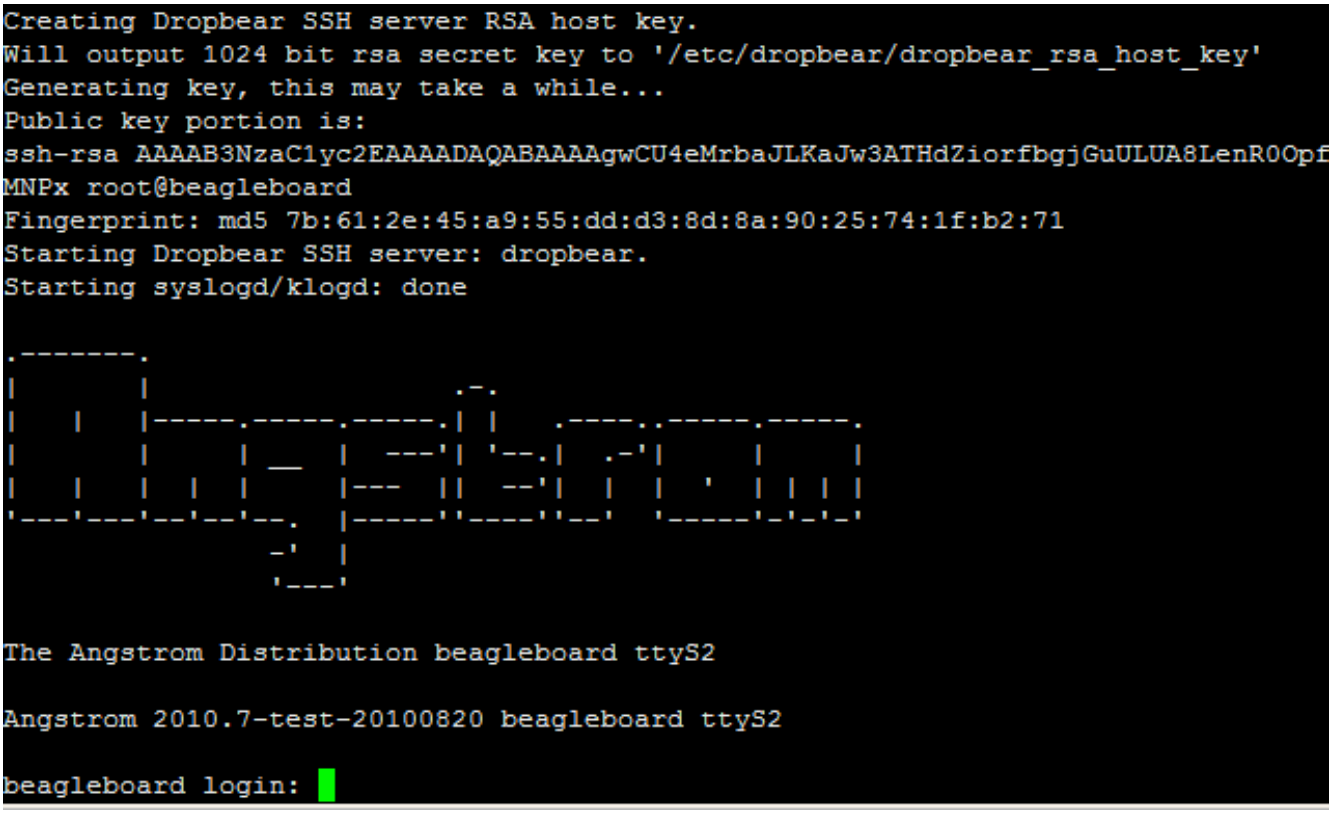

U-Boot 2010.06-rc2 (Jun 14 2010 - 20:56:52)

OMAP3530-GP ES3.1, CPU-OPP2 L3-165MHz
OMAP3 Beagle board + LPDDR/NAND
I2C:   ready
DRAM:  256 MiB
NAND:  256 MiB
In:    serial
Out:   serial
Err:   serial
Board revision Ax/Bx
Die ID #4d5e0004000000004036abc09019012
OMAP3 beagleboard.org #
 CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.4    | VT102 |      Offline
```

หรือใช้โปรแกรม Gtkterm เพื่อเปิด Serial Console ดังแสดงในรูปข้างล่าง




GtkTerm
File Configuration Control signals View Help
Die ID #437a00011ff00000015739eb0c01701a
Hit any key to stop autoboot: 0
reading boot.scr
** Unable to read "boot.scr" from mmc 0:1 **
reading uImage
2908412 bytes read
Booting from mmc ...
Booting kernel from Legacy Image at 82000000 ...
Image Name: Angstrom/2.6.36/omap3-multi
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 2908348 Bytes = 2.8 MiB
Load Address: 80008000
Entry Point: 80008000
Verifying Checksum ... OK
Loading Kernel Image ... OK
OK
Starting kernel ...
Uncompressing Linux... done, booting the kernel.
/dev/ttyUSB3 : 115200,8,N,1
DTR RTS CTS CD DSR RI

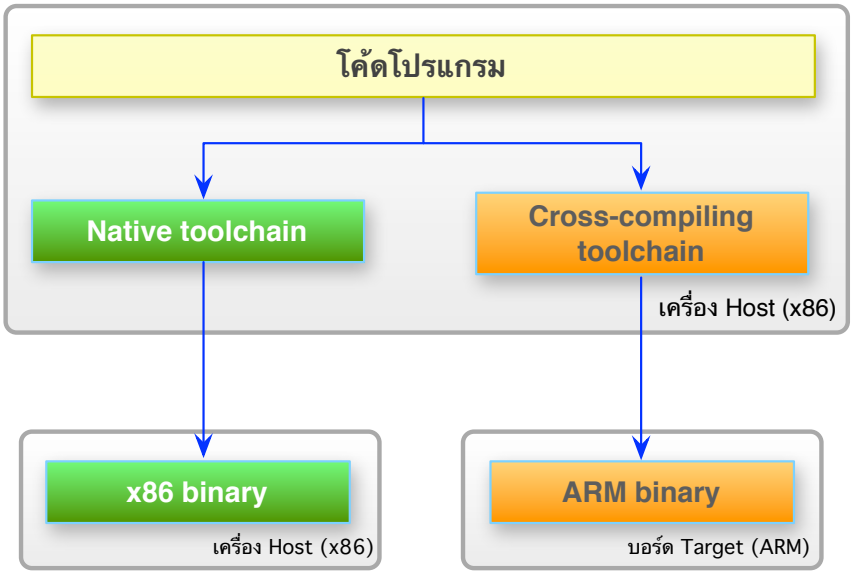

Creating Dropbear SSH server RSA host key.
Will output 1024 bit rsa secret key to '/etc/dropbear/dropbear_rsa_host_key'
Generating key, this may take a while...
Public key portion is:
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAAAgwCU4eMrbaJLKaJw3ATHdZiorfbgjGuULUA8LenR0Opf
MNPx root@beagleboard
Fingerprint: md5 7b:61:2e:45:a9:55:dd:d3:8d:8a:90:25:74:1f:b2:71
Starting Dropbear SSH server: dropbear.
Starting syslogd/klogd: done
The Angstrom Distribution beagleboard ttyS2
Angstrom 2010.7-test-20100820 beagleboard ttyS2
beagleboard login:

เครื่องมือ Cross Toolchains

ระบบปฏิบัติการลีนุกซ์ได้จัดเตรียมเครื่องมือพื้นฐานสำหรับการพัฒนาไว้เบื้องต้นอยู่แล้วซึ่งเรียกว่า “Native Toolchain” ซึ่งตัว native toolchain จะเป็นเครื่องมือหลักในการสร้างรหัสโปรแกรมที่ให้สามารถทำงานได้ บนแพลตฟอร์มของเครื่องคอมพิวเตอร์นั้นๆ (เช่น x86 Platform) แต่ในกรณีการพัฒนาโปรแกรมสำหรับระบบสมองกลฝังตัวนั้น ไม่สามารถใช้ native toolchain ที่มากับระบบปฏิบัติการได้ เนื่องจากระบบสมองกลฝังตัวมีข้อจำกัดในเรื่องของขนาดหน่วยความจำและการเก็บข้อมูล การทำงานช้ากว่าเครื่องคอมพิวเตอร์ส่วนบุคคลและที่สำคัญไม่สามารถจะติดตั้งเครื่องมือพัฒนาทั้งหมดให้อยู่บนระบบสมองกลฝังตัวได้

ดังนั้นต้องใช้เครื่องมือพัฒนาที่สามารถแปลงรหัสเพื่อให้สามารถใช้ข้ามแพลตฟอร์มได้ซึ่งจะถูกเรียกว่า “Cross-compiling Toolchain” หรือ “Cross Toolchains” ดังรูปข้างล่างแสดงให้เห็นว่า cross-compiling toolchain จะคอมไพล์โปรแกรมที่เขียนขึ้นมาให้สามารถทำงานได้บนสถาปัตยกรรมนั้นๆ (เช่น สถาปัตยกรรม ARM) ตามที่ถูกตั้งค่าไว้ก่อนเริ่มทำการคอมไพล์ ดังตัวอย่างข้างล่าง

```
$ export ARCH=ARM
$ export CROSS_COMPILE=arm-linux-gnueabi-
$ make
```



รูปที่ 3.9 แสดงการคอมไพล์โค้ดโปรแกรมสำหรับแพลตฟอร์มทั้ง 2 แบบ

ประเภทของ Cross Toolchains

ในปัจจุบันนี้มีเครื่องมือ cross toochains ที่ช่วยในการพัฒนางานด้านสมองกลฝังตัวอยู่หลายตัว โดยอาจแบ่งออกได้เป็น 2 กลุ่มใหญ่ๆ ได้แก่

1. Toolchains สำเร็จรูป

- CodeSourcery (http://www.codesourcery.com/)

CodeSourcery เป็นตัวที่ออกแบบเพื่อให้สามารถใช้กับ Sourcery G++ และ Eclipse IDE ที่ทำงานร่วมกันกับ GNU Toolchain (gcc, gdb เป็นต้น) ที่รองรับสถาปัตยกรรม ARM, Coldfire, MIPS, SuperH และ PowerPC

- Linaro (http://linaro.org/)

Linaro เป็นองค์กรที่ไม่แสวงหากำไร ซึ่งเป็นการรวมตัวกันของเหล่าวิศวกรมือดี เพื่อร่วมกันสร้างและปรับแต่ง open source และเครื่องมือต่างๆ ให้กับสถาปัตยกรรม ARM ทาง Linaro เองก็ได้จ้างคนที่พัฒนา CodeSourcery เพื่อเน้นให้เป็นเครื่องมือที่เหมาะกับสถาปัตยกรรม ARM ให้มากที่สุด

- DENX ELDK (http://www.denx.de/)

DENX Embedded Linux Development Kit (ELDK) จะเป็นชุดเครื่องมือที่เหมาะกับระบบสมองกลฝังตัวที่ทำงานแบบเวลาจริง (real-time) บนสถาปัตยกรรม ARM, PowerPC และ MIPS ชุดเครื่องมือที่สามารถใช้ได้ฟรีตามเงื่อนไข GPL และ Free Software Licenses ประกอบไปด้วย Cross Development Tools (Compiler, Assembler, Linker เป็นต้น), Native Tools (Shell, commands และ libraries), U-Boot, Linux kernel ที่มีโค้ดโปรแกรม ไลบรารีสำหรับบอร์ดสมองกล, Xenomai (RTOS Emulation framework) และ SELF (Simple Embedded Linux Framework)

- Scratchbox (http://www.scratchbox.org/)

Scratchbox เป็นเครื่องมือที่ใช้ในการพัฒนา Maemo (Nokia 770, N800, N810 Internet Tablets และ Nokia N900) ซึ่งเน้นสนับสนุนสถาปัตยกรรม ARM และ x86

2. Toolchain ที่ต้องสร้างเอง

- Buildroot (http://buildroot.uclibc.org/)

Buildroot เป็นเครื่องมือที่สมบูรณ์ที่นิยมตัวหนึ่งสำหรับการพัฒนาในระบบสมองกลฝังตัว และรองรับสถาปัตยกรรมหลากหลาย นักพัฒนาสามารถสร้าง RFS image (Root File System Image) ที่พร้อมจะสามารถเขียนลงแฟลช (flash) ได้ทันที นอกจากนั้นยังสามารถปรับแต่งหรือเพิ่มเติม open

source อื่นเข้าไปใน RFS Image ได้ โดยตัว buildroot นี้จะสนับสนุนไลบรารีเพียง uClibc แต่ไม่สนับสนุนไลบรารี glibc

- Crosstool-NG (http://crosstool-ng.org/)

Crosstool-NG เป็นเครื่องมือพัฒนาที่ได้รับความนิยมไม่แพ้ตัว buildroot โดยมีรูปแบบในการปรับแต่งเคอร์เนลผ่าน make menuconfig เช่นเดียวกับตัว buildroot สามารถรองรับได้ทั้ง uClibc และ glibc และมาพร้อมกับเครื่องมือในการดีบักโปรแกรม เช่น gdb, strace, dmalloc

- Bitbake

Bitbake คือเครื่องมืออย่างง่ายที่ใช้ในการสร้างชุดแพกเก็ต ซึ่งถูกนำมาใช้ในโครงการ OpenEmbedded เช่น บอร์ด Ångström เป็นต้น

อ้างอิงมาจาก: http://www.wikipedia.com

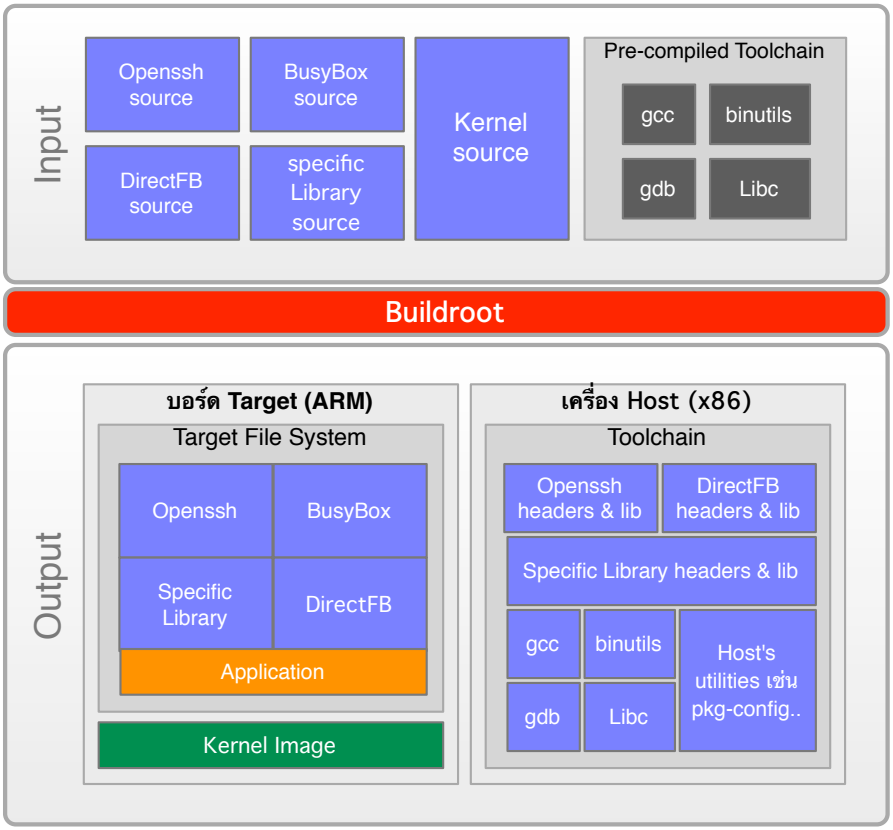
องค์ประกอบหลักภายใน Cross Toolchains

รายละเอียดภายในเครื่องมือ cross toolchains ที่จะต้องนำมาใช้ในการตั้งค่าการคอมไพล์ตัวซอร์สของลีนุกซ์คอร์เนล โปรแกรมที่จำเป็นสำหรับระบบสมองกลฝังตัว (เช่น BusyBox, Openssh, DirectFB เป็นต้น) และไลบรารีที่เกี่ยวข้องต่างๆ ประกอบไปด้วย

- Binutils - เป็นชุดเครื่องมือที่เอาไว้ใช้ในการสร้างและจัดการไบนารีภายในโปรแกรมเพื่อให้เหมาะกับสภาพแวดล้อมบนสถาปัตยกรรมนั้นๆตัวอย่างคำสั่งได้แก่ as (Assembler), ld (Linker), ar, ranlib (สำหรับสร้างไลบรารีแบบ static), objdump, readelf, size, nm, strings, strip เป็นต้น
- GCC Compiler - ชุด GNU C Compiler ซึ่งเป็นชุดที่แจกฟรีสำหรับใช้ในการคอมไพล์ภาษาโปรแกรมต่างๆ เช่น C, C++, Ada, Fortran, Java, Objective-C, Objective-C++ เป็นต้น เพื่อให้สามารถทำงานได้บนสถาปัตยกรรมที่หลากหลายเช่น ARM, AVR, Blackfin, CRIS, FRV, M32, MIPS, MN10300, PowerPC, SH, v850, i386, x86_64, IA64, Xtensa เป็นต้น
- C/C++ Libraries - เป็นชุดไลบรารีที่มีความสำคัญต่อระบบปฏิบัติการลีนุกซ์เป็นอย่างมากซึ่งทำหน้าที่เป็นตัวกลางในการเชื่อมต่อระหว่างโปรแกรมและลีนุกซ์คอร์เนลสำหรับการพัฒนาในระบบสมองกลฝังตัว C Libraries มีอยู่หลายแบบเช่น glibc, uClibc, eglibc, dietlibc, newlib เป็นต้น
- GDB Debugger - GDB เริ่มพัฒนาโดยนายริชาร์ด สตอลแมน เมื่อ พ.ศ. 2529 เพื่อให้เป็นส่วนหนึ่งของระบบ GNU หลังจากที่เขาพัฒนา GNU Emacs จนมีความเสถียรในระดับที่น่าพอใจ ซึ่งแนวความคิดของ GDB นั้นได้มาจาก Dbx ซึ่งเป็นโปรแกรมดีบั๊กเกอร์ที่มากับระบบปฏิบัติการ Unix BSD และในปัจจุบัน GDB ก็ได้ถูกดูแลโดย GDB Steering Committee ซึ่งเป็นคณะกรรมการที่ถูกตั้งโดยมูลนิธิซอฟต์แวร์เสรี (FSF) โดย GDB จะทำงานในแบบการพิมพ์คำสั่ง (command line)

ผ่านหน้าจอ Console แต่ถ้าหากต้องการดีบักผ่านโปรแกรมในลักษณะกราฟฟิก (GUI) ก็สามารถใช้โปรแกรมชื่อว่า DDD แทนได้ ซึ่งโปรแกรมนี้จะทำการเรียกใช้คำสั่งของ GDB อีกต่อหนึ่ง

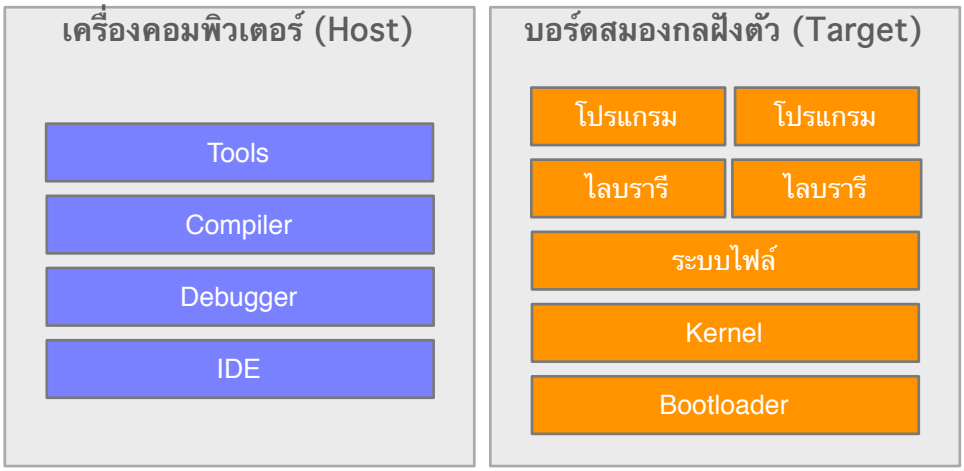
รูปข้างล่างเป็นตัวอย่างการนำชุดเครื่องมือ Buildroot มาใช้ในการสร้างและประกอบลีนุกซ์เคอร์เนล ไลบรารีและโปรแกรมประยุกต์ เพื่อนำไปฝังเข้าไปในบอร์ดสมองกลที่มีสถาปัตยกรรม ARM ต่อไปได้



รูปที่ 3.10 แสดงรายละเอียดโปรแกรมต่างๆ และผลลัพธ์จากการสร้างระบบไฟล์ด้วย Buildroot

ขั้นตอนการเตรียมระบบสำหรับพัฒนาบน Embedded Linux

ในการเตรียมระบบนั้นจะมีองค์ประกอบสำคัญที่ต้องเตรียมไว้สำหรับติดตั้งลงไปในเครื่อง Host และบอร์ด Target ภายใต้ Embedded Linux ซึ่งประกอบไปด้วย



รูปที่ 3.11 องค์ประกอบสำคัญภายในเครื่อง Host และบอร์ด Target

1) การเตรียมและติดตั้งชุดเครื่องมือ Cross-compiling toolchains เพื่อไว้สำหรับติดตั้งไลบรารีที่เกี่ยวข้อง สำหรับการคอมไพล์โค้ดต่างๆที่จะถูกใช้บนบอร์ดสมองกลฝังตัว
2) การเตรียมและติดตั้ง Bootloader เข้าไปในบอร์ดสมองกลฝังตัวเพื่อให้ถูกเรียกใช้งานในครั้งแรกตั้งแต่เริ่มจ่ายไฟเลี้ยงให้บอร์ด โดยตัว bootloader จะโหลดค่าพื้นฐานต่างๆที่ต้องเริ่มต้นสถานะการทำงานให้กับฮาร์ดแวร์ภายในบอร์ดแล้วจึงจะโหลดเคอร์เนลเพื่อเริ่มเข้าสู่ระบบปฏิบัติการลีนุกซ์แบบฝังตัว เป็นลำดับต่อไป
3) ลีนุกซ์เคอร์เนลซึ่งเป็นเคอร์เนลที่ถูกตั้งค่าและปรับแต่งให้เหมาะสมและสามารถทำงานเข้ากันได้กับสถาปัตยกรรมภายในบอร์ดสมองกลฝังตัวโดยได้บรรจุตัวจัดการโปรเซส ตัวจัดการหน่วยความจำ ตัวจัดการการเชื่อมต่อระบบเครือข่าย device drivers และบริการต่างๆสำหรับโปรแกรมประยุกต์ที่ต้องการเรียกใช้งาน
4) Filesystem หรืออาจใช้คำว่า Root file system ซึ่งภายในไฟล์นี้จะบรรจุโปรแกรมระบบ โปรแกรมประยุกต์ที่สำคัญต่างๆและโปรแกรมที่ถูกพัฒนาขึ้นมาเอง นอกจากนั้นก็ยังมีไลบรารีและคำสั่งสคริปท์ที่เตรียมไว้สำหรับการเรียกใช้งานจากโปรแกรมประยุกต์ภายในระบบปฏิบัติการลีนุกซ์แบบฝังตัวบนบอร์ดสมองกล

การเตรียมสภาพแวดล้อมให้กับเครื่อง HOST

ระบบปฏิบัติการลีนุกซ์เช่นตัว Ubuntu ที่อยู่ในเครื่อง Host ในเบื้องต้นถึงแม้ว่า Ubuntu จะมีการจัดเตรียมสภาพแวดล้อมสำหรับการใช้งานแก่ผู้ใช้ในแต่ละระดับอยู่บ้างแล้วก็ตามที แต่สำหรับงานพัฒนาบนระบบสมองกลฝังตัวนั้นก็ยังต้องมีโปรแกรมและไลบรารีสำหรับการพัฒนาภาษาโปรแกรมให้ทำงานได้บนสถาปัตยกรรมอื่นๆ รวมทั้งอาจจะต้องมีเครื่องมือพิเศษเพิ่มเติมที่ต้องติดตั้งเข้าไป อย่างเช่น **autoconf automake libtool libncurses5-dev ncurses-dev bison flex patch curl cvs texinfo build-essential subversion python-dev texi2html nfs-kernel-server tftp minicom gtkcom** เป็นต้น การพัฒนาอยู่บนเครื่อง Host นั้นบางครั้งนักพัฒนาอาจจะไม่จำเป็นจะต้องใส่สิทธิ์ super user (หรือ root) ในการตั้งค่า (Configuration) การคอมไพล์ (Make) แต่ในบางกรณีก็จำเป็นต้องใช้สิทธิ์ในฐานะ root เช่น การสร้าง device node สำหรับใช้ใน root file systems (คำสั่ง **mknode**) การติดตั้งโปรแกรมหรือไลบรารีเพิ่มเติมด้วยชุดคำสั่ง APT การตั้งค่าบริการระบบเครือข่าย (**/etc/init.d/nfs-kernel-server, /etc/init.d/networking**)หรือการติดตั้งไลบรารี/เครื่องมือจากโปรเจ็คภายนอก (**make install**)

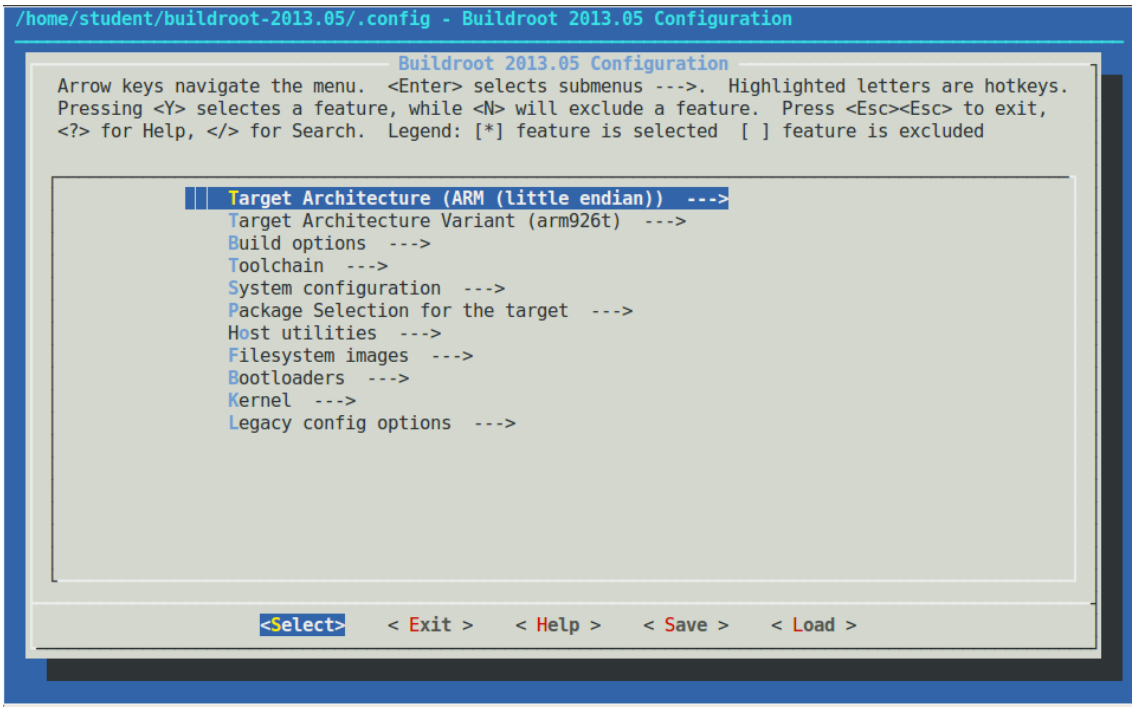
แต่อย่างไรก็ตามสิทธิ์ root นั้นก็เริ่มมีความจำเป็นน้อยลงเรื่อยๆ เนื่องจากเริ่มมีชุดโปรแกรมที่ช่วยอำนวยความสะดวกในการพัฒนาระบบสมองกลฝังตัวแบบกึ่งสำเร็จรูปให้ใช้มากขึ้น ตัวอย่างเช่น Buildroot, CodeSourcery (Sourcery G++ Lite), Linaro, crossTool NG เป็นต้น

ตัวอย่างข้างล่างนี้แสดงขั้นตอนการติดตั้งชุดโปรแกรมเครื่องมือและชุดโปรแกรมสำเร็จรูปพื้นฐานเพิ่มเติมให้กับ Ubuntu

```
$ sudo apt-get install autoconf automake libtool libncurses5-dev
ncurses-dev bison flex patch curl cvs texinfo build-essential subver-
sion python-dev g++ gcc texi2html nfs-kernel-server tftp minicom
gtkterm
```

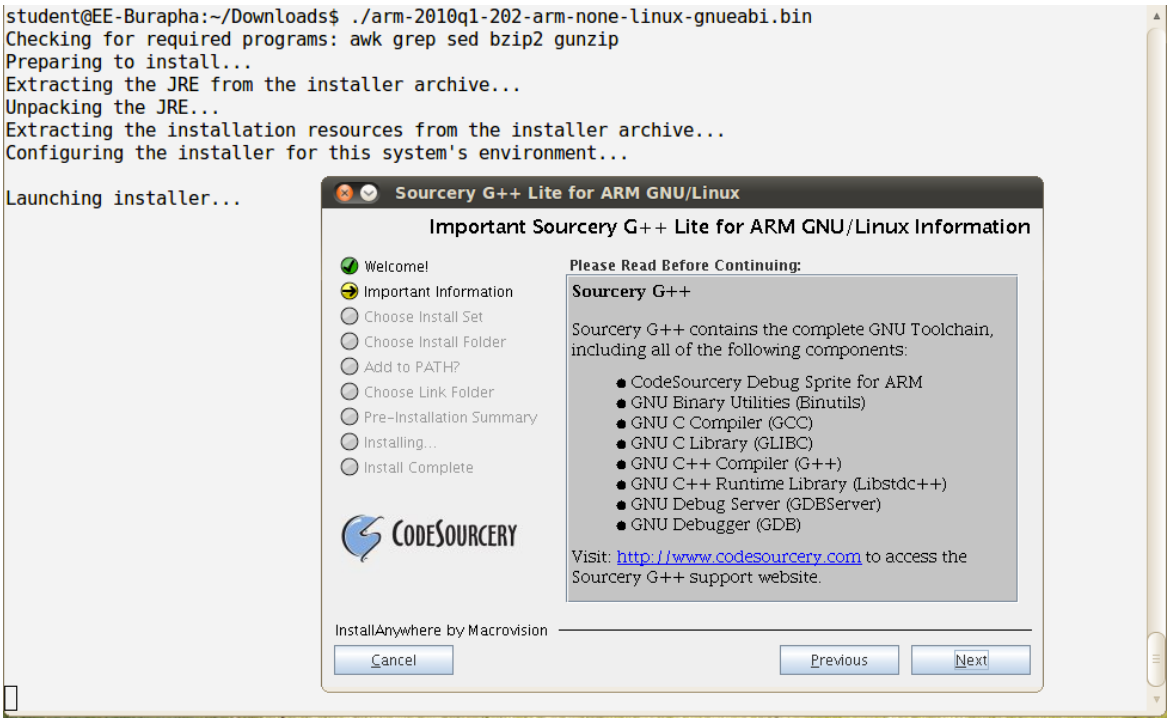
ตัวอย่างการติดตั้งเครื่องมือที่จะช่วยในการพัฒนาระบบสมองกลฝังตัวด้วย Buildroot ดังรูปข้างล่าง

```
$ wget http://buildroot.uclibc.org/downloads/buildroot-2013.05.tar.gz
$ tar -xzvf buildroot-2013.05.tar.gz
$ cd buildroot-2013.05/
$ make menuconfig
```



ตัวอย่างการติดตั้งเครื่องมือชื่อว่า Sourcery G++ Lite ดังรูปข้างล่าง

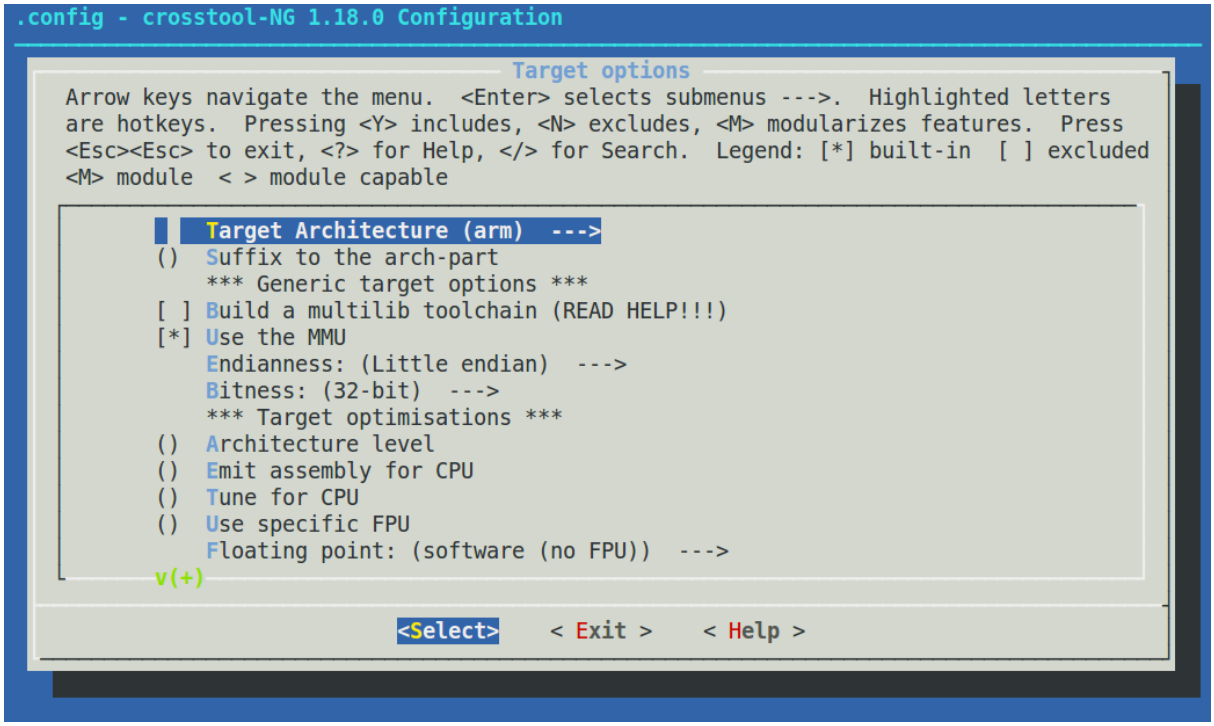
```
$ wget
http://www.codesourcery.com/sgpp/lite/arm/portal/package6490/public/ar
m-none-linux-gnueabi/arm-2010q1-202-arm-none-linux-gnueabi.bin
$ chmod 755 arm-2010q1-202-arm-none-linux-gnueabi.bin
$ ./arm-2010q1-202-arm-none-linux-gnueabi.bin
```



ตัวอย่างการติดตั้งเครื่องมือช่วยในการพัฒนาระบบสมองกลฝังตัวด้วย Croostool-NG ดังรูปข้างล่าง

```
$ mkdir -p ctool-ng
$ cd ctool-ng/
```

```
$ wget http://crosstool-ng.org/download/crosstool-ng/crosstool-ng-1.18.0.tar.bz2
$ tar -xjvf crosstool-ng-1.18.0.tar.bz2
$ cd crosstool-ng-1.18.0/
$ ./configure
$ make
$ sudo make install
$ ct-ng list-samples
$ ct-ng arm-unknown-linux-gnueabi
$ ct-ng menuconfig
```



TOOLCHAIN OPTIONS ที่สำคัญ

ในระหว่างการใช้ชุดโปรแกรม toolchain นั้นนักพัฒนาอาจจะต้องมีการกำหนดค่าสำคัญค่าหนึ่งที่เรียกว่า ABI (Application Binary Interface) ซึ่งเป็นมาตรฐานหนึ่งที่ถูกพัฒนาโดยบริษัท ARM ร่วมกับบริษัทพันธมิตร โดยที่ ABI นั้นจะเป็นการระบุวิธีการเชื่อมต่อและเรียกใช้ในระดับล่างระหว่างโปรแกรมและระบบปฏิบัติการหรือระหว่างโปรแกรมด้วยกันเองตัวอย่างเช่น รูปแบบข้อมูลไฟล์ การผ่านค่าอาร์กิวเม้นต์/การส่งค่ากลับระหว่างฟังก์ชัน การจัดเรียงค่าภายในโครงสร้างแสต็กข้อมูล การเรียกใช้รีจิสเตอร์ เป็นต้น

ดังนั้นจำเป็นอย่างยิ่งที่จะต้องระบุชนิด ABI ที่จะใช้ภายในเคอร์เนลและโปรแกรมที่ทำงานอยู่ภายใต้ระบบปฏิบัติการลีนุกซ์แบบฝังตัวเดียวกัน ซึ่งในปัจจุบันสถาปัตยกรรม ARM จะนิยมใช้ ABI ที่เรียกว่า EABI (Embedded Application Binary Interface) เป็นตัวหลักในการเชื่อมต่อและเรียกใช้ในระดับล่าง

นอกจากนั้นนักพัฒนาควรจะทำความเข้าใจในการตั้งค่า Floating Point เนื่องจากบางสถาปัตยกรรมรองรับการคำนวณแบบ floating point ได้ในระดับฮาร์ดแวร์แต่บางสถาปัตยกรรมอาจจะ

ไม่ได้รองรับ (ไม่มี Floating Point Unit - FPU) แต่จะถูกตั้งค่าให้ใช้การจำลองการคำนวณ floating point ในระดับซอฟท์แวร์แทน เช่น สถาปัตยกรรม ARMv4 และ สถาปัตยกรรม ARMv5 ซึ่งถ้าถูกตั้งค่าให้รองรับการคำนวณแบบ floating point ตัว toolchain เองจะทำการสร้าง hard float code เพื่อเรียกใช้ชุดคำสั่ง floating point ให้โดยตรงด้วยพารามิเตอร์ `-mfloat-abi=hard` ในกรณีที่ไม่มี FPU ก็จะถูกตั้งค่าเป็น `-mfloat-abi=softfp` แทนและสุดท้ายเพื่อไม่ให้เกิดปัญหาในระดับล่างก็จะต้องตั้งค่าให้ถูกต้องกับรุ่นสถาปัตยกรรมนั้นๆ โดยสามารถระบุพารามิเตอร์ตามสถาปัตยกรรมและรุ่น ARM Core เพิ่มเติมโดยการตั้งค่าดังนี้ `-march=armv7 -mcpu=cortex-a8`

Bootloaders

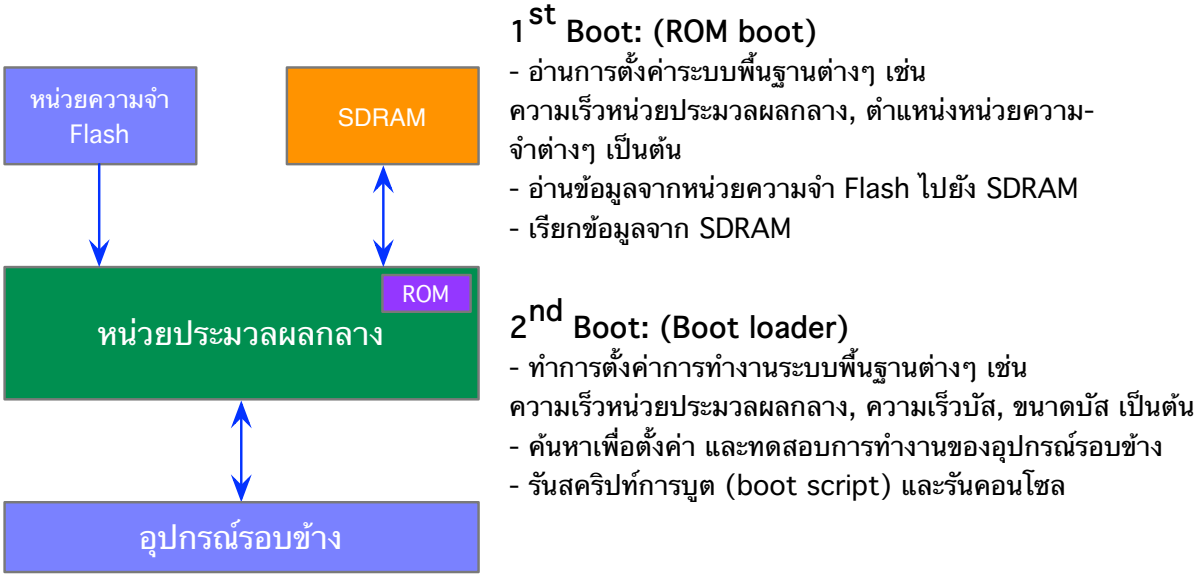
bootloader เป็นส่วนสำคัญอีกส่วนหนึ่งของระบบสมองกลฝังตัวที่มีหน้าที่ในการเตรียมการให้ระบบฮาร์ดแวร์พื้นฐาน (Hardware Initialization) เช่น หน่วยประมวลผลกลาง (CPU) หน่วยความจำ (DRAM) และ MMC Controller ให้พร้อมที่จะทำงานได้ ในขั้นตอนต่อมาก็จะทำการโหลด Kernel image (bzImage, zImage) ที่ถูกบีบอัดและเก็บอยู่ในอุปกรณ์จัดเก็บข้อมูล (เช่น MMC/SD card, NAND Flash, USB Storage หรือผ่านระบบเครือข่ายด้วยโปรโตคอล tftp NFS เป็นต้น) หลังจากนั้นก็จะทำการแตกโปรแกรมไบนารีที่ถูกบรรจุอยู่ภายใน Kernel image ที่ถูกบีบอัดนั้นอยู่ เพื่อนำไปโหลดขึ้นสู่หน่วยความจำภายใน (RAM) เพื่อติดต่อกับอุปกรณ์อื่นๆ เช่น พอร์ต Ethernet หรือ พอร์ต USB เป็นต้น ในระหว่างการทำงานของ Bootloader นั้นสามารถที่จะเข้าสู่หน้าต่างคอนโซล (console) เพื่อเรียกใช้ชุดคำสั่งที่เตรียมไว้ให้ เช่นการดาวน์โหลดข้อมูลผ่านระบบเครือข่ายหรือจากอุปกรณ์จัดเก็บข้อมูล ทางด้านตรวจสอบสถานะของหน่วยความจำ ตรวจสอบสถานะการทำงานของฮาร์ดแวร์ภายใน เป็นต้น

เพื่อให้นักพัฒนาได้เห็นภาพมากขึ้นจึงขอสรุปขั้นตอนการทำงานของ bootloader ตั้งแต่เริ่มบูทเข้าระบบของบอร์ดสมองกลฝังตัวได้ดังนี้ (ดังแสดงในรูปที่ 3.12 และ 3.13)

1. หน่วยประมวลผลกลางจะเริ่มอ่านที่ตำแหน่งของหน่วยความจำ ROM ที่ได้ถูกกำหนดไว้ และดำเนินการทำตามชุดคำสั่งภายในโค้ดที่ถูกเก็บอยู่ภายใน
2. เริ่มต้นเตรียมระบบฮาร์ดแวร์พื้นฐานสำคัญ เช่น หน่วยประมวลผลกลาง ส่วนควบคุมหน่วยความจำ (SDRAM Controller) ส่วนควบคุมตัวบันทึกข้อมูล (MMC Controller) ส่วนควบคุมอินพุท/เอาท์พุท (I/O Controllers) และ ส่วนควบคุมการแสดงผล (Graphics Controllers) เป็นต้น
3. เริ่มต้นเตรียมระบบหน่วยความจำ SDRAM โดยการจัดสรรพื้นที่ภายในหน่วยความจำให้กับฟังก์ชันที่จะใช้ควบคุมฮาร์ดแวร์ของแต่ละส่วน และจัดสรรพื้นที่ให้กับตัวเคอร์เนลที่ถูกเก็บอยู่ภายในอุปกรณ์บันทึกข้อมูล

4.เมื่อเคอร์เนลได้ถูกโหลดขึ้นหน่วยความจำ SDRAM แล้ว จะทำการติดต่อและตรวจสอบสถานะการทำงานของฮาร์ดแวร์ส่วนที่เหลือทั้งหมดเพื่อเตรียมทำการโหลดระบบปฏิบัติการในส่วนที่เหลือที่เรียกว่า root file system ซึ่งบรรจุสคริปท์ ไลบรารี และโปรแกรมต่างๆเอาไว้

5.เมื่อ root file system ถูกโหลดได้สำเร็จ ระบบปฏิบัติการก็จะเรียกโปรแกรมระบบพื้นฐานทั้งหมดที่เกี่ยวกับการจัดการระบบไฟล์ (file system) โปรแกรมจัดการโปรเซส (process) โปรแกรมจัดการลำดับการทำงานของโปรเซส (process scheduling) และเตรียมระบบสำหรับผู้ใช้เพื่อสามารถใช้โปรแกรมประยุกต์ภายในระบบปฏิบัติการสมองกลฝังตัว



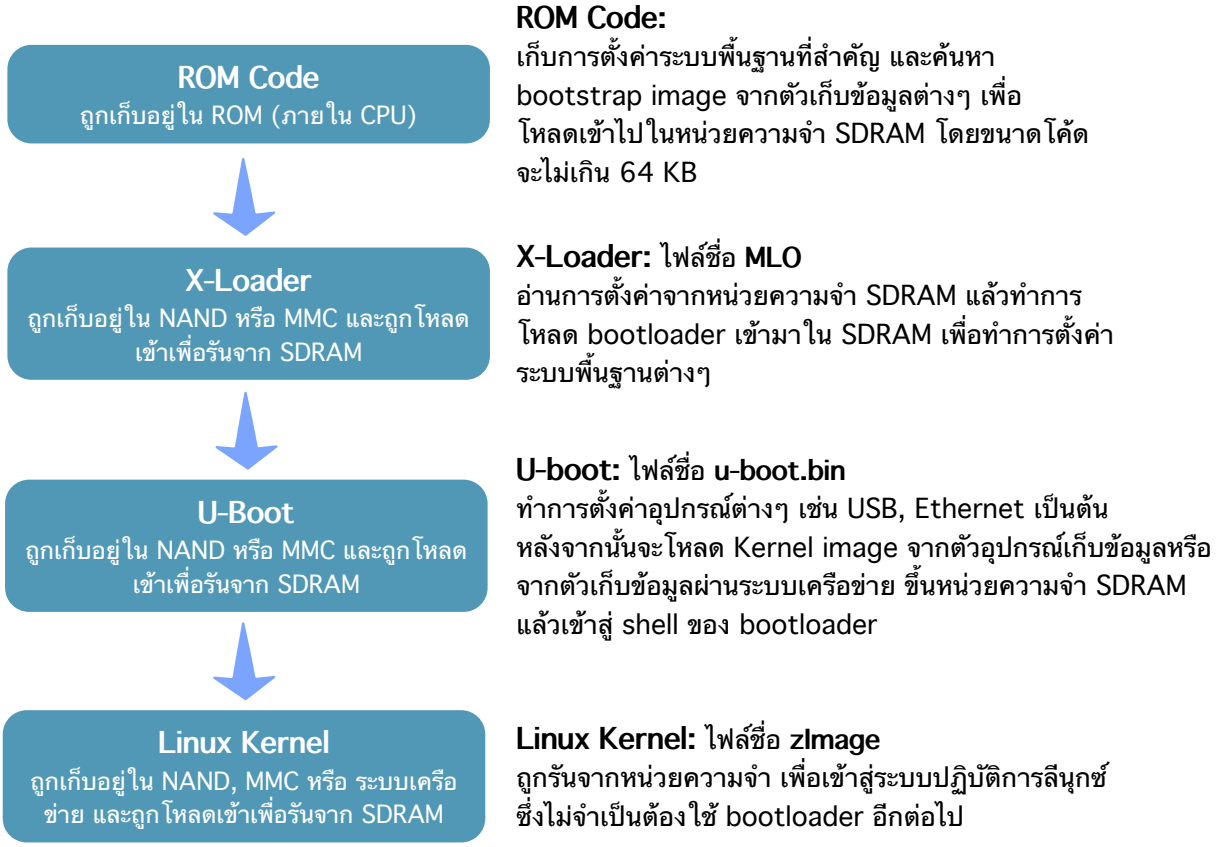
รูปที่ 3.12 ขั้นตอนการบูทระบบ

ขั้นตอนการทำงานเริ่มต้นของลีนุกซ์คอร์เนล (Kernel Initialisation)



รูปที่ 3.13 ขั้นตอนการเริ่มต้นของ Linux Kernel

ตัวอย่างขั้นตอนการทำงานจริงของ bootloader บนบอร์ดสมองกลฝังตัวที่ใช้ OMAP3 SoC ของบริษัท TI ดังแสดงในรูปข้างล่าง



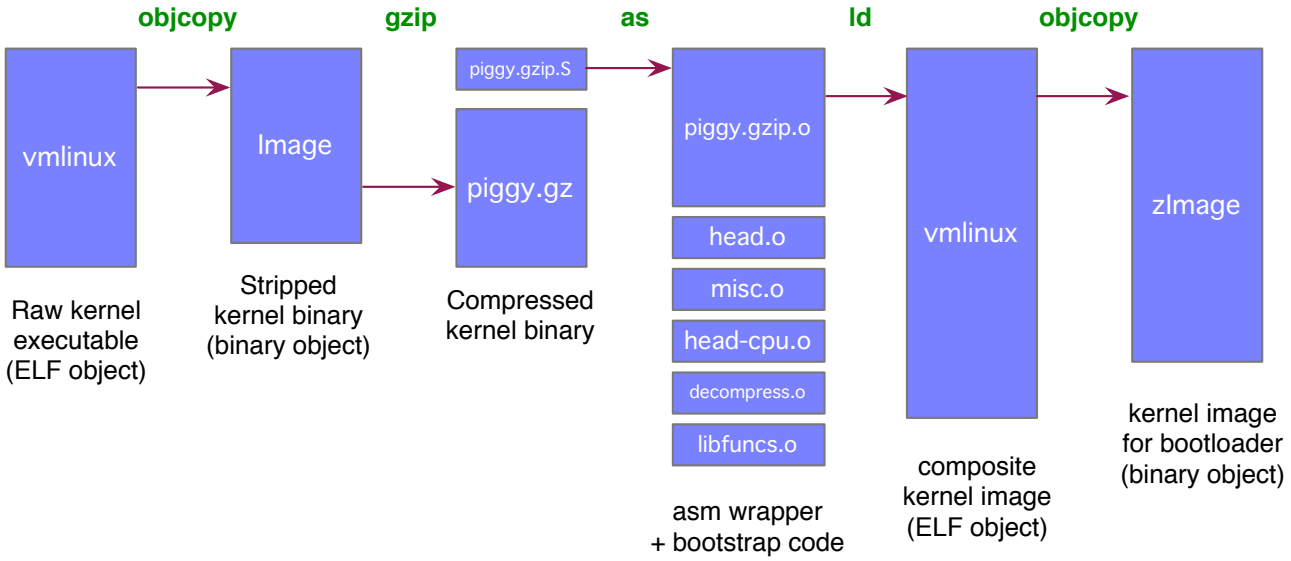
รูปที่ 3.14 แสดงขั้นตอนการทำงานของ bootloader บน OMAP3 SoC

รายละเอียดภายใน KERNEL IMAGE ที่ใช้ในบอร์ดสมองกลฝังตัว

ไฟล์สำคัญต่างๆที่ถูกสร้างหลังจากการคอมไพล์ลีนุกซ์เคอร์เนลได้แก่ Kernel image (เช่น bzImage, zImage) และ root file system (RFS) ซึ่งไฟล์ Kernel image จะถูกนำมาบรรจุลงอยู่ในบอร์ดสมองกลเพื่อทำหน้าที่ตามขั้นตอนที่ได้อธิบายไปก่อนหน้านี้ แต่ก่อนที่จะกลายมาเป็นไฟล์ Kernel image ได้นั้นจะถูกประกอบมาจากไฟล์ไบนารีต่างๆ ที่ถูกเก็บอยู่ในไดเรกทอรี `<linux version>/arch/arm/boot/compressed/` ดังตัวอย่างผลจากการคอมไพล์ด้วยเครื่องมือ buildroot สำหรับสถาปัตยกรรม ARM ที่ใช้ SoC รุ่น Versatile

```
$ ls ~/buildroot/output/build/linux-3.9.4/arch/arm/boot/compressed
ashldi3.o       head-shark.S     misc.c          piggy.xzkern.S
ashldi3.S       head-sharpsl.S   misc.o          sdhi-sh7372.c
atags_to_fdt.c  head-shmobile.S  mmcif-sh7372.c  sdhi-shmobile.c
big-endian.S    head-xscale.S    ofw-shark.c     sdhi-shmobile.h
decompress.c    lib1funcs.o      piggy.gzip      string.c
decompress.o    lib1funcs.S      piggy.gzip.o    string.o
head.o          libfdt_env.h     piggy.gzip.S    vmlinux
head.S          ll_char_wr.S     piggy.lzma.S    vmlinux.lds
head-sa1100.S   Makefile         piggy.lzo.S     vmlinux.lds.in
```

จากรูปข้างล่างแสดงรายละเอียดของการประกอบไฟล์ไบนารีที่มีหน้าที่แตกต่างกันไป โดยนำคำสั่งของชุดเครื่องมือ Binutils มาใช้ประกอบในแต่ละขั้นตอนจนได้มาเป็นไฟล์ Kernel image ที่ถูกบีบอัดในชื่อว่า zImage ในที่สุด

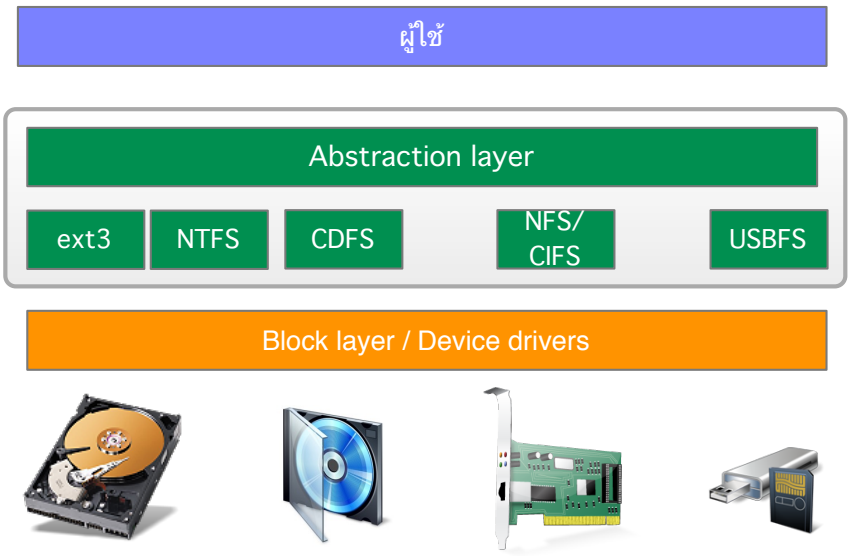


รูปที่ 3.15 ขั้นตอนการสร้างไฟล์ Kernel Image

Linux File Systems

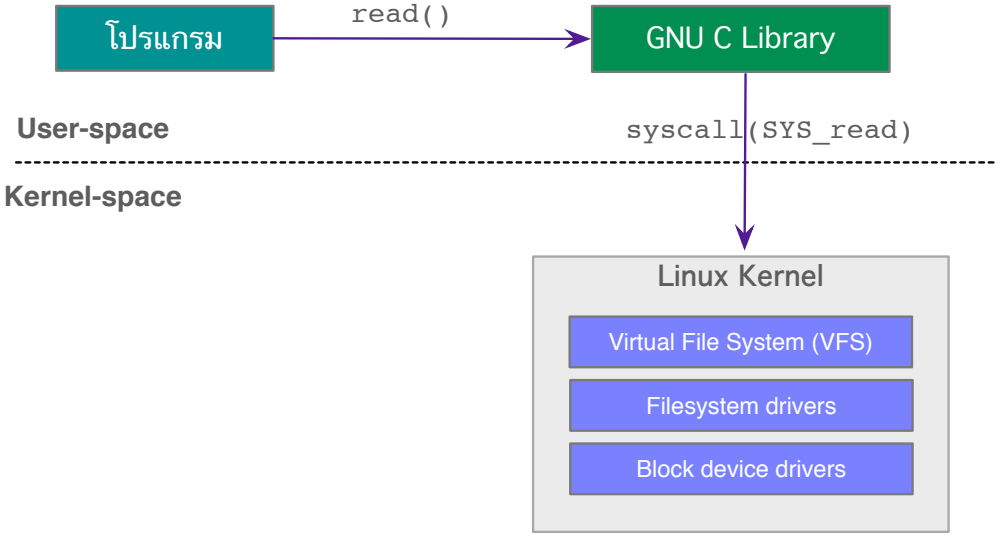
VIRTUAL FILESYSTEMS

ระบบไฟล์เสมือนหรือเรียกอีกชื่อว่า Virtual Filesystem Switch (VFS) ได้เตรียมชุดการเชื่อมต่อหรือฟังก์ชันสำหรับการเข้าถึงทรัพยากรระบบ (System Call Interface) ระหว่างโปรแกรมในส่วนผู้ใช้ (user-space) และระบบไฟล์ที่แตกต่างกันของอุปกรณ์เก็บข้อมูล เพื่อไม่ต้องให้นักพัฒนาต้องพยายามทำความเข้าใจในรายละเอียดทางด้านเทคนิคเชิงลึกภายในระบบไฟล์ของอุปกรณ์เก็บข้อมูลชนิดนั้นๆ (ระบบไฟล์ Ext3 ในฮาร์ดิสก์ หรือ ระบบไฟล์ ISO 9660 ในซีดีรอม เป็นต้น) ซึ่งสถาปัตยกรรมของระบบไฟล์ภายในระบบปฏิบัติการลีนุกซ์นั้นถือได้ว่าเป็นตัวอย่างที่น่าสนใจอย่างยิ่งที่สามารถจัดการความซับซ้อนในการเข้าถึงระบบไฟล์ที่มีหลายหลากในปัจจุบันให้กลายเป็นเรื่องง่ายในการเรียกใช้งาน (abstraction layer) ซึ่งจะจัดเตรียม uniform interface สำหรับระบบไฟล์ของอุปกรณ์เก็บข้อมูลชนิดต่างๆ เช่น ext3 NTFS (ฮาร์ดดิสก์) CDFS (ซีดีรอม) NFS/CIFS (Network Interface Card) และ USBFS (USB Storage) เป็นต้น ดังแสดงในรูปข้างล่าง



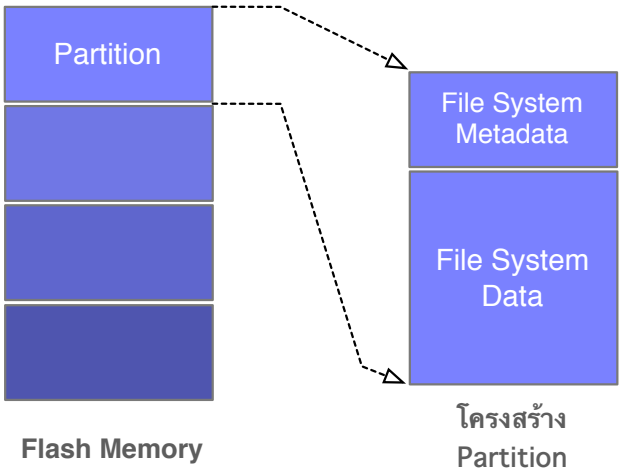
รูปที่ 3.16 แสดงโครงสร้างของชั้น Virtual Filesystem

ฟังก์ชันพื้นฐาน (system call functions) ที่ระบบปฏิบัติการลีนุกซ์ได้เตรียมไว้ให้ตัวอย่างเช่น ฟังก์ชัน `read()` ที่สามารถอ่านข้อมูลออกมาเป็นไบต์จากไฟล์ที่ถูกเก็บอยู่ในอุปกรณ์เก็บข้อมูลโดยไม่ต้องสนใจว่าอุปกรณ์เก็บข้อมูลนั้นใช้ระบบไฟล์ชนิดใด แต่จะเป็นหน้าที่ของลีนุกซ์คอร์เนลซึ่งอยู่ในโปรแกรมระดับ kernel-space เองที่จะเข้าถึงระบบไฟล์ว่าจะต้องใช้เทคนิคในการอ่าน/เขียนอย่างไรตามรูปแบบมาตราฐานโปรโตคอลของเทคโนโลยีนั้นๆ ดังแสดงในรูปข้างล่าง



รูปที่ 3.17 ตัวอย่างการเรียกใช้ฟังก์ชัน (system call) ของ Virtual Filesystem

พาร์ทิชั่น (partition) คือส่วนของพื้นที่ ที่อาจจะถูกแบ่งเป็นตัวบันทึกข้อมูลเสมือน (logical disk) อยู่บนพื้นที่ของอุปกรณ์จัดเก็บข้อมูล (physical disk) เช่น ฮาร์ดดิสหรืออุปกรณ์ชนิด flash memory ซึ่งพื้นที่ของพาร์ทิชั่นจะเป็นส่วนเก็บข้อมูลของระบบไฟล์ที่เรียกกันว่า file system ดังแสดงในรูปข้างล่าง



รูปที่ 3.18 โครงสร้างภายในพาร์ทิชันของหน่วยความจำแฟลช

หากต้องการทราบรายละเอียดต่างๆภายในเครื่องว่ามีการแบ่งพื้นที่เป็นกี่พาร์ทิชั่นและเป็นชนิดใด สามารถใช้ คำสั่ง fdisk ซึ่งคำสั่งนี้สามารถตรวจสอบชนิดของพาร์ทิชั่นที่มีอยู่ในปัจจุบันได้ไม่น้อยกว่า 90 ชนิด ดังตัวอย่างการใช้งานข้างล่าง

```
$ sudo fdisk -l
Disk /dev/sda: 107.4 GB, 107374182400 bytes
255 heads, 63 sectors/track, 13054 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000c74

 Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1       13055   104855552   83  Linux

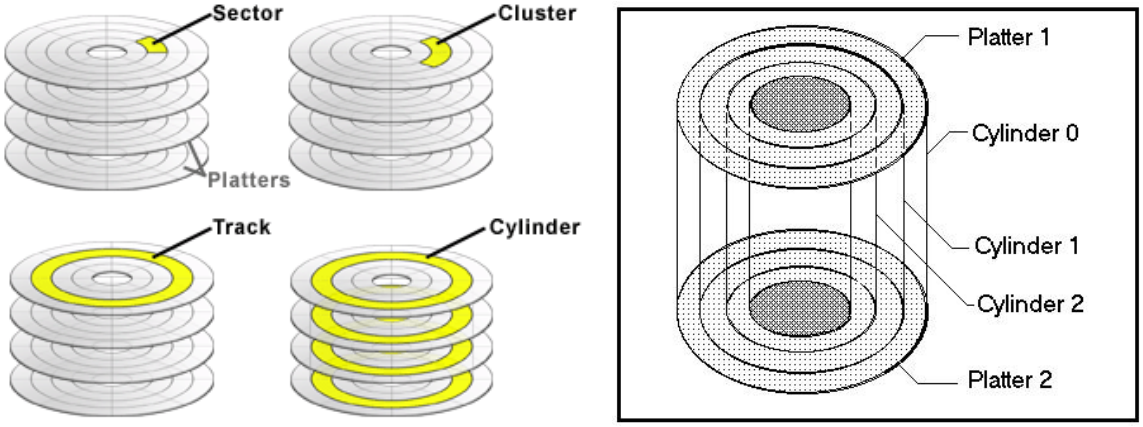
Disk /dev/sdb: 17.2 GB, 17179869184 bytes
255 heads, 63 sectors/track, 2088 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x0001e836

 Device Boot      Start         End      Blocks   Id  System
/dev/sdb1               1        2089    16775168   82  Linux swap / Solaris
```

จากผลลัพธ์ข้างต้น ค่าจำนวน heads ที่ 255 บ่งถึงจำนวนของจาน platters หรือ disks ซึ่งจะถูกตั้งชื่อของแต่ละ disk เป็น D1, D2, ... D255 โดยแต่ละแผ่นจะมี track ในลักษณะวงกลมและจะเริ่มนับจากวงนอกเข้าสู่วงในสุดซึ่งมีจำนวนทั้งสิ้น 13,054 tracks/disk โดยจะตั้งชื่อแต่ละ track ว่าเป็น T1, T2, ...T13054

ซึ่งแต่ละ track ของแต่ละชั้นของ disk จะถูกแบ่งออกเป็นส่วนๆเรียกว่า cylinder ให้อยู่ในตำแหน่งเดียวกัน ยกตัวอย่างเช่น tracks ที่ T2 ของแผ่น ตั้งแต่ D1, D2, ถึง D255 จะรวมกันเป็น

cylinder C2 จากผลลัพพธ์คำสั่ง fdisk ข้างต้น แต่ละ track จะมี logical sectors เหมือนกันคือมีค่าเท่ากับ 63 sectors ซึ่งจะถูกตั้งชื่อเป็น S1, S2, ... S63 ตามลำดับโดยแต่ละ sector จะมีขนาดความจุเท่ากับ 512 bytes



รูปที่ 3.19 รายละเอียดของการเก็บข้อมูลภายในจานแม่เหล็ก

ดังนั้นเมื่อคำนวณพื้นที่ของฮาร์ดดิสที่สามารถใช้ได้ด้วยสูตรข้างล่างนี้ จะมีพื้นที่เท่ากับ

ขนาดพื้นที่ฮาร์ดดิสที่สามารถใช้งานได้ (Byte) = (จำนวน heads หรือ disks) * (จำนวน tracks ต่อ disk) * (จำนวน sectors ต่อ track) * (จำนวน bytes ต่อ sector หรือ sector size)

ตัวอย่างการคำนวณจากค่าที่ได้จากคำสั่ง fdisk ข้างต้น ดังนี้

ขนาดพื้นที่ฮาร์ดดิสที่สามารถใช้งานได้ = 255 * 13054 * 63 * 512 = **107,372,805,120** bytes

หมายเหตุ:

ค่าที่คำนวณจากสูตรข้างต้นจะแตกต่างเล็กน้อยจากขนาดพื้นที่ความจุของฮาร์ดดิสก์จริง (107,374,182,400 bytes) เนื่องจากสูตรไม่ได้สนใจจำนวนไบต์ในส่วนท้ายสุดของ cylinder หรือความไม่สมบูรณ์ของส่วน cylinder

ตัวอย่างโปรแกรม partition_info.c ที่ใช้ในการอ่านรายละเอียดของพาร์ทิชั่น

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
```

```
#define SECTOR_SIZE 512
#define MBR_SIZE SECTOR_SIZE
#define MBR_DISK_SIGNATURE_OFFSET 440
#define MBR_DISK_SIGNATURE_SIZE 4
#define PARTITION_TABLE_OFFSET 446
#define PARTITION_ENTRY_SIZE 16 // sizeof(PartEntry)
#define PARTITION_TABLE_SIZE 64 // sizeof(PartTable)
#define MBR_SIGNATURE_OFFSET 510
#define MBR_SIGNATURE_SIZE 2
#define MBR_SIGNATURE 0xAA55
#define BR_SIZE SECTOR_SIZE
#define BR_SIGNATURE_OFFSET 510
#define BR_SIGNATURE_SIZE 2
#define BR_SIGNATURE 0xAA55

typedef struct {
    unsigned char boot_type; // 0x00 - Inactive; 0x80 - Active (Bootable)
    unsigned char start_head;
    unsigned char start_sec:6;
    unsigned char start_cyl_hi:2;
    unsigned char start_cyl;
    unsigned char part_type;
    unsigned char end_head;
    unsigned char end_sec:6;
    unsigned char end_cyl_hi:2;
    unsigned char end_cyl;
    unsigned long abs_start_sec;
    unsigned long sec_in_part;
} PartEntry;

typedef struct {
    unsigned char boot_code[MBR_DISK_SIGNATURE_OFFSET];
    unsigned long disk_signature;
    unsigned short pad;
    unsigned char pt[PARTITION_TABLE_SIZE];
    unsigned short signature;
} MBR;

void print_computed(unsigned long sector) {
    unsigned long heads, cyls, tracks, sectors;

    sectors = sector % 63 + 1 /* As indexed from 1 */;
    tracks = sector / 63;
    cyls = tracks / 255 + 1 /* As indexed from 1 */;
    heads = tracks % 255;
    printf("(%3d/%5d/%1d)", heads, cyls, sectors);
}

int main(int argc, char *argv[]) {
    char *dev_file = "/dev/sda";
    int fd, i, rd_val;
    MBR m;
```

```
    PartEntry *p = (PartEntry *)(m.pt);
    if (argc == 2) {
        dev_file = argv[1];
    }
    if ((fd = open(dev_file, O_RDONLY)) == -1) {
        fprintf(stderr, "Failed opening %s: ", dev_file);
        perror("");
        return 1;
    }
    if ((rd_val = read(fd, &m, sizeof(m))) != sizeof(m)) {
        fprintf(stderr, "Failed reading %s: ", dev_file);
        perror("");
        close(fd);
        return 2;
    }
    close(fd);
    printf("\nDOS type Partition Table of %s:\n", dev_file);
    printf("  B Start (H/C/S)   End (H/C/S) Type  StartSec    TotSec\n");
    for (i = 0; i < 4; i++) {
        printf("%d:%d (%3d/%4d/%2d) (%3d/%4d/%2d)  %02X %10d %9d\n",
            i + 1, !!(p[i].boot_type & 0x80),
            p[i].start_head,
            1 + ((p[i].start_cyl_hi << 8) | p[i].start_cyl),
            p[i].start_sec,
            p[i].end_head,
            1 + ((p[i].end_cyl_hi << 8) | p[i].end_cyl),
            p[i].end_sec,
            p[i].part_type,
            p[i].abs_start_sec, p[i].sec_in_part);
    }
    printf("\nRe-computed Partition Table of %s:\n", dev_file);
    printf("  B Start (H/C/S)   End (H/C/S) Type  StartSec    TotSec\n");
    for (i = 0; i < 4; i++) {
        printf("%d:%d ", i + 1, !!(p[i].boot_type & 0x80));
        print_computed(p[i].abs_start_sec);
        printf(" ");
        print_computed(p[i].abs_start_sec + p[i].sec_in_part - 1);
        printf(" %02X %10d %9d\n", p[i].part_type,
            p[i].abs_start_sec, p[i].sec_in_part);
    }
    printf("\n");
    return 0;
}
```

นอกจากนั้นคำสั่ง fdisk ที่ใช้ดูรายละเอียดของพาร์ทิชั่นแล้ว ยังสามารถใช้คำสั่งอื่นๆเช่น คำสั่ง parted ที่สามารถแบ่งหรือปรับขนาดของพาร์ทิชั่นและสามารถตรวจสอบระบบไฟล์นั้นว่าเป็นชนิดใด ดังตัวอย่างการใช้งานข้างล่าง

```
$ parted -l
Model: VMware, VMware Virtual S (scsi)
```

```
Disk /dev/sda: 107GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End    Size   Type     File system  Flags
 1      1049kB  107GB  107GB  primary  ext4         boot
Model: VMware, VMware Virtual S (scsi)
Disk /dev/sdb: 17.2GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End     Size    Type     File system     Flags
 1      1049kB  17.2GB  17.2GB  primary  linux-swap(v1)

$ cat /etc/fstab
# /etc/fstab: static file system information.
#
# Use 'blkid -o value -s UUID' to print the universally unique identifier
# for a device; this may be used with UUID= as a more robust way to name
# devices that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point>   <type>  <options>              <dump>  <pass>
proc            /proc           proc    nodev,noexec,nosuid      0       0
/dev/sda1       /               ext4    errors=remount-ro        0       1
/dev/sdb1       none            swap    sw                       0       0
/dev/fd0        /media/floppy0  auto    rw,user,noauto,exec,utf8 0       0
```

สำหรับระบบสมองกลฝังตัวนั้นไม่เพียงแต่เคอร์เนลจะเป็นส่วนสำคัญส่วนหนึ่งของระบบปฏิบัติการลีนุกซ์แล้ว ยังมีอีกส่วนหนึ่งที่เป็นส่วนสำคัญไม่น้อยกว่ากันคือ Root file system (RFS) ซึ่งบรรจุชุดโปรแกรมพื้นฐานไลบรารีและโปรแกรมประยุกต์ต่างๆที่จะทำให้บอร์ดสมองกลฝังตัวสามารถทำงานได้อย่างมีระบบและตามเงื่อนไขการทำงานที่ได้ถูกกำหนดไว้ โดย RFS นั้นจะถูกบันทึกเก็บไว้ในอุปกรณ์จัดเก็บข้อมูล ซึ่งเมื่อบอร์ดสมองกลถูกจ่ายไฟและเริ่มบูทตัวเองขึ้นตัว bootloader ที่อยู่ภายใน ROM ก็จะเริ่มทำงานและโหลดไฟล์ Kernel image ขึ้นสู่หน่วยความจำ (RAM) หลังจากนั้น Kernel จะทำการโหลด RFS ที่ถูกเก็บอยู่บนพาร์ทิชั่นของตัวอุปกรณ์บันทึกข้อมูลหรือภายในพาร์ทิชั่นที่อยู่บนเครื่องคอมพิวเตอร์ภายในระบบเครือข่าย เพื่อเริ่มขั้นตอนเข้าสู่ระบบปฏิบัติการลีนุกซ์อย่างเต็มรูปแบบ ซึ่งตัวอุปกรณ์บันทึกข้อมูลจะมีรูปแบบระบบไฟล์อยู่หลักๆ 2 แบบคือ แบบ Block และ แบบ Flash

แบบ Block จะสามารถอ่านและเขียนเพิ่มเติมลงไปได้โดยไม่จำเป็นต้องทำการลบก่อนที่จะทำการเขียนซ้ำ ตัวอย่างของอุปกรณ์ที่เป็นแบบ Block ได้แก่ ฮาร์ดดิส แผ่นดิสก์แบบอ่อน (floppy disk) และดิสก์หน่วยความจำ (RAM disk) เป็นต้น แต่อย่างไรก็ตามก็ยังมีอุปกรณ์บันทึกข้อมูลแบบ Flash ที่ใช้วิธีการบันทึกข้อมูลแบบ Block ตัวอย่างเช่น Compact Flash, Smart Media, Memory Stick, Multimedia Card , Flash Drive, MMC/SD card แต่สำหรับอุปกรณ์บันทึกข้อมูลแบบ Flash เช่น NOR flash หรือ NAND flash จะต้องทำการลบข้อมูลเดิมก่อนที่จะเขียนข้อมูลตัวใหม่เพิ่มเข้าไป โดยตัว NOR flash นั้นจะทนต่อการลบและเขียนข้อมูลได้ประมาณ 10,000 ครั้ง ซึ่งความเร็วในการอ่านข้อมูล

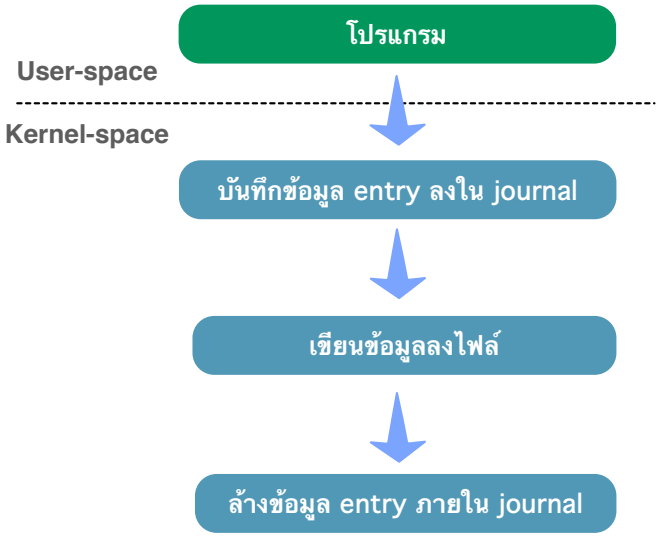
จะเร็ว แต่ความเร็วในการเขียนและลบข้อมูลจะค่อนข้างช้า ดังนั้นจึงเหมาะที่จะนำมาใช้แทนหน่วยความจำ ROM ที่ไม่ต้องเขียนหรือลบข้อมูลบ่อยๆ เช่น BIOS (Basic Input/Output System) ของเครื่องคอมพิวเตอร์, เก็บเฟิร์มแวร์ (firmware) บนบอร์ดไมโครคอลโทรเลอร์ และเก็บ bootloader ภายในระบบสมองกลฝังตัว เป็นต้น

สำหรับอุปกรณ์ NAND flash นั้นได้ถูกสร้างโดยบริษัทโตชิบา ในปี ค.ศ. 1989 ซึ่งสามารถลบและเขียนข้อมูลได้รวดเร็วกว่า ชิพมีขนาดเล็กกว่า ความจุสูงกว่าและมีราคาต่ำกว่าตัว NOR flash นอกจากนั้นก็ยังมีความทนทานกว่าถึงสิบเท่า แต่ก็ยังทำงานได้ช้ากว่าในกรณีการเข้าถึงข้อมูลแต่ละไบต์ในรูปแบบไม่เรียงตามลำดับ (random access) และอาจมีโอกาสเกิดความผิดพลาดในระดับบิตสูงกว่าอุปกรณ์ NOR Flash อีกด้วย

การป้องกันข้อมูลภายในระบบไฟล์

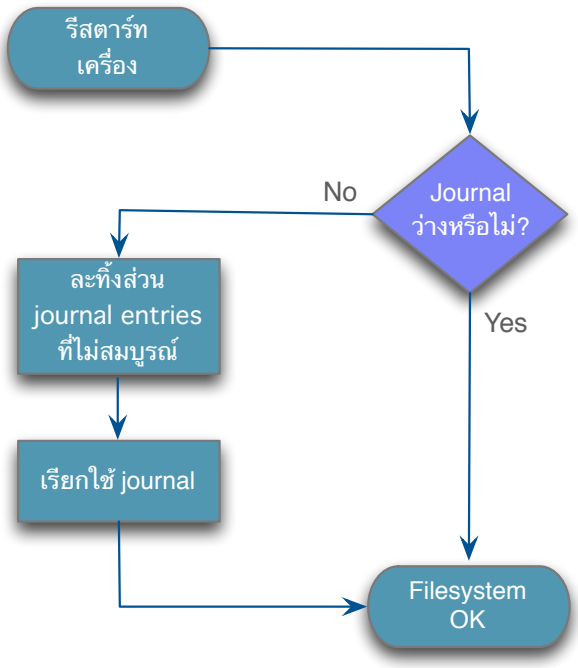
ระบบไฟล์แต่ละชนิดจะมีวิธีการป้องกันการสูญหายของข้อมูลที่แตกต่างกัน ซึ่งระบบไฟล์สมัยก่อน (Traditional BLock Filesystems) ในกรณีที่เกิดการผิดพลาดของระบบหรือมีการปิดตัวระบบแบบกระทันหัน ระบบจะไม่สามารถเก็บสถานะการทำงาน (State) ขณะก่อนปิดระบบได้ จำเป็นจะต้องมีการตรวจสอบและซ่อมแซมทั้งหมดอีกครั้งเมื่อเปิดเครื่องเพื่อเข้าสู่ระบบใหม่ ตัวอย่างเช่น ext2 (Linux filesystem), vfat (Windows filesystem) ต่างๆ จะต้องมีคำสั่งที่สามารถทำการซ่อมแซมในกรณีที่ระบบไฟล์เกิดความผิดพลาดได้แก่คำสั่ง fsck.ext2 หรือ fsck.vat เป็นต้น

แต่ในปัจจุบันระบบไฟล์ก็ได้มีการพัฒนาที่ฉลาดขึ้น โดยมีพื้นฐานของระบบไฟล์เป็นชนิด Journaled Filesystems ซึ่งได้ถูกออกแบบมาให้สามารถเก็บสถานะการทำงานที่อยู่ในตำแหน่งที่ถูกต้องก่อนที่จะเกิดเหตุการณ์ผิดพลาดหรือมีการปิดตัวระบบแบบกระทันหัน โดยใช้วิธีการบันทึกสถานะทั้งหมดลงใน Journal ดังแสดงขั้นตอนในรูปข้างล่าง



รูปที่ 3.20 ขั้นตอนการบันทึกข้อมูลลง journal

เมื่อระบบกลับมาทำงานอีกครั้งระบบไฟล์แบบ Journal นี้จะทำการตรวจสอบ journal ที่อยู่ในไฟล์ก่อนที่ระ บบปิดทำงานก่อนหน้านี้ว่ามีหรือไม่ ถ้ามีก็จะทำการกู้ระบบไฟล์ที่เสียหายเหล่านั้นกลับคืนมาด้วยขั้นตอนดัง แสดงในรูปข้างล่าง



รูปที่ 3.21 ขั้นตอนการอ่านข้อมูลจาก journal ในขณะบูทเครื่อง

ซึ่งระบบปฏิบัติการลีนุกซ์ในปัจจุบันได้เปลี่ยนมาใช้ระบบไฟล์แบบ ext3 และ ext4 ซึ่งเป็นแบบ journal ทั้งหมดแล้ว ซึ่งพัฒนาต่อยอดมากจาก ext2 โดยเพิ่มความสามารถของระบบไฟล์แบบ journal และปรับปรุงให้รองรับขนาดความจุข้อมูลที่สูงขึ้นของอุปกรณ์จัดเก็บข้อมูล เมื่อผู้ใช้ต้องการตรวจสอบและซ่อมแซมระบบไฟล์ด้วยตัวเองก็สามารถใช้คำสั่ง e2fsck ดังตัวอย่างข้างล่าง

```
$ sudo e2fsck /dev/sda1
e2fsck 1.41.11 (14-Mar-2010)
/dev/sda1 is mounted.

WARNING!!!  The filesystem is mounted. If you continue you ***WILL*** cause
***SEVERE*** filesystem damage.

Do you really want to continue (y/n)?
```

หลังจากเปิดตัวระบบไฟล์ ext3 และ ext4 ของระบบปฏิบัติการลีนุกซ์ตั้งแต่ปี ค.ศ. 2008 นาย Theodore Ts'o ซึ่งเป็นนักพัฒนาหลักของระบบไฟล์ ext2 ext3 และ ext4 ได้ให้ความเห็นว่าในอนาคตอันใกล้นี้อาจจะมีระบบไฟล์ตัวใหม่เข้ามาทดแทนชื่อว่า Btrfs (B-tree filesystem) ที่ถูกพัฒนาโดยบริษัท Oracle เนื่องจากทั้ง ext3 และ ext4 เกิดขึ้นจากพื้นฐานเทคโนโลยีเก่าซึ่งอาจจะไม่เหมาะกับการเปลี่ยนแปลงของเทคโนโลยีการจัดเก็บข้อมูลในอนาคตที่ต้องการความเสถียรภาพ (reliability) ความสามารถในการขยายตัว (scalability) และง่ายต่อการดูแลจัดการ

ตัวอย่างการสร้างระบบไฟล์ชนิด ext2, ext3, ext4 โดยการใช้คำสั่ง mkfs.ext<*หมายเลข*> เพื่อสร้าง empty filesystem

```
$ mkfs.ext2 /dev/hda3
$ mkfs.ext3 /dev/sda2
$ mkfs.ext4 /dev/sda3
```

ในกรณีที่ต้องการสร้างระบบไฟล์สำหรับไฟล์ image เดิมที่มีอยู่แล้ว

```
$ mkfs.ext2 disk.img
```

ในกรณีที่ต้องการกำหนดขนาดระบบไฟล์ในรูปแบบไฟล์ image สามารถทำตามคำสั่งข้างล่างนี้

```
# dd if=/dev/zero of=rootfs.image bs=1MB count=1024
1024+0 records in
1024+0 records out
1024000000 bytes (1.0 GB) copied, 15.3415 s, 66.7 MB/s

# mkfs.ext3 rootfs.image
mke2fs 1.41.11 (14-Mar-2010)
rootfs.image is not a block special device.
Proceed anyway? (y,n) y
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
62592 inodes, 250000 blocks
12500 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=260046848
8 block groups
32768 blocks per group, 32768 fragments per group
7824 inodes per group
Superblock backups stored on blocks:
     32768, 98304, 163840, 229376

Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 25 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.

# mkdir /mnt/rootfs
# mount rootfs.image /mnt/rootfs/ -o loop
# rsync -a ~/busybox/_install/ /mnt/rootfs/
# chown -R root:root /mnt/rootfs/
# sync
# umount /mnt/rootfs
```

ข้อมูลที่เพิ่มเข้าไปใน /mnt/rootfs/ ก็จะถูกเก็บอยู่ใน rootfs.image เช่นเดียวกัน

สำหรับนักพัฒนาที่อยากจะสร้างไดเรกทอรีต่างๆเพื่อประกอบเป็น rootfs ขึ้นมาเอง นอกจากจะต้องมีความเข้าใจในโครงสร้างไดเรกทอรีของ rootfs เป็นอย่างดีแล้วจะต้องเข้าใจการสร้างไฟล์ชนิด device พื้นฐานสำคัญได้ ดังตัวอย่างการสร้าง rootfs ข้างล่างนี้

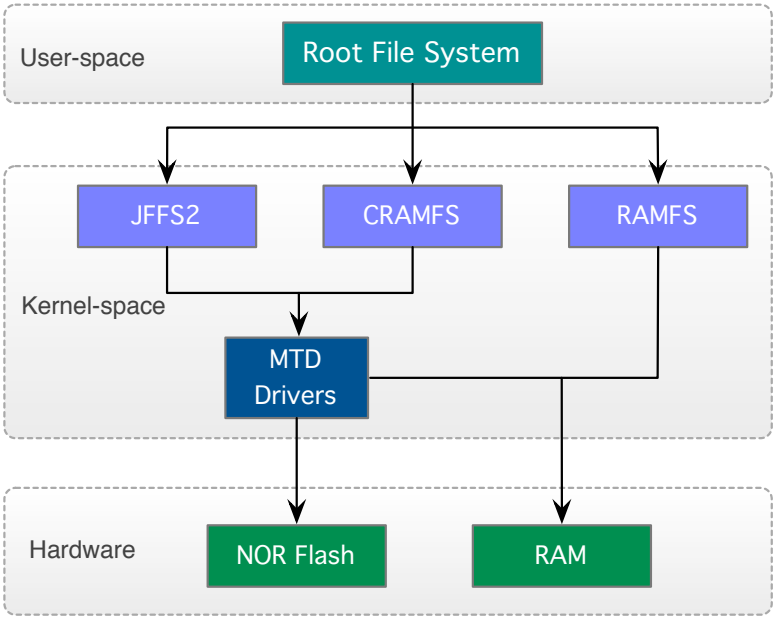
```
$ mkdir ~/rootfs
$ cd ~/rootfs

$ mkdir bin dev etc lib proc sbin tmp usr var
$ chmod 1777 tmp
$ mkdir usr/{bin,lib,sbin}
$ mkdir var/{lib,lock,log,run,tmp}
$ chmod 1777 var/tmp
# cd dev/
# mknod -m 600 mem c 1 1              // Physical memory access
# mknod -m 666 null c 1 3             // Null device
# mknod -m 666 zero c 1 5             // Null byte source
# mknod -m 644 random c 1 8           // Nondeterministic rnd nbr
# mknod -m 600 tty0 c 4 0             // Current virtual console
# mknod -m 600 tty1 c 4 1             // First virtual console
# mknod -m 600 ttyS0 c 4 64            // First UART serial port
# mknod -m 666 tty c 5 0               // Current TTY device
# mknod -m 600 console c 5 1           // System console

$ ln -s /proc/self/fd fd
$ ln -s /proc/self/fd/0 stdin
$ ln -s /proc/self/fd/1 stdout
$ ln -s /proc/self/fd/2 stderr
```

Embedded Linux File System

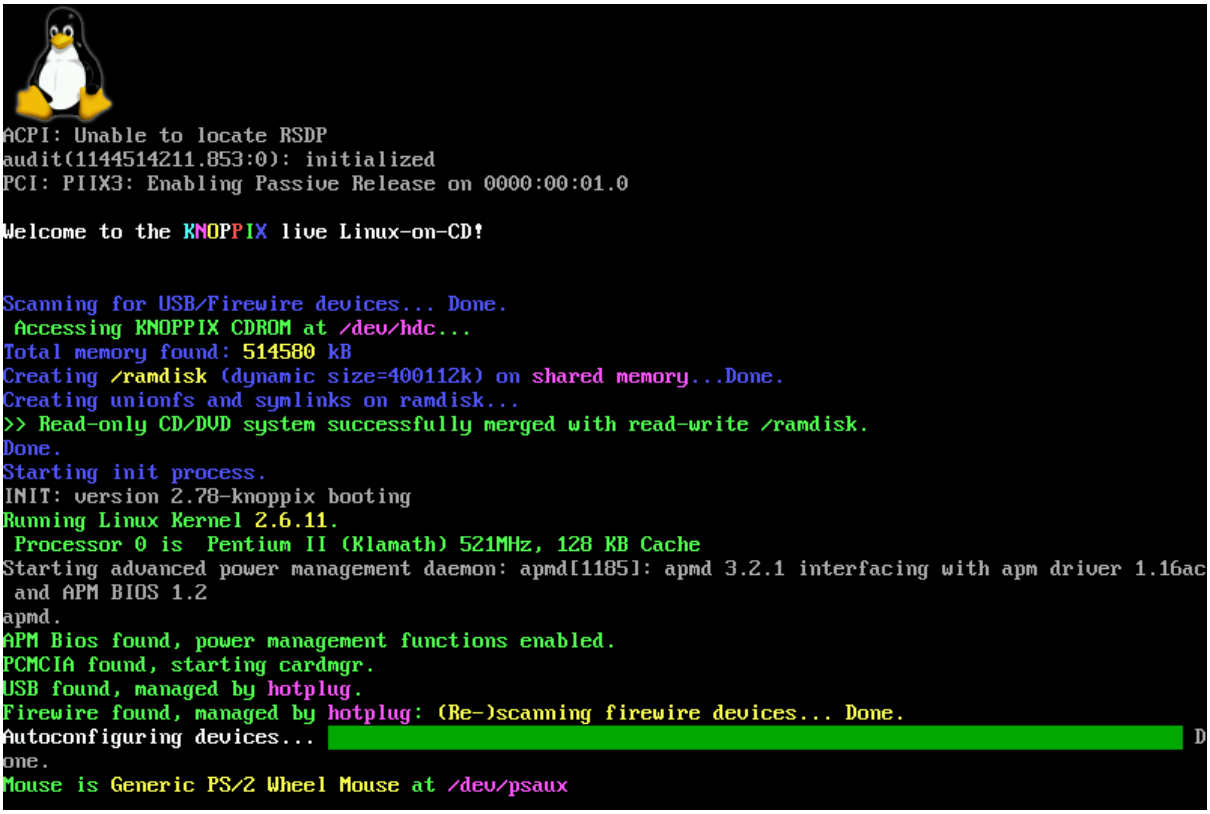
เทคโนโลยีจำนวนไม่น้อยที่มีการพัฒนาระบบ file system เพื่อรองรับระบบปฏิบัติการลีนุกซ์โดยเฉพาะอย่างยิ่งที่มีการใช้ระบบไฟล์แบบ Flash ซึ่งเหมาะกับการนำไปใช้ในเทคโนโลยีระบบสมองกลฝังตัว จากรูปข้างล่างแสดงถึงการใช้อุปกรณ์จัดเก็บข้อมูลชนิด NOR Flash ในขณะที่ในส่วนของ device driver จะถูกดูแลจัดการผ่านตัว MTD Drivers



รูปที่ 3.22 การจัดการอุปกรณ์จัดเก็บข้อมูลผ่าน MTD driver

นักพัฒนาทางด้านระบบสมองกลฝังตัวตั้งแต่ระดับพื้นฐานจะเจอกับไฟล์ที่ชื่อว่า initrd (initial RAM disk) ซึ่งเป็นระบบไฟล์ที่ถูกบีบอัด (compressed filesystem) โดยในช่วงเริ่มต้นของตัวระบบไฟล์ initrd นี้ได้ถูกพัฒนาขึ้นด้วยวัตถุประสงค์เพื่อเป็นตัวการในการนำระบบปฏิบัติการลีนุกซ์แบบย่อให้สามารถถูกนำไปบรรจุลงในแผ่นดิสก์แบบอ่อน (floppy disk) ได้และยังสามารถทำการบูทและติดตั้งระบบปฏิบัติการลีนุกซ์ตัวเต็มลงในฮาร์ดดิสก์ได้อีกด้วย ซึ่งหลังจากลีนุกซ์เคอร์เนลรุ่น 2.6 ได้ถูกปล่อยออกมา ตัวระบบไฟล์ initrd ก็ได้ถูกเรียกในชื่อใหม่ว่า “initramfs” (Initial RAM FileSystem)

ซึ่งในปัจจุบันจะพบเห็นในลักษณะแผ่น Live CD (เช่น Knoppix เป็นต้น) จนถึงการทำ Live USB กันมากขึ้น ซึ่ง Live CD หรือ Live USB นั้นโดยมีระบบปฏิบัติการลีนุกซ์แบบย่อที่สามารถถูกโหลดขึ้นมาอยู่บนหน่วยความจำของเครื่องคอมพิวเตอร์ (DDRAM) ได้ทันที โดยไม่จำเป็นต้องมีฮาร์ดดิสก์อยู่ภายในเครื่องแต่อย่างใดดังตัวอย่างในรูปข้างล่าง แต่อย่างไรก็ตามผู้ใช้ก็ยังสามารถติดตั้งระบบปฏิบัติการลีนุกซ์ลงในฮาร์ดดิสก์ผ่านเมนูในระหว่างบูทเข้าสู่ระบบได้เช่นกัน



รูปที่ 3.23 แสดงหน้าต่างการบูทเข้าสู่ระบบปฏิบัติการลีนุกซ์

การใช้ Live CD นั้นยังมีข้อจำกัดหลายอย่างกล่าวคือขนาดไฟล์ของ initramfs จะต้องมีขนาดไม่เกินความจุของแผ่น CD-ROM ทั่วไปซึ่งมีขนาด 700 MB เท่านั้น แต่ถ้าต้องการระบบปฏิบัติการที่มีขนาดใหญ่ขึ้น Live USB จึงกลายเป็นทางเลือกที่น่าสนใจที่สุดไม่ว่าจะเป็นระบบปฏิบัติการลีนุกซ์หรือระบบปฏิบัติการวินโดส์ เป็นต้น

ระบบไฟล์ในระบบสมองกลฝังตัว

ตัวอย่างระบบไฟล์ที่นิยมใช้ในระบบสมองกลฝังตัวได้แก่

ระบบไฟล์ชนิด cramfs

cramfs เป็นระบบไฟล์ถูกบีบอัดเอาไว้ และอ่านได้อย่างเดียว ซึ่งถูกพัฒนาขึ้นโดย Linus Torvalds และต่อมาก็ได้ถูกบรรจุอยู่ในลีนุกซ์คอร์เนลเป็นที่เรียบร้อยแล้ว ระบบไฟล์นี้จะถูกบีบอัดข้อมูลในลักษณะแยกเป็น เพจ (page) และสามารถเข้าถึงข้อมูลได้ในลักษณะไม่เรียงตามลำดับ (random access) ข้อดีของการทำให้พาร์ทิชั่นเป็นแบบอ่านได้อย่างเดียวของ cramfs นั้นคือการสร้างความน่าเชื่อถือของระบบเพราะจะไม่มีเหตุการณ์ไฟล์ระบบเสียหายหรือสูญหายอย่างแน่นอน

ระบบไฟล์ชนิด ramfs

ramfs ได้ถูกพัฒนาขึ้นโดย Linus Torvalds เช่นเดียวกัน ซึ่งจะทำการโหลดไฟล์ทั้งหมดขึ้นไว้ในหน่วยความจำเครื่อง (RAM) แต่จะเป็นระบบไฟล์ที่ถูกใช้อยู่บนอุปกรณ์ชนิด flash เป็นหลัก จึงทำให้สามารถเก็บข้อมูลที่มีการเปลี่ยนแปลงหรือเพิ่มเติมได้แต่จะไม่เหมือนกับระบบไฟล์ชนิด initrd หรือ initramfs ที่ถูกกำหนดขนาดตายตัวและไม่สามารถเปลี่ยนแปลงได้

ระบบไฟล์ชนิด jffs2

jffs2 เป็นระบบไฟล์ยอดนิยมตัวหนึ่งซึ่งสามารถเขียน/อ่าน และถูกบีบอัดในรูปแบบ journalling flash filesystem โดยตัว jffs2 นี้ได้ถูกนำไปใช้ในอุปกรณ์ชนิด Flash memory มากกว่าจะเป็นชนิด RAM เพราะสามารถเก็บข้อมูลที่เปลี่ยนแปลงได้เช่นเดียวกับระบบไฟล์ชนิด ramfs แต่จะมีข้อดีที่สำคัญอีกอย่างคือระบบไฟล์ jffs2 นั้นสามารถตรวจสอบและกู้คืนข้อมูลในขณะเกิดความผิดพลาดภายในระบบไฟล์ได้เหมือน ext2/ext3

ตาราง 3.3 แสดงรายละเอียดคุณสมบัติของแต่ละระบบไฟล์

File System	Write	Persistent	Power down reliability	Compression	Lives in RAM
CRAMFS	No	No	N/A	Yes	No
JFFS2	Yes	Yes	Yes	Yes	No
RAMFS	Yes	Yes	Yes	No	Yes
YAFFS2	Yes	Yes	Yes	No	No
Ext2 over NFTL	Yes	Yes	No	No	No
Ext3 over NFTS	Yes	Yes	Yes	No	No
Ext2 over RAM disk	Yes	No	No	No	Yes

Memory Technology Devices (MTD)

MTD เป็นชิฟหน่วยความจำแบบ flash ที่มีพื้นฐานจาก NAND/NOR flash ที่ถูกนำไปใช้ในการเก็บข้อมูลที่ไม่ค่อยถูกเปลี่ยนแปลงบ่อย (non-volatile data) ตัวอย่างเช่นไฟล์ boot image และเก็บค่า config ต่างๆภายใน bootloader

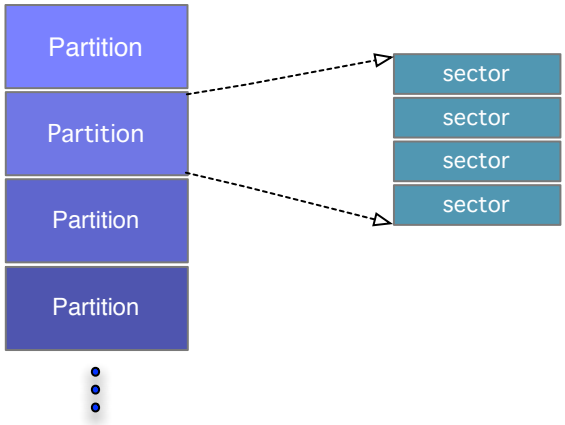
ถึงแม้ว่าอุปกรณ์ MTD เป็นชนิด flash แต่ระบบไฟล์จะไม่เหมือนกับอุปกรณ์ชนิด flash เช่น USB sticks, MMC/SD cards เพราะอุปกรณ์ MTD จะเป็นตัวเก็บข้อมูลที่สามารถถูกแบ่งเป็นพาร์ทิชั่นได้เหมือนฮาร์ดดิสก์ แต่ก็มีการทำงานที่แตกต่างจากฮาร์ดดิสก์และหน่วยความจำ (RAM) ในหลายๆด้านซึ่งจุดที่แตกต่างชัดเจนที่สุดคือเซกเตอร์ของฮาร์ดดิสจะสามารถถูกเขียนข้อมูลได้ทันที แต่ในขณะที่เซกเตอร์ของอุปกรณ์ MTD นั้นจะต้องถูกลบก่อนที่จะถูกเขียนลงไป (ที่นิยมเรียกกันว่า erase-block)

นอกจากนั้นจำนวนการเขียนของอุปกรณ์ MTD ก็จำกัดจำนวนครั้งเพียง 1,000 ถึง 10,000 ครั้งเท่านั้น แต่ในปัจจุบันอุปกรณ์ MTD ก็ยังถูกนำมาใช้หลักๆสำหรับบอร์ดสมองกลฝังตัวเพื่อเก็บ bootloaders และ Kernel image

ตัวอย่างการอ่านรายละเอียดของพาร์ทิชั่นของอุปกรณ์ MTD โดยอ่านผ่านไฟล์ /proc/mtd

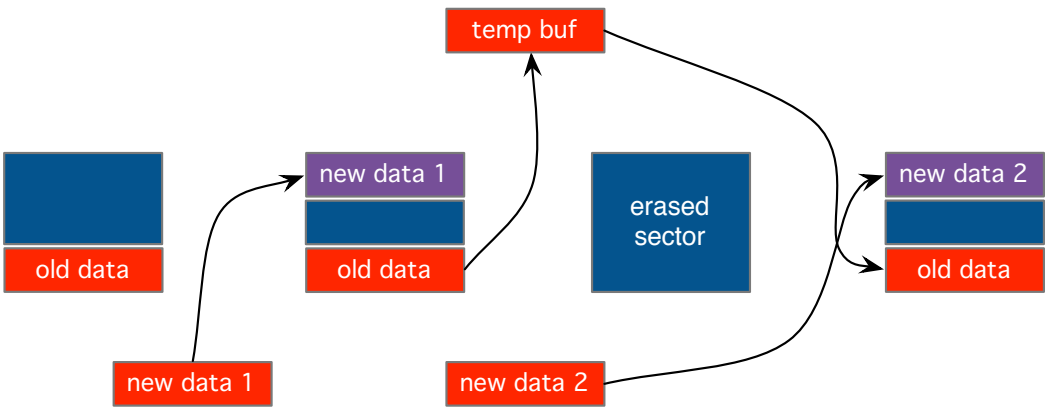
```
$ cat /proc/mtd
dev:  size    erasesize name
mtd0: 000a0000 00020000 "misc"
mtd1: 00420000 00020000 "recovery"
mtd2: 002c0000 00020000 "boot"
mtd3: 0fa00000 00020000 "system"
mtd4: 02800000 00020000 "cache"
mtd5: 0af20000 00020000 "userdata"
```

รูปแสดงการแบ่งเซกเตอร์ของแต่ละพาร์ทิชั่นภายในอุปกรณ์ MTD



รูปที่ 3.24 แสดงรายละเอียดของ sector ภายในพาร์ทิชัน

จากที่ได้อธิบายขบวนการเขียนข้อมูลลงในอุปกรณ์ MTD ข้างต้นว่าจะต้องมีการลบก่อนจะเขียนข้อมูลใหม่ลงไป สามารถดูขั้นตอนการทำงานได้จากรูปข้างล่างนี้



รูปที่ 3.25 แสดงขบวนการเขียนข้อมูลลงอุปกรณ์ MTD

จากรูปด้านบนแสดงขบวนการเขียนข้อมูลลงไปในอุปกรณ์ MTD สังเกตจากด้านซ้ายจะพบว่าหลังจากการลบครั้งล่าสุดก็เริ่มการเขียนข้อมูลใหม่ตัวแรก (new data 1) ลงในตำแหน่ง offset ที่ศูนย์ เมื่อผู้ใช้ต้องการจะเขียนข้อมูลใหม่ตัวถัดมา (new data 2) ลงในตำแหน่งเดิมของข้อมูลตัวแรก ตัวอุปกรณ์ MTD จะต้องคัดลอกข้อมูลเก่าที่อยู่ด้านล่างสุดไปพักไว้ในบัฟเฟอร์ชั่วคราวก่อน (temp buf) แล้วจึงสามารถดำเนินการลบเซกเตอร์นั้นทั้งหมดได้ แล้วจึงค่อยนำข้อมูลเก่า (temp buf) และข้อมูลตัวใหม่ (new data 2) มาเก็บตามลำดับ

ซึ่งขั้นตอนเหล่านี้จะถูกดำเนินการในชั้นของ flash translation layer (FTL) เพื่อลดความยุ่งยากในการเขียนโปรแกรมสำหรับผู้ใช้ ซึ่งเทคนิคนี้ได้ถูกนำไปใช้ในอุปกรณ์ MTD เช่น JFFS2 และ YAFFS โดยทั้ง JFFS2 และ YAFFS file systems นั้นจะไม่สามารถสร้างเป็น loopback device และไม่สามารถถูกทำการ mount ตัวระบบไฟล์ได้ (ด้วยคำสั่ง -o loop) เหมือนกับ linux filesystem ทั่วไป เนื่องจากไม่ได้เป็นอุปกรณ์ที่เป็นแบบ block ดังที่อธิบายไปข้างต้น ดังนั้นวิธีการที่จะให้สามารถอุปกรณ์ MTF สามารถถูก mount ตัวระบบไฟล์ของทั้งสองตัวได้ จะต้องทำการโหลด MTD Driver ชื่อว่า mtdram module เสียก่อน โดยโปรแกรมไดรเวอร์นี้จะทำการจำลองอุปกรณ์ MTD ขึ้นมาดังตัวอย่างคำสั่งข้างล่างนี้

```
# modprobe mtdram total_size=65536 erase_size=128
```

สามารถตรวจสอบอุปกรณ์ MTD ที่ถูกจำลองขึ้นด้วยคำสั่ง

```
# cat /proc/mtd
dev: size erasesize name
mtd0: 04000000 00020000 "mtdram test device"
```

ใช้คำสั่ง dd เพื่อสร้าง jffs2 file system และตั้งชื่ออุปกรณ์ใหม่ชื่อ mtdblock0

```
# dd if=rootfs.jffs2 of=/dev/mtdblock0
```

mount ตัว jffs2 filesystem ด้วยคำสั่ง

```
# mount -t jffs2 /dev/mtd0 ~/rootfs
```

ตัวอย่างการเขียนโปรแกรมเพื่อเข้าถึงอุปกรณ์ MTD

ตัวอย่างโปรแกรมแสดงรายละเอียดพื้นฐานของอุปกรณ์ MTD

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/ioctl.h>
```

```
#include <mtd/mtd-user.h>

int main()
{
    mtd_info_t mtd_info;
    int fd = open("/dev/mtd5", O_RDWR);
ioctl(fd, MEMGETINFO, &mtd_info);

    printf("MTD type: %u\n", mtd_info.type);
    printf("MTD total size : %u bytes\n", mtd_info.size);
    printf("MTD erase size : %u bytes\n", mtd_info.erasesize);

    return 0;
}
```

ตัวอย่างชุดคำสั่งเพื่อใช้ในการอ่านและเขียนข้อมูลลงในอุปกรณ์ MTD

```
unsigned char buf[64];
unsigned char data[7] = { 'B', 'U', 'R', 'A' , 'P' , 'H' , 'A' };
/* read something from last sector */
lseek(fd, -mtd_info.erasesize, SEEK_END);
read(fd, buf, sizeof(buf));
/* write something to last sector */
lseek(fd, -mtd_info.erasesize, SEEK_END);
write(fd, data, sizeof(data));
```

ตัวอย่างโปรแกรมประยุกต์เพื่อให้ติดต่อกับอุปกรณ์ MTD ผ่านไฟล์ device ชื่อ /dev/mtd0

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <mtd/mtd-user.h>

int main() {
      mtd_info_t mtd_info;          // the MTD structure
      erase_info_t ei;              // the erase block structure
      int i;

      unsigned char data[20] = { 0xDE, 0xAD, 0xBE, 0xEF,  // our data to write
            0xDE, 0xAD, 0xBE, 0xEF, 0xDE, 0xAD, 0xBE, 0xEF, 0xDE, 0xAD, 0xBE,
            0xEF, 0xDE, 0xAD, 0xBE, 0xEF };
      unsigned char read_buf[20] = { 0x00 }; // empty array for reading

      int fd = open("/dev/mtd0", O_RDWR); // open the mtd device for reading and
      // writing. Note you want mtd0 not mtdblock0
      // also you probably need to open permissions
      // to the dev (sudo chmod 777 /dev/mtd0)

      ioctl(fd, MEMGETINFO, &mtd_info);   // get the device info
```

```
	// dump it for a sanity check, should match what's in /proc/mtd
	printf("MTD Type: %x\nMTD total size: %x bytes\nMTD erase size: %x bytes\n",
mtd_info.type, mtd_info.size, mtd_info.erasesize);

	ei.length = mtd_info.erasesize;   //set the erase block size
	for (ei.start = 0; ei.start < mtd_info.size; ei.start += ei.length) {
		ioctl(fd, MEMUNLOCK, &ei);
	// printf("Eraseing Block %#x\n", ei.start); // show the blocks erasing
		// warning, this prints a lot!
		ioctl(fd, MEMERASE, &ei);
	}

	lseek(fd, 0, SEEK_SET);                // go to the first block
	read(fd, read_buf, sizeof(read_buf)); // read 20 bytes

	// sanity check, should be all 0xFF if erase worked
	for (i = 0; i < 20; i++)
		printf("buf[%d] = 0x%02x\n", i, (unsigned int) read_buf[i]);

	lseek(fd, 0, SEEK_SET);        // go back to first block's start
	write(fd, data, sizeof(data)); // write our message

	lseek(fd, 0, SEEK_SET);              // go back to first block's start
	read(fd, read_buf, sizeof(read_buf));   // read the data

	// sanity check, now you see the message we wrote!
	for (i = 0; i < 20; i++)
		printf("buf[%d] = 0x%02x\n", i, (unsigned int) read_buf[i]);

	close(fd);
	return 0;
}
```

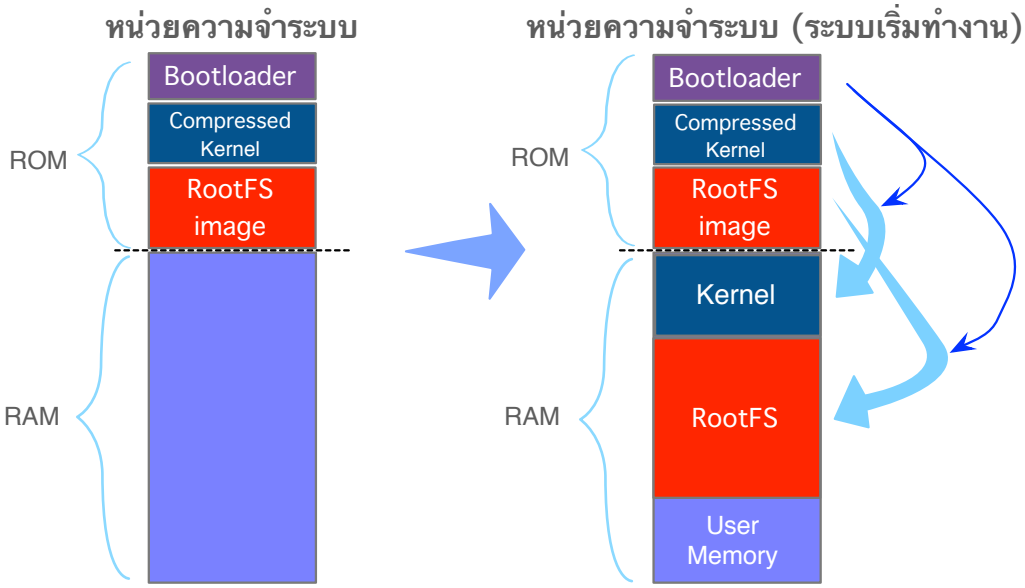
ขั้นตอนการโหลดระบบไฟล์เพื่อเข้าสู่ระบบปฏิบัติการ EMBEDDED LINUX

เพื่อทำให้ระบบสมองกลฝังตัวทำงานขึ้นมาได้นั้น จะต้องมีการเตรียมไฟล์หลักๆได้แก่ไฟล์ Kernel Image (initrd, vmlinux, bzImage, zImage เป็นต้น) และไฟล์ Root file system (ext2/3/4, cramfs, jffs2, hffs เป็นต้น) แล้วนำไปวางไว้ในพาร์ทิชั่นของอุปกรณ์จัดเก็บข้อมูลที่บอร์ดสมองกลฝังตัวสามารถเข้าถึงได้ ดังตำแหน่งพาร์ทิชั่นเหล่านี้

- **พาร์ทิชั่นบนฮาร์ดดิสก์ หรือ USB Storage**
  - `root=/dev/sdXY` ที่ซึ่ง `X` แทนด้วยชื่ออุปกรณ์ และ `Y` แทนด้วยหมายเลขพาร์ทิชั่น เช่น `root=/dev/sdb2` *(พาร์ทิชั่นที่ 2 ของดิสก์ที่ 2)*
- **พาร์ทิชั่นบน MMC/SD Card**
  - `root=/dev/mmcblkXpY` ที่ซึ่ง `X` แทนด้วยหมายเลขอุปกรณ์ และ `Y` แทนด้วยหมายเลขพาร์ทิชั่น เช่น `root=/dev/mmcblk0p2` *(พาร์ทิชั่นที่ 2 ของดิสก์แรก)*

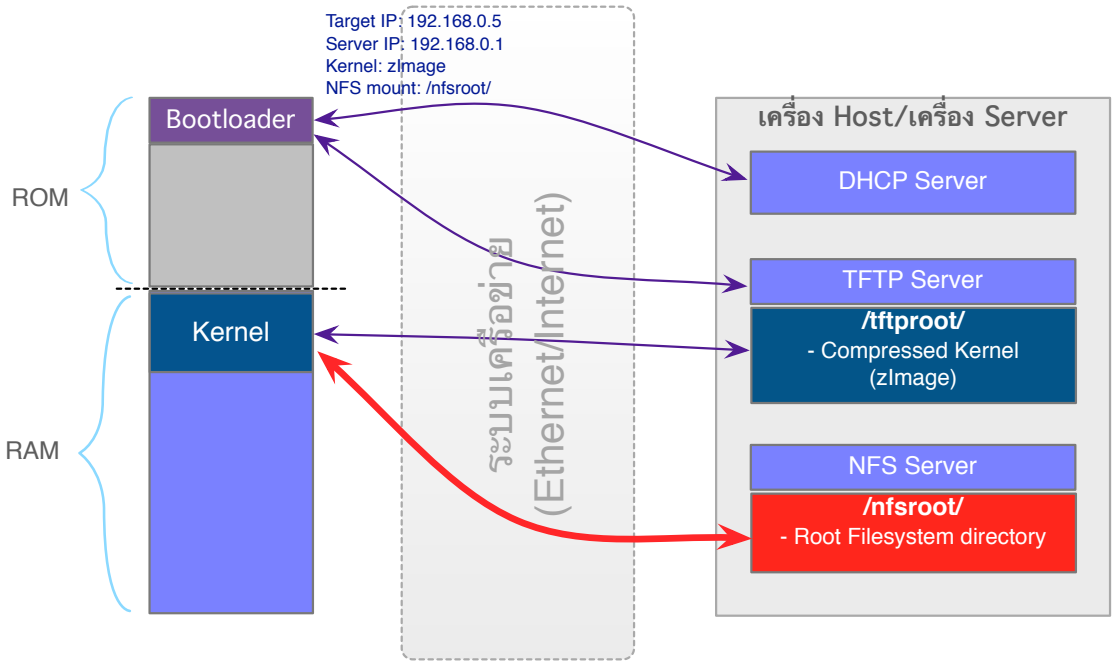
- **พาร์ทิชั่นบน NAND flash**
  - `root=/dev/mtdblockX` ที่ซึ่ง `X` แทนหมายเลขพาร์ทิชั่น เช่น `root=/dev/mtdblock3` *(พาร์ทิชั่นที่ 4)*
- **พาร์ทิชั่นบน NFS ของเครื่องภายในระบบเครือข่าย**
- **หน่วยความจำภายใน (ROM)**

รูปข้างล่างแสดงการใช้พื้นที่ของหน่วยความจำหลักของระบบ (RAM) ในขณะที่อยู่ในสภาวะก่อนทำงานและสภาวะกำลังทำงาน และแสดงขั้นตอนการโหลด Kernel image และ root filesystem ที่ถูกเก็บอยู่ภายใน ROM ของเครื่อง



รูปที่ 3.26 แสดงการโหลดเคอร์เนล และ RootFS ภายในบอร์ด

รูปข้างล่างแสดงขั้นตอนการโหลด Kernel image และ root filesystem ที่ถูกเก็บอยู่ในเครื่องอื่น ที่อยู่ภายในระบบเครือข่าย



รูปที่ 3.27 แสดงการโหลดเคอร์เนลและ RootFS จากเครื่อง server

ตารางแสดงการรองรับการเข้าถึง filesystem ที่นิยมสูงสุดในการเก็บข้อมูล 5 อันดับแรกของระบบปฏิบัติการยอดนิยมในปัจจุบัน

ตาราง 3.4 แสดงการรองรับการเข้าถึงระบบไฟล์ชนิดต่างๆของแต่ละระบบปฏิบัติการ

FS	WINDOWS	MAC	LINUX	ANDROID	CHROME OS	IPAD	XBOX360	PS3
FAT32	Yes	Yes	Yes	Yes	Yes	(1)	Yes	Yes
EXFAT	Yes(2)	No(3)	(3)	Yes	No	No	No	No
EXt3	(4)	Yes	Yes	No(6)	Yes	No	No	No
NTFS	Yes	Yes	Yes	(8)	Yes	No	No[9]	No
HFS+	No[10]	Yes	(11)	(8)	(12)	(1)	No	No

โดยที่:

(1) *iPads (ไม่รวม iPhones) สามารถอ่านอุปกรณ์จัดเก็บข้อมูล ที่ใช้ระบบไฟล์ FAT และ HFS ได้ แต่จะอ่านไฟล์ที่เป็นชนิดรูปภาพ และวิดีโอได้เท่านั้น*

(2) *exFAT ที่ปรับปรุงมาจาก FAT ซึ่งทำงานได้ดีบนระบบปฏิบัติการ Vista / Windows 7 / Windows 8, และ XP*

(3) *ต้องติดตั้งโปรแกรม FUSE เพิ่มเข้าไป เพื่อให้สามารถอ่านและเขียนข้อมูลได้*

(4) *ระบบปฏิบัติการ Windows จะต้องติดตั้งโปรแกรม explore2fs, Ext2Fsd หรือ Ext2IFS เพื่อให้สามารถอ่านระบบไฟล์ประเภท ext2/ext3/ext4 ได้*

(5) *จะต้องติดตั้งโปรแกรม OSXFUSE fuse-ext2, หรือ Paragon*

(6) *เฉพาะบาง ROMs ที่ถูกปรับแต่งให้สนับสนุนระบบไฟล์ ext3 เท่านั้น*

(7) ระบบปฏิบัติการ MacOS โดยพื้นฐานสามารถอ่านระบบไฟล์ NTFS ได้อย่างเดียว ถ้าต้องการเขียนได้ จะต้องติดตั้ง OSXFUSE with NTFS-3G, Paragon หรือ Tuxera

(8) บนระบบปฏิบัติการ Android ต้องติดตั้งโปรแกรม Paragon app เพื่อให้สามารถเข้าถึง ระบบไฟล์ NTFS และ HFS ได้

(9) Xbox 360 ของบริษัท Microsoft เอง ไม่สนับสนุนระบบไฟล์ NTFS

(10) จะต้องติดตั้งโปรแกรม Paragon เพิ่มเติม

(11) ระบบปฏิบัติการลีนุกซ์ สามารถอ่านระบบไฟล์ HFS+ ได้อย่างเดียว ถ้าต้องการเขียนได้ ก็จะต้องปิดความสามารถของ journalling ไป

(12) สามารถอ่านระบบไฟล์ journaled HFS+ ได้อย่างเดียว

BusyBox มีดพกพาสารพัดประโยชน์

BusyBox ได้ถูกพัฒนาขึ้นโดยนาย Bruce Perens ในปี ค.ศ. 1996 ซึ่งเดิมทีถูกนำมาใช้เป็นโปรแกรมช่วยในการติดตั้งและกู้คืนในแผ่นดิสก์เก็ตแบบบูทตัวเองได้ของระบบปฏิบัติการ Debian (bootable Debian Linux system) โดยมีเงื่อนไขว่าลีนุกซ์คอร์เนลและโปรแกรมพื้นฐานทั้งหมดจะมีขนาดรวมกันแล้วไม่เกิน 1.4 MB ถึง 1.7 MB เนื่องจากข้อจำกัดในการเก็บข้อมูลภายในแผ่นดิสก์เก็ตในยุคนั้น ดังนั้นหลักการสำคัญที่จะต้องทำให้ BusyBox มีขนาดเล็กแต่ยังคงเต็มไปด้วยชุดคำสั่งมากมายคือการใช้ส่วนฟังก์ชันพื้นฐานที่เหมือนกันร่วมกันตัวอย่างเช่น ตัว Busy-Box จะทำการรวมโปรแกรม grep และ find เป็นโปรแกรมเดียวกันเนื่องจากใช้หลักการค้นหาไฟล์โดยจะเข้าไปถึงภายในไดเรกทอรีทุกไดเรกทอรี (recursive) เช่นเดียวกัน เป็นผลให้ขนาดไฟล์ที่เกิดจากการรวมโปรแกรมทั้งสองลดลงอย่างมาก ดังนั้น BusyBox ก็สามารถใช้หลักการนี้กับโปรแกรมอื่นๆที่มีฟังก์ชันพื้นฐานเดียวกันจากเดิมที่มีขนาดทั้งสิ้น 3.5 MB ก็สามารถลดลงเหลือเพียง 200 KB เท่านั้นได้



รูปที่ 3.28 มีด Swiss Army Knife

ดังนั้น BusyBox จึงกลายมาเป็นเครื่องมือสำคัญที่ถูกนำมาใช้ภายในระบบปฏิบัติการของระบบสมองกลฝังตัวอื่นๆจนถูกตั้งฉายาว่าเป็น “มีดพกพาสารพัดประโยชน์” (Swiss Army Knife) ซึ่งได้รวมชุดโปรแกรมและชุดคำสั่งอรรถประโยชน์ของคำสั่งพื้นฐานต่างๆที่จำเป็นในระบบปฏิบัติการลีนุกซ์ ตัวอย่างเช่น ชุดโปรแกรมที่ใช้จัดการบริการต่างๆของระบบในขณะเครื่องกำลังเริ่มทำงาน (Init program), ชุดคำสั่งจัดการระบบและตั้งค่าระบบ (System & Configuration) เป็นต้น ตัวอย่างคำสั่งใน BusyBox version 1.13 ได้แก่

```
[, [[ , addgroup, adduser, adjtimex, ar, arp, arping, ash, awk, base-
name, bbconfig, bbsh, brctl, bunzip2, busybox, bzcat, bzip2, cal, cat,
catv, chat, chattr, chcon, chgrp, chmod, chown, chpasswd, chpst,
chroot, chrt, chvt, cksum, clear, cmp, comm, cp, cpio, crond, crontab,
```

```
cryptpw, cttyhack, cut, date, dc, dd, deallocvt, delgroup, deluser,
depmod, devfsd, df, dhcprelay, diff, dirname, dmesg, dnsd, dos2unix,
dpkg, dpkg_deb, du, dumpkmap, dumpleases, e2fsck, echo, ed, egrep,
eject, env, envdir, envuidgid, ether_wake, expand, expr, fakeidentd,
false, fbset, fbsplash, fdflush, fdformat, fdisk, fetchmail, fgrep,
find, findfs, fold, free, freeramdisk, fsck, fsck_minix, ftpget,
ftpput, fuser, getenforce, getopt, getsebool, getty, grep, gunzip,
gzip, halt, hd, hdparm, head, hexdump, hostid, hostname, httpd, hush,
hwclock, id, ifconfig, ifdown, ifenslave, ifup, inetd, init, inotifyd,
insmod, install, ip, ipaddr, ipcalc, ipcrm, ipcs, iplink, iproute,
iprule, iptunnel, kbd_mode, kill, killall, killall5, klogd, lash,
last, length, less, linux32, linux64, linuxrc, ln, load_policy, load-
font, loadkmap, logger, login, logname, logread, losetup, lpd, lpq,
lpr, ls, lsattr, lsmod, lzmacat, makedevs, man, matchpathcon, md5sum,
mdev, mesg, microcom, mkdir, mke2fs, mkfifo, mkfs_minix, mknod,
mkswap, mktemp, modprobe, more, mount, mountpoint, msh, mt, mv,
nameif, nc, netstat, nice, nmeter, nohup, nslookup, od, openvt, parse,
passwd, patch, pgrep, pidof, ping, ping6, pipe_progress, pivot_root,
pkill, poweroff, printenv, printf, ps, pscan, pwd, raidautorun, rdate,
rdev, readahead, readlink, readprofile, realpath, reboot, renice, re-
set, resize, restorecon, rm, rmdir, rmmod, route, rpm, rpm2cpio,
rtcwake, run_parts, runcon, runlevel, runsv, runsvdir, rx, script,
sed, selinuxenabled, sendmail, seq, sestatus, setarch, setconsole, se-
tenforce, setfiles, setfont, setkeycodes, setlogcons, setsebool, set-
sid, setuidgid, sh, sha1sum, showkey, slattach, sleep, softlimit,
sort, split, start_stop_daemon, stat, strings, stty, su, sulogin, sum,
sv, svlogd, swapoff, swapon, switch_root, sync, sysctl, syslogd, tac,
tail, tar, taskset, tcpsvd, tee, telnet, telnetd, test, tftp, tftpd,
time, top, touch, tr, traceroute, true, tty, ttysize, tune2fs, udhcpc,
udhcpd, udpsvd, umount, uname, uncompress, unexpand, uniq, unix2dos,
unlzma, unzip, uptime, usleep, uudecode, uuencode, vconfig, vi, vlock,
watch, watchdog, wc, wget, which, who, whoami, xargs, yes, zcat, zcip
```

ตัวอย่างข้างล่างแสดงแนวคิดการสร้างโปรแกรมเพื่อที่จะทำให้กลายเป็นการรันมาจากเพียงโปรแกรมเดียวให้ผู้ใช้รู้สึกว่ามีหลายคำสั่งโปรแกรมให้เรียกใช้งานได้ แต่จริงๆแล้วเป็นชุดคำสั่งที่อยู่ภายในโปรแกรมเพียงตัวเดียวเท่านั้น เช่นเดียวกันโปรแกรม BusyBox ที่จะใช้หลักการการส่งผ่านค่าอาร์กิวเม้นต์ (argument passing) ไปยังฟังก์ชันภาษาซี

```c
// myBusyBox.c
#include <stdio.h>
int main( int argc, char *argv[] )
{
  int i;

  for (i = 0 ; i < argc ; i++) {
    printf("argv[%d] = %s\n", i, argv[i]);
  }

  return 0;
```

```
}
```

จากข้างบนตัว argc เป็นตัวแปรเก็บจำนวนของอาร์กิวเม้นต์ และ argv เป็นตัวแปรชนิดอะเรย์ที่เก็บข้อความแต่ละอาร์กิวเม้นต์

หลังจากนั้นทำการคอมไพล์โปรแกรม และทดสอบการเรียกโปรแกรมพร้อมการส่งผ่านค่าอาร์กิวเม้นต์ดังนี้

```
$ gcc -o myBusyBox myBusyBox.c
$ ./myBusyBox arg1 arg2
argv[0] = ./myBusyBox
argv[1] = arg1
argv[2] = arg2
```

สร้างไฟล์ชนิด symbolic link ชื่อว่า my_grep และ my_find แล้วให้ชี้ไปยังชื่อโปรแกรม myBusyBox เดิม

```
$ ln -s myBusyBox my_grep
$ ln -s myBusyBox my_find
$ ls -al my*
-rwxr-xr-x 1 student student 8503 2013-08-13 08:21 myBusyBox
-rw-r--r-- 1 student student  162 2013-08-13 08:21 myBusyBox.c
lrwxrwxrwx 1 student student    9 2013-08-13 08:27 my_find -> myBusyBox
lrwxrwxrwx 1 student student    9 2013-08-13 08:22 my_grep -> myBusyBox

$ ./my_grep -R 'declare' /usr/include/
argv[0] = ./my_grep
argv[1] = -R
argv[2] = declare
argv[3] = /usr/include/

$ ./my_find /usr/include/ -name 'declare' -print
argv[0] = ./my_find
argv[1] = /usr/include/
argv[2] = -name
argv[3] = declare
argv[4] = -print
```

จากโปรแกรมข้างต้นเป็นการแสดงให้เห็นแนวคิดและวิธีการทำงานของโปรแกรม BusyBox ซึ่งโปรแกรม myBusyBox ที่ได้สร้างขึ้นนั้นจะมีฟังก์ชันพื้นฐานเกี่ยวกับการค้นหาข้อมูล (grep) และค้นหาไฟล์ (find) อยู่ภายใน ดังนั้นเมื่อผู้ใช้ต้องการเรียกใช้คำสั่งใดก็ตามจะสังเกตเห็นได้ว่าค่าใน argv[0] จะบรรจุชื่อคำสั่งที่ผู้ใช้พิมพ์ ดังนั้นนักพัฒนาก็สามารถทำการพัฒนาฟังก์ชันต่างๆเพื่อแทนคำสั่งนั้นๆต่อไปได้ แล้วตัวโปรแกรม myBusyBox ก็จะนำค่าภายใน argv[0] ไปตรวจเช็คต่อไปว่าเป็นตัวฟังก์ชันใด

ด้วยแนวคิดการรวมฟังก์ชันพื้นฐานที่ใช้ร่วมกันของชุดคำสั่งต่างๆในระบบปฏิบัติการลีนุกซ์ให้กลายมาเป็น BusyBox เพียงโปรแกรมเดียวนั้น จะสามารถลดขนาดโปรแกรมลงได้ถึง 3 ถึง 4 เท่าเมื่อเทียบกับการใช้โปรแกรมพื้นฐานทั้งหมด ซึ่งจะสังเกตจากการใช้คำสั่ง ls -al ดังรูปข้างล่างว่าทุกคำสั่งจะถูกชี้ (symbolic link) ไปยังโปรแกรม busybox ทั้งสิ้น

```
student@EE-Burapha:~/Downloads/busybox-1.17.1/_install/bin$ ls -al
total 868
drwxr-xr-x  2 student student   4096 2013-08-05 07:11 .
drwxr-xr-x 10 student student   4096 2013-08-05 07:16 ..
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 addgroup -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 adduser -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 ash -> busybox
-rwxr-xr-x  1 student student 879928 2013-08-05 07:11 busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 cat -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 catv -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 chattr -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 chgrp -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 chmod -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 chown -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 cp -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 cpio -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 cttyhack -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 date -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 dd -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 delgroup -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 deluser -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 df -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 dmesg -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 dnsdomainname -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 dumpkmap -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 echo -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 ed -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 egrep -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 false -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 fdflush -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 fgrep -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 fsync -> busybox
lrwxrwxrwx  1 student student      7 2013-08-05 07:11 getopt -> busybox
```

Busybox Inittab

เมื่อต้องมีการปรับแต่งหรือสร้าง BSP (Board Support Package) ของตัวเองขึ้นมาเพื่อใช้สำหรับระบบปฏิบัติการลีนุกซ์แบบฝังตัว การตั้งค่าการโหลดเริ่มต้น (Initialization) เพื่อใช้ในการกำหนดลำดับการโหลดโปรแกรมสำหรับตั้งค่าบริการของระบบ (System services) ต่างๆ หรือใช้ในการโหลดโปรแกรมระบบ โปรแกรมประยุกต์ที่พัฒนาขึ้นเองในระหว่างที่บอร์ดสมองกลฝังตัวกำลังเริ่มต้นการทำงาน ดังนั้นนักพัฒนาควรจะทำความเข้าใจวิธีการสร้างไฟล์ Inittab ที่อยู่ภายในไดเรกทอรี /etc/ ซึ่งภายในไฟล์ inittab นั้นมีรูปแบบการเขียนดังนี้

```
< id >:< runlevels >:< action >:< process >
```

โดยที่:

- < id > จะถูกใช้โดย BusyBox init เพื่อระบุ console ที่เปิดขึ้นมา (controlling tty) สำหรับโปรเซสที่ได้ระบุไว้แต่อย่างไรก็ตามถ้า BusyBox ตรวจพบว่ามีการใช้ serial console อยู่แล้วค่านี้ก็จะถูกเพิกเฉยไป
- < runlevels > ค่านี้จะไม่มีการใช้ใน BusyBox โดยเฉพาะในระบบสมองกลฝังตัวเนื่องจากไม่การแยกระดับของผู้ใช้ แต่จะใช้เฉพาะในเครื่องคอมพิวเตอร์เท่านั้น
- < action > ประกอบด้วย sysinit, respawn, askfirst, wait, once, restart, ctrlaltdel, และ shutdown โดยมีรายละเอียดดังนี้
  - sysinit กำหนดให้เรียกโปรเซสในระหว่างที่ระบบกำลังจะเริ่มถูกบูทขึ้น
  - respawn กำหนดให้เรียกโปรเซสใหม่อีกครั้งอัตโนมัติ เมื่อใดก็ตามที่โปรเซสนั้นถูกปิดลง
  - askfirst มีการทำงานคล้ายกับ respawn แต่ก่อนจะเริ่มเรียกโปรเซสที่กำหนดไว้ จะมีการแสดงข้อความดังนี้ "`Please press Enter to activate this console.`" เพื่อรอให้ผู้ใช้กดปุ่ม Enter แล้วจึงทำการเรียกโปรเซสลำดับถัดไป
  - ctrlaltdel กำหนดให้เรียกโปรเซส เมื่อ init ได้รับสัญญาณ SIGINT ที่มีการกดปุ่ม CTRL-ALT-DEL พร้อมกัน

ตัวอย่างไฟล์ inittab

```
# Boot-time system configuration/initialization script.
# This is run first except when booting in single-user mode.
#
::sysinit:/etc/init.d/rcS

# Put a getty on the serial line (for a terminal)
::respawn:/sbin/getty -L ttyS 115200 vt100

# Stuff to do when restarting the init process
::restart:/sbin/init

# Stuff to do before rebooting
::ctrlaltdel:/sbin/reboot
```

การพัฒนาระบบสมองกลฝังตัวภายใต้ระบบจำลองเสมือนจริง

สมัยก่อนนักพัฒนาทางด้านระบบสมองกลฝังตัวค่อนข้างจะมีทางเลือกน้อยในการหาซื้อบอร์ดสมองกลฝังตัวที่ใช้สถาปัตยกรรมที่ต้องการนำมาศึกษาเพื่อทดลองพัฒนาโปรแกรมหรือปรับแต่งระบบตามโจทย์ในงานประยุกต์นั้นๆ แต่ไม่กี่ปีที่ผ่านมาระบบสมองกลฝังตัวก็เริ่มเข้ามามีบทบาทอยู่ในชีวิตประจำวันมากยิ่งขึ้น ทำให้เกิดการสร้างสรรนวัตกรรมใหม่ๆ ผลิตภัณฑ์ใหม่ๆในวงกว้าง ส่งผลดีต่อนักพัฒนาในปัจจุบันที่สามารถหาซื้อบอร์ด Evaluation Kit ที่มีวางจำหน่ายจากบริษัทต่างๆ ตัวอย่างเช่นบอร์ด MicroBlaze, Sitara, Beagleboard, Beaglebone, Pandaboard, FriendlyARM, Raspberry Pi, ODROID, Versatile Express, Gumstix เป็นต้น โดยจะมีราคาตั้งแต่หลักพันกว่าบาทถึงหลายหมื่นบาท ทำให้มีตัวเลือกมากยิ่งขึ้น ทำให้นักพัฒนาอิสระ นักเรียนนักศึกษา หรือนักพัฒนาในบริษัทต่างๆสามารถเริ่มเรียนรู้เทคโนโลยีทางด้านระบบสมองกลฝังตัวได้อย่างลึกซึ้งมากขึ้น ซึ่งคาดว่าในอีกไม่กี่ปีข้างหน้ามูลค่าทางด้านการออกแบบและพัฒนาทางด้านระบบสมองกลฝังตัวนี้จะสูงมากกว่ามูลค่าจากการผลิตอย่างเห็นได้ชัด

ตาราง 3.5 บอร์ดสมองกลฝังตัวแต่ละยี่ห้อที่ได้รับความนิยม

บอร์ด	SoC	ความเร็ว	RAM	I/O	ราคา
BeagleBone	Sitara AM3358	500MHz (on USB)/ 720MHz (on DC)	256MB	USB OTG, USB Host, Ethernet, onboard Serial, onboard JTAG, expansion headers, microSD	$89
BeagleBoard xM	Davinci DM3730	1GHz	512MB	USB OTG, USB Host, Ethernet, expansion headers, microSD, DVI-D, LCD header, S-Video, Camera header, Stereo IN/OUT	$149
iMX53 QSB	i.MX53	1GHz	1GB	USB OTG, USB Host, Ethernet, Serial, expansion headers, SD, microSD, SATA, VGA, LCD header, Stereo IN/OUT	$149
PandaBoard ES	OMAP4 dual-core	1.2GHz	1GB	USB OTG, USB Host, Ethernet, WLAN, Bluetooth, Serial, expansion headers, SD, HDMI, DVI, LCD header, Camera header, Stereo IN/OUT	$182
AM335x Starter Kit	Sitara AM3358	720MHz	256MB	USB OTG, USB Host, Ethernet, WLAN, Bluetooth, onboard Serial, onboard JTAG, expansion headers, microSD, capacity-touch LCD, Accelerometer, Stereo OUT	$199
OrigenBoard	Exynos 4210 dual-core	1.2GHz	1GB	USB OTG, USB Host, Ethernet, WLAN, Bluetooth, Serial, JTAG, SD, HDMI, LCD header, Camera header, Stereo IN/OUT	$199
Origen 4 Quad	Exynos 4 quad-core	1.4GHz	1GB	USB OTG, USB Host, Ethernet, SD, JTAG, Serial, HDMI, onboard LCD header, audio	$199

บอร์ด	SoC	ความเร็ว	RAM	I/O	ราคา
ODROID-X2	Quad core ARM Cortex-A9 MPCore	1.7GHz	2GB	USB OTG, USB Host, Ethernet, onboard Serial, expansion headers, SD, Stereo IN/OUT	$135
DragonBoard APQ8060A	Snapdragon dual-core	1.2GHz	1GB	USB OTG, USB Host, Ethernet, WLAN, Bluetooth, GPS, onboard Serial, onboard JTAG, expansion headers, microSD, capacity-touch LCD, Accelerometer, Stereo OUT	$499
Snapdragon MDP	Snapdragon S4 dual-core	1.5GHz	1GB	USB OTG, WLAN, Bluetooth, GPS, Accelerometer, Gyroscope, Compass, Proximity sensor, Temperature sensor, SD, HDMI, LCD Panel , Camera, Stereo OUT	$999
RaspBerry Pi	Broadcom ARM11 BCM2835	700MHz	512MB	USB OTG, USB Host, onboard Ethernet, onboard Serial, expansion headers, MMC, SD, VGA, HDMI, Stereo OUT	$35
SABRE	i.MX53	1GHz	1GB	USB OTG, USB Host, Ethernet, WLAN, Bluetooth, GPS, ZigBee, Accelerometer, Light sensor, Serial, JTAG, eMMC, SD, SATA, NOR Flash, VGA, HDMI, LCD Panel, Camera, Stereo IN/OUT	$999

สำหรับอีกทางเลือกหนึ่งที่เหมาะสำหรับนักพัฒนามือใหม่ หรือนักพัฒนาที่ยังไม่ได้ตัดสินใจซื้อบอร์ดสมองกลฝังตัวคือการใช้โปรแกรมจำลองเสมือนจริง (Virtualization) เพื่อจำลองสภาพแวดล้อมและการทำงานให้เป็นไปตามสถาปัตยกรรมของหน่วยประมวลผลกลางนั้นเช่น ARM, MIPS, PowerPC, x86 เป็นต้น ซึ่งโปรแกรมจำลองเสมือนจริงที่นิยมกันมานานและใช้กันในระบบปฏิบัติการลีนุกซ์ก็คือโปรแกรม QEMU (Quick EMUlator) นักพัฒนาสามารถพัฒนาโปรแกรม และทดสอบการทำงานของโปรแกรมให้เป็นไปตามเงื่อนไขที่กำหนดบนสถาปัตยกรรมนั้นๆ นอกจากนั้นก็ยังสามารถปรับแต่งลีนุกซ์คอร์เนล หรือทดสอบโปรแกรมที่พัฒนาในระดับเคอร์เนลว่าสามารถทำงานได้ดีบนสถาปัตยกรรมนั้นหรือไม่ โปรแกรม QEMU สามารถรองรับโหมดการทำงานได้ 2 แบบได้แก่

A. *user-mode emulation* คือการอนุญาตให้โปรเซสที่ถูกออกแบบมาให้ทำงานบนหน่วยประมวลผล กลางตัวเดียว (เช่นบนบอร์ด target) ให้สามารถทำงานได้บนหน่วยประมวลผลกลางตัวอื่นๆได้ (เช่นบนเครื่อง Host) ด้วยตัว Dynamic Translator ตัวอย่างเช่น การเรียกตัวจำลองเลียนแบบ Wine Windows API (http://www.winehq.org) หรือเพื่อช่วยให้การ cross-compile และ cross-debugging ข้ามสถาปัตยกรรมสะดวกขึ้น นอกจากนั้นก็เป็นการทำให้โปรเซสสามารถเข้าถึงทรัพยากรของเครื่อง Host ซึ่งสถาปัตยกรรมที่รองรับสำหรับโหมดนี้ได้แก่ x86, PowerPC, ARM, 32-bit MIPS, Sparc32/64, ColdFire(m68k), CRISv32 and MicroBlaze CPUs

B. *System-mode emulation* คือการจำลองระบบสถาปัตยกรรมทั้งระบบ ไม่ว่าจะเป็นหน่วยประมวลผลกลาง อุปกรณ์ฮาร์ดแวร์รอบข้างต่างๆ เปรียบเสมือนเป็นเครื่องเสมือนจริง (Virtual Machine) ตัวอย่างเช่นการสร้างเครื่องเสมือนจริงที่ใช้สถาปัตยกรรม PowerPC ที่กำลังใช้ระบบ

ปฏิบัติการ Debian แต่จริงๆแล้วตัวระบบจำลองนี้ทำงานอยู่บนเครื่องคอมพิวเตอร์ที่ใช้สถาปัตยกรรม x86-64 บิต ภายใต้ระบบปฏิบัติการ Ubuntu เป็นต้น ตารางข้างล่างแสดงรายการสถาปัตยกรรมที่ QEMU รองรับ ได้แก่

ตาราง 3.6 สถาปัตยกรรมต่างๆที่ถูกสนับสนุนในโปรแกรม QEMU

สถาปัตยกรรม	
PC (x86 or x86_64 processor)	Luminary Micro LM3S6965EVB (ARM Cortex-M3)
ISA PC (old style PC without PCI bus)	Freescale MCF5208EVB (ColdFire V2)
PREP (PowerPC processor)	Arnewsh MCF5206 evaluation board (ColdFire V2)
G3 Beige PowerMac (PowerPC processor)	Palm Tungsten|E PDA (OMAP310 processor)
Mac99 PowerMac (PowerPC processor, in progress)	N800 and N810 tablets (OMAP2420 processor)
Sun4m/Sun4c/Sun4d (32-bit Sparc processor)	MusicPal (MV88W8618 ARM processor)
Sun4u/Sun4v (64-bit Sparc processor, in progress)	Gumstix "Connex" and "Verdex" motherboards (PXA255/270)
Malta board (32-bit and 64-bit MIPS processors)	Siemens SX1 smartphone (OMAP310 processor)
MIPS Magnum (64-bit MIPS processor)	Syborg SVP base model (ARM Cortex-A8)
ARM Integrator/CP (ARM)	AXIS-Devboard88 (CRISv32 ETRAX-FS)
ARM Versatile baseboard (ARM)	Petalogix Spartan 3aDSP1800 MMU ref design (MicroBlaze)
ARM RealView Emulation/Platform baseboard (ARM)	Luminary Micro LM3S811EVB (ARM Cortex-M3)
Spitz, Akita, Borzoi, Terrier and Tosa PDAs (PXA270 processor)	

ตัวอย่างการติดตั้งโปรแกรม QEMU ที่ถูกปรับแต่งโดย Linaro ซึ่งเน้นรองรับสถาปัตยกรรม ARM เป็นพิเศษ

```
$ sudo add-apt-repository ppa:linaro-maintainers/tools
$ sudo apt-get update
$ sudo apt-get install qemu-user-static qemu-system
$ qemu --version
QEMU PC emulator version 0.12.3 (qemu-kvm-0.12.3), Copyright (c) 2003-2008
Fabrice Bellard
```

ตัวอย่างชุดคำสั่ง QEMU สำหรับแต่ละสถาปัตยกรรม

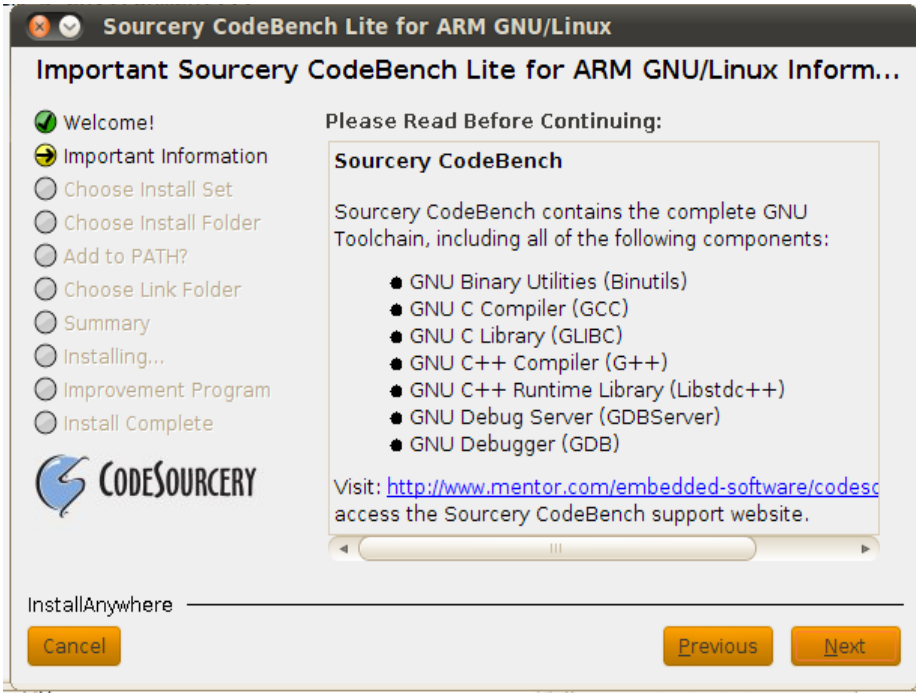
```
qemu-alpha          qemu-ppc            qemu-system-mips64el
qemu-arm            qemu-ppc64          qemu-system-mipsel
qemu-armeb          qemu-ppc64abi32     qemu-system-moxie
qemu-cris           qemu-s390x          qemu-system-or32
qemu-ga             qemu-sh4            qemu-system-ppc
qemu-i386           qemu-sh4eb          qemu-system-ppc64
qemu-img            qemu-sparc          qemu-system-ppcemb
```

```
qemu-io                 qemu-sparc32plus        qemu-system-s390x
qemu-m68k               qemu-sparc64            qemu-system-sh4
qemu-microblaze         qemu-system-alpha       qemu-system-sh4eb
qemu-microblazeel       qemu-system-arm         qemu-system-sparc
qemu-mips               qemu-system-cris        qemu-system-sparc64
qemu-mips64             qemu-system-i386        qemu-system-unicore32
qemu-mips64el           qemu-system-lm32        qemu-system-x86_64
qemu-mipsel             qemu-system-m68k        qemu-system-xtensa
qemu-mipsn32            qemu-system-microblaze  qemu-system-xtensaeb
qemu-mipsn32el          qemu-system-microblazeel qemu-unicore32
qemu-nbd                qemu-system-mips        qemu-x86_64
qemu-or32               qemu-system-mips64
```

ขั้นตอนการทดสอบการรันโปรแกรมภาษาซี บน QEMU

ทำการติดตั้ง Sourcery CodeBench ซึ่งเป็น cross-compiling toolchain ที่พร้อมสำหรับการพัฒนาโปรแกรมภาษา C/C++ บนระบบสมองกลฝังตัว โดยโหลดจากรุ่นล่าสุดจากลิงค์ของบริษัท Mentor Graphics

```
$ chmod +x arm-2013.05-24-arm-none-linux-gnueabi.bin
$ ./arm-2013.05-24-arm-none-linux-gnueabi.bin
Checking for required programs: awk grep sed bzip2 gunzip
Preparing to install...
Extracting the JRE from the installer archive...
Unpacking the JRE...
Extracting the installation resources from the installer archive...
Configuring the installer for this system's environment...
Launching installer...
```



ทำการสร้างไดเรกทอรีสำหรับทดสอบโปรแกรมโดยสร้างโปรแกรมชื่อ init.c เพื่อให้แสดงข้อความ “Hello World”

```
$ mkdir ~/qemu
$ cd ~/qemu
```

```
$ vim init.c
#include <stdio.h>
void main() {
  printf("Hello World!\n");
  while(1);
}
```

คอมไพล์โปรแกรมด้วย cross-compiler และวางเข้าไปในไฟล์ ramdisk

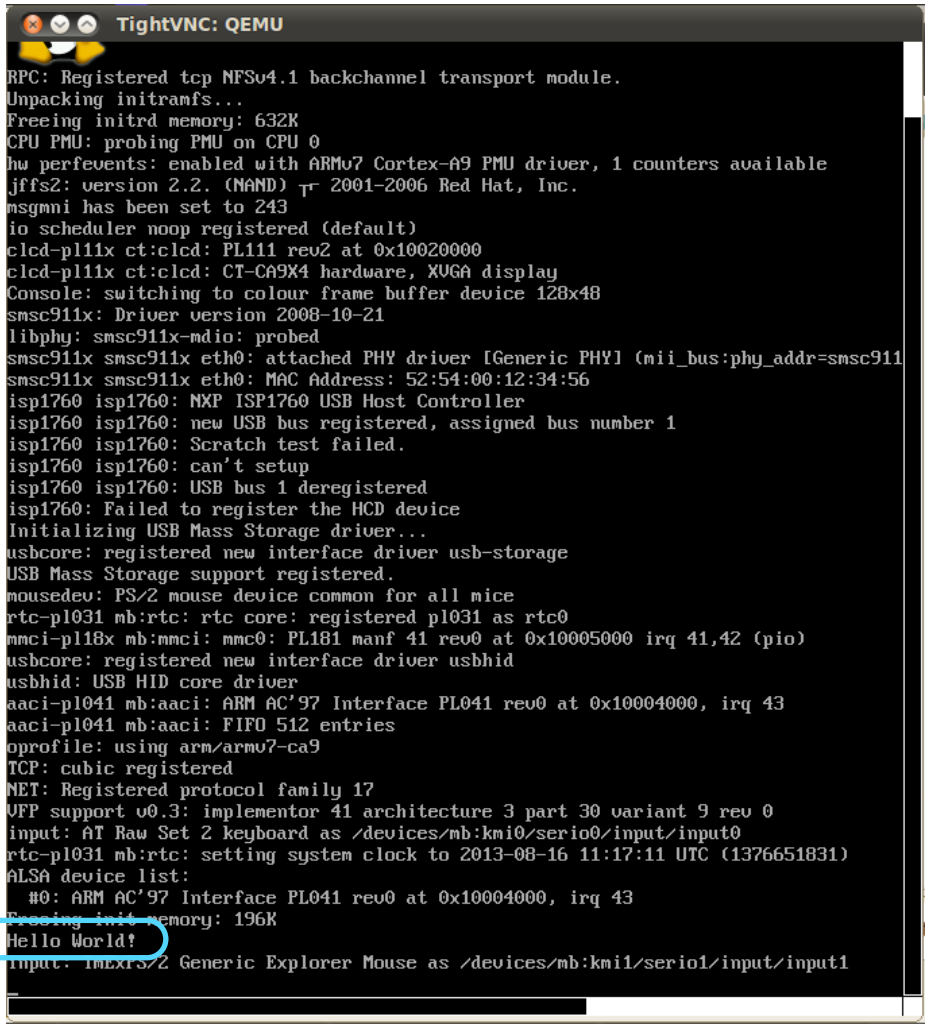
```
$ arm-none-linux-gnueabi-gcc -static init.c -o init
$ echo init|cpio -o --format=newc > initramfs
```

ดาวน์โหลดลีนุกซ์คอร์เนลเวอร์ชัน 3.9

```
$ wget https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.9.tar.xz
$ tar -xJvf linux-3.9.tar.xz
$ cd linux-3.9
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- vexpress_defconfig
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- all
$ cp arch/arm/boot/zImage ~/qemu
$ cd ~/qemu
$ ls -al
total 74488
drwxr-xr-x  3 student student     4096 2013-08-16 11:05 .
drwxr-xr-x 46 student student     4096 2013-08-16 11:16 ..
-rwxr-xr-x  1 student student   648380 2013-08-16 03:50 init
-rw-r--r--  1 student student       76 2013-08-16 03:20 init.c
-rw-r--r--  1 student student   648704 2013-08-16 03:54 initramfs
drwxr-xr-x 24 student student     4096 2013-08-16 04:10 linux-3.9
-rw-r--r--  1 student student 72104164 2013-04-28 17:40 linux-3.9.tar.xz
-rw-r--r--  1 student student      699 2013-08-16 09:33 README
-rwxr-xr-x  1 student student  2844736 2013-08-16 04:16 zImage
```

ทดสอบโปรแกรม init โดยการเรียกโปรแกรม qemu-system-arm ดังคำสั่งข้างล่าง

```
$ qemu-system-arm -M vexpress-a9 -kernel zImage -initrd initramfs
-serial stdio -append "console=tty1"
```

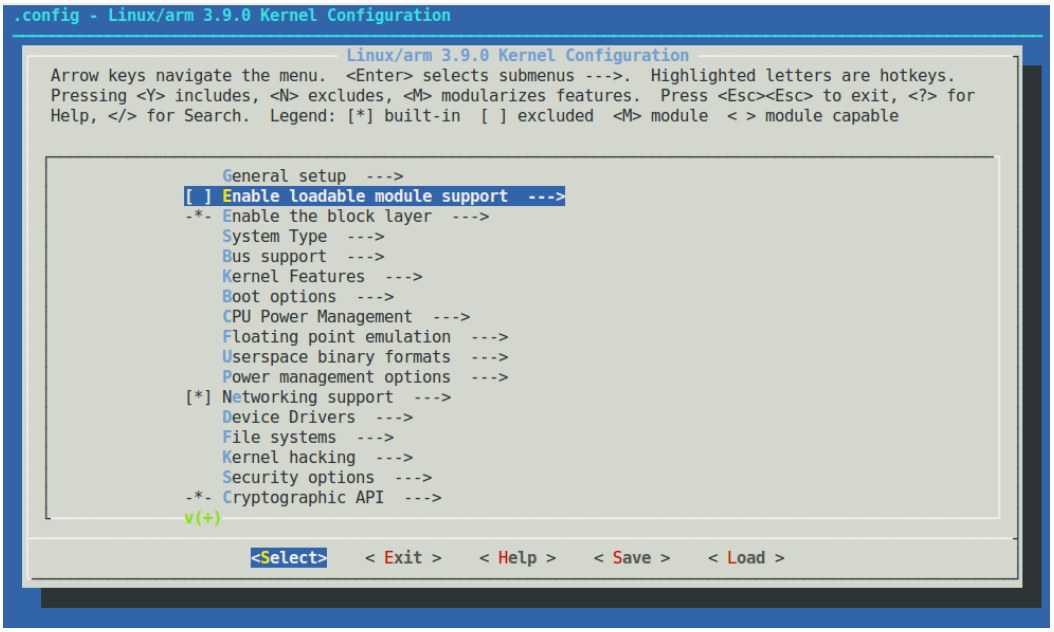


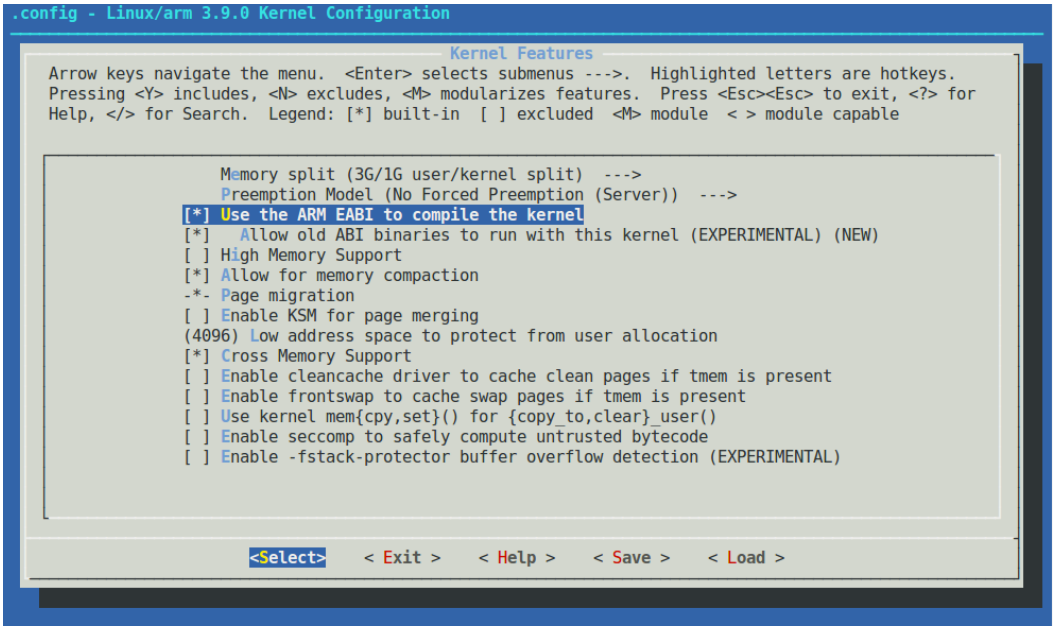
ขั้นตอนการทดสอบ BusyBox ภายใน root filesystem บน QEMU

ทำการคอมไพล์ลีนุกซ์คอร์เนลรุ่น 3.9 ที่ได้อธิบายข้างต้น เพื่อรองรับสถาปัตยกรรมของ ARM SoC ที่ชื่อว่า `versatile` ด้วยคำสั่งข้างล่างนี้

```
$ make mrproper
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- versatile_defconfig
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- menuconfig
```

ปรับแต่งค่าเคอร์เนล โดยการไม่เลือก “Enable loadable module support” feature แต่ให้เลือก “Use the ARM EABI to compile the kernel” แทน ที่อยู่ในเมนู Kernel Features





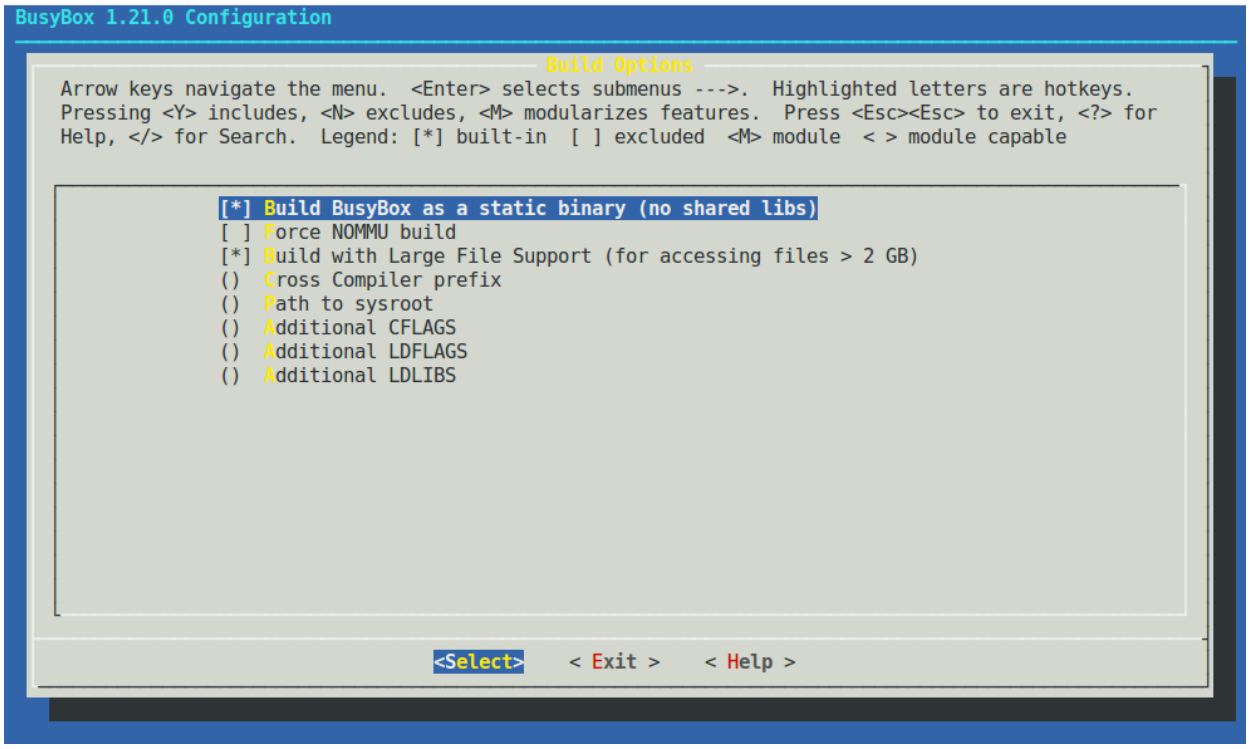
ทำการคอมไฟล์เคอร์เนลหลังจากตั้งค่าต่างๆเรียบร้อยแล้ว

```
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- all
```

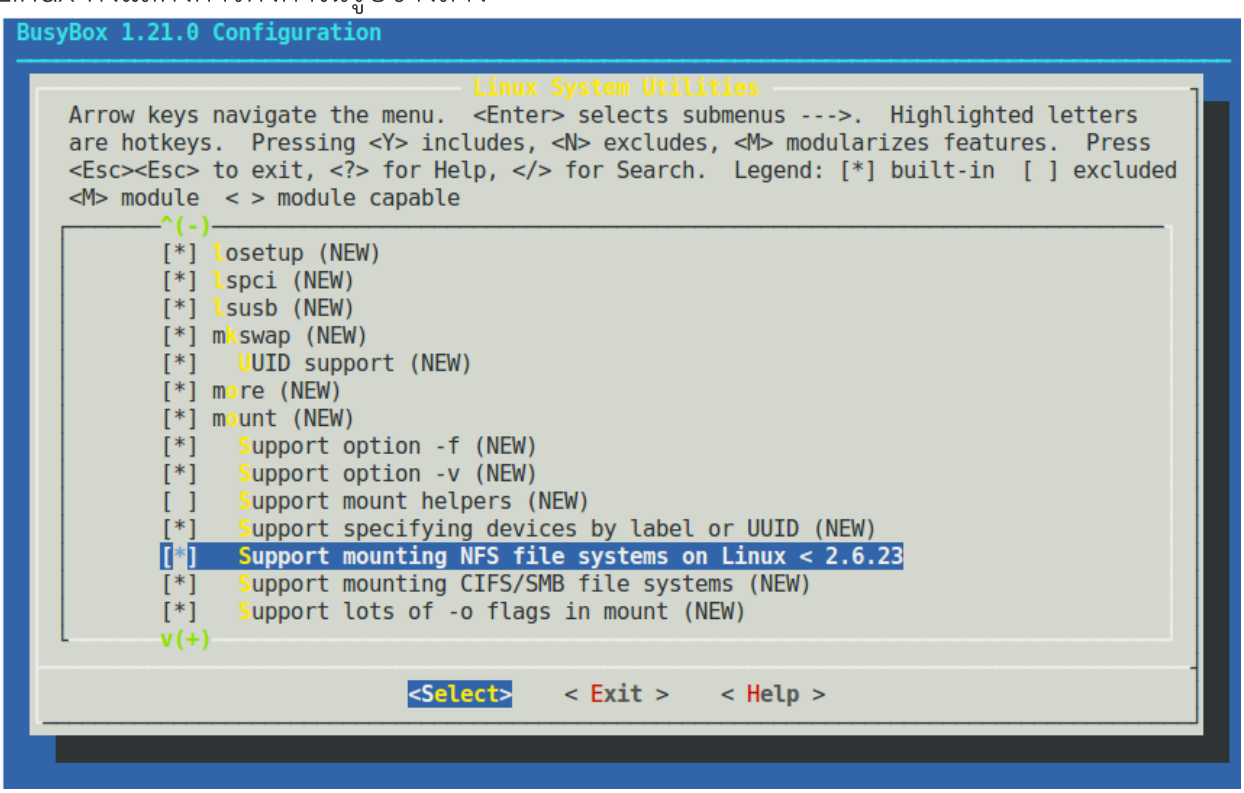
ดาวน์โหลดโปรแกรม BusyBox รุ่น 1.21.0 ลงในไดเรกทอรี ~/qemu ก่อนหน้านี้แล้วทำการตั้งค่าตามลำดับขั้นตอนดังนี้

```
$ wget http://busybox.net/downloads/busybox-1.21.0.tar.bz2
$ tar -xjvf busybox-1.21.0.tar.bz2
$ cd busybox-1.21.0
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- defconfig
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- menuconfig
```

ตั้งค่าให้สร้าง BusyBox เป็นแบบ static binary โดยการเข้าไปในเมนู Busybox Setting -> Build Options -> ทำการ check Build BusyBox as a static binary



ในกรณีที่ต้องการให้ระบบปฏิบัติการลีนุกซ์แบบฝังตัวสามารถเชื่อมต่อหรือ mount ไดเรกทอรีภายในพาร์ทิชั่นของ NFS (Network File System) ของเครื่องอื่นๆภายในระบบเครือข่ายได้จะต้องตั้งค่าใน BusyBox ซึ่งอยู่ภายในเมนู “Linux System Utilities” ชื่อว่า Support mounting NFS file systems on Linux ดังแสดงการตั้งค่าในรูปข้างล่าง



ทำการรันคำสั่ง install เพื่อติดตั้ง BusyBox ลงในไดเรกทอรี _install

```
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- install
...
./_install//usr/sbin/ubimkvol -> ../../bin/busybox
./_install//usr/sbin/ubirmvol -> ../../bin/busybox
./_install//usr/sbin/ubirsvol -> ../../bin/busybox
./_install//usr/sbin/ubiupdatevol -> ../../bin/busybox
./_install//usr/sbin/udhcpd -> ../../bin/busybox
--------------------------------------------------
You will probably need to make your busybox binary
setuid root to ensure all configured applets will
work properly.
--------------------------------------------------
```

สร้างไดเรกทอรีจำเป็นพื้นฐาน สำหรับ root filesystem

```
$ cd _install
$ mkdir proc sys dev etc etc/init.d
$ ls -al
total 36
drwxr-xr-x  9 student student 4096 2013-08-16 12:14 .
drwxr-xr-x 34 student student 4096 2013-08-16 11:58 ..
drwxr-xr-x  2 student student 4096 2013-08-16 11:58 bin
drwxr-xr-x  2 student student 4096 2013-08-16 12:14 dev
drwxr-xr-x  3 student student 4096 2013-08-16 12:14 etc
lrwxrwxrwx  1 student student   11 2013-08-16 11:58 linuxrc -> bin/busybox
drwxr-xr-x  2 student student 4096 2013-08-16 12:14 proc
drwxr-xr-x  2 student student 4096 2013-08-16 11:58 sbin
drwxr-xr-x  2 student student 4096 2013-08-16 12:14 sys
drwxr-xr-x  4 student student 4096 2013-08-16 11:58 usr
```

สร้างไฟล์ rcS เข้าไปไดเรกทอรี etc/init.d/ เพื่อทำการ mount ไดเรกทอรี /proc และ /sysfs ซึ่งเป็น virtual filesystem

```
$ vim etc/init.d/rcS
```

เพิ่ม shell script เข้าไปในไฟล์ rcS ดังนี้

```
#!/bin/sh
mount -t proc none /proc
mount -t sysfs none /sys
/sbin/mdev -s
```

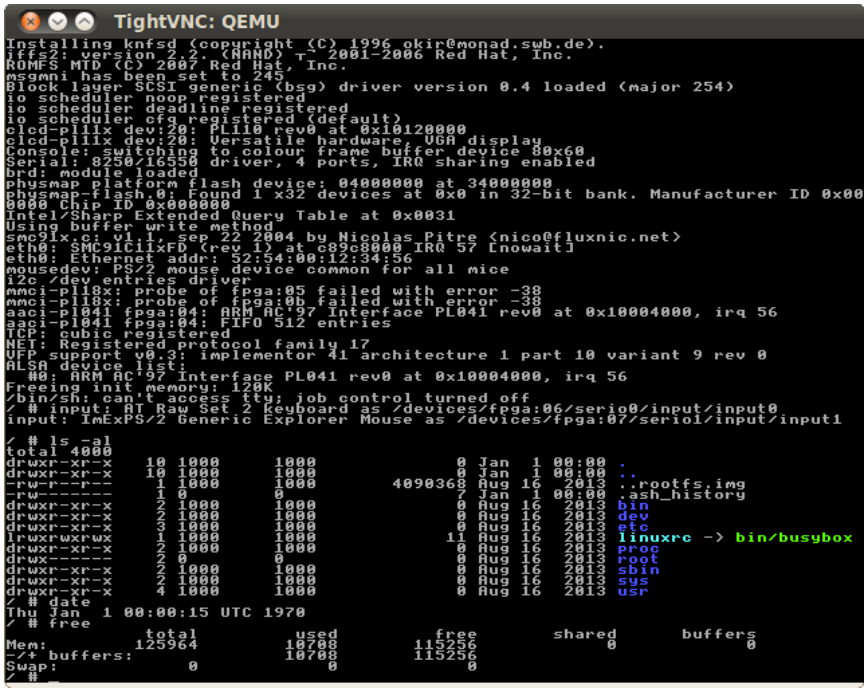
แปลงสิทธิ์ไฟล์ rcS ให้สามารถรันได้ และเริ่มดำเนินการสร้าง root filesystem ชื่อ rootfs.img จากไดเรกทอรี _install

```
$ chmod +x etc/init.d/rcS
$ find . | cpio -o --format=newc > ../rootfs.img
11984 blocks
```

```
$ cd ..
$ gzip -c rootfs.img > rootfs.img.gz
$ cp rootfs.img.gz ~/qemu
```

เรียกโปรแกรม qemu-system-arm เพื่อทดสอบคำสั่งพื้นฐานของ BusyBox

```
$ cd ~/qemu
$ qemu-system-arm -M versatilepb -m 128M -kernel zImage -initrd
rootfs.img.gz -append "root=/dev/ram rdinit=/bin/sh"
```



การสร้างระบบจำลองสำหรับสถาปัตยกรรม ARM ด้วยชุดเครื่องมือ BUILDROOT บนโปรแกรม QEMU

ทำการติดตั้งโปรแกรมและไลบรารีที่เกี่ยวข้อง

```
$ sudo apt-get install autoconf automake libtool libexpat1-dev
libncurses5-dev bison flex patch curl cvs texinfo build-essential sub-
version gawk python-dev gperf ncurses-dev g++ gcc texi2html nfs-
kernel-server
```

สร้างไดเรกทอรีสำหรับเก็บ root filesystem ของระบบปฏิบัติการ Embedded Linux ด้วยโปรโตคอลประยุกต์ NFS (Network File System) ผ่านระบบเครือข่าย LAN

```
$ mkdir ~/nfsroot
```

แล้วทำการตั้งค่าไดเรกทอรีที่ต้องการแชร์ผ่าน NFS ภายในไฟล์ /etc/exports ด้วยคำสั่งข้างล่างนี้

```
$ sudo vim /etc/exports
/home/student/nfsroot *(rw,sync,no_root_squash,no_subtree_check)
```

เริ่มเรียกบริการ NFS Server ใหม่อีกครั้ง

```
$ sudo /etc/init.d/nfs-kernel-server restart
```

ติดตั้งเครื่องมือ buildroot ซึ่งเป็น cross-compiling toolchain

```
$ wget http://buildroot.uclibc.org/downloads/buildroot-2013.05.tar.gz
$ tar -xzvf buildroot-2013.05.tar.gz
$ cd buildroot-2013.05/
```

ตั้งค่าระบบปฏิบัติการลีนุกซ์แบบฝังตัวเพื่อให้รองรับรายละเอียดของสถาปัตยกรรมและรายละเอียดอื่นๆ ดังนี้

```
$ make menuconfig
```

โดยให้ตั้งค่าดังนี้:

Target Architecture ==> ARM (little endian)

Target Architecture Variant ==> arm926t

Target ABI ==> EABI

System Configuration ==> serial port to run getty on (ttyAMA0)

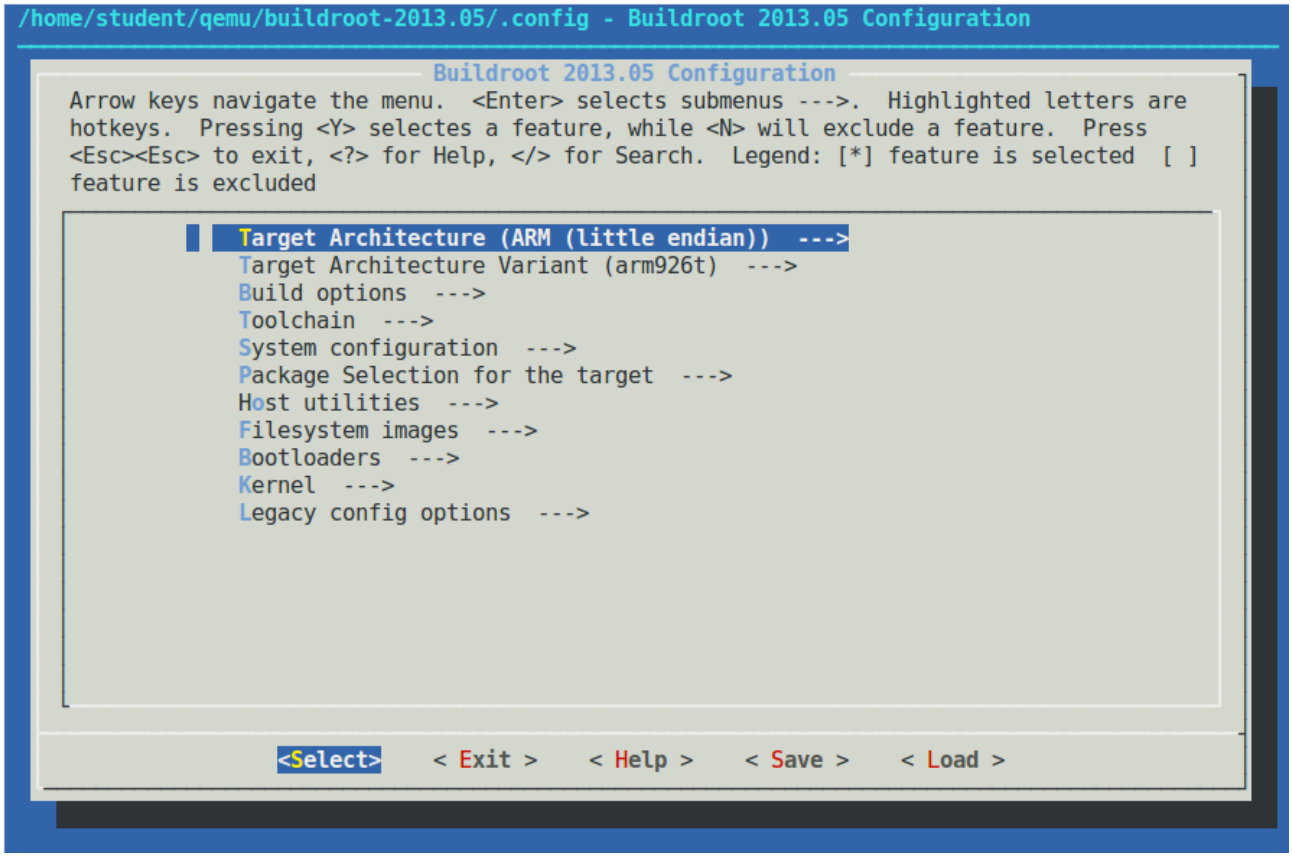
Package Selection ==> Busybox

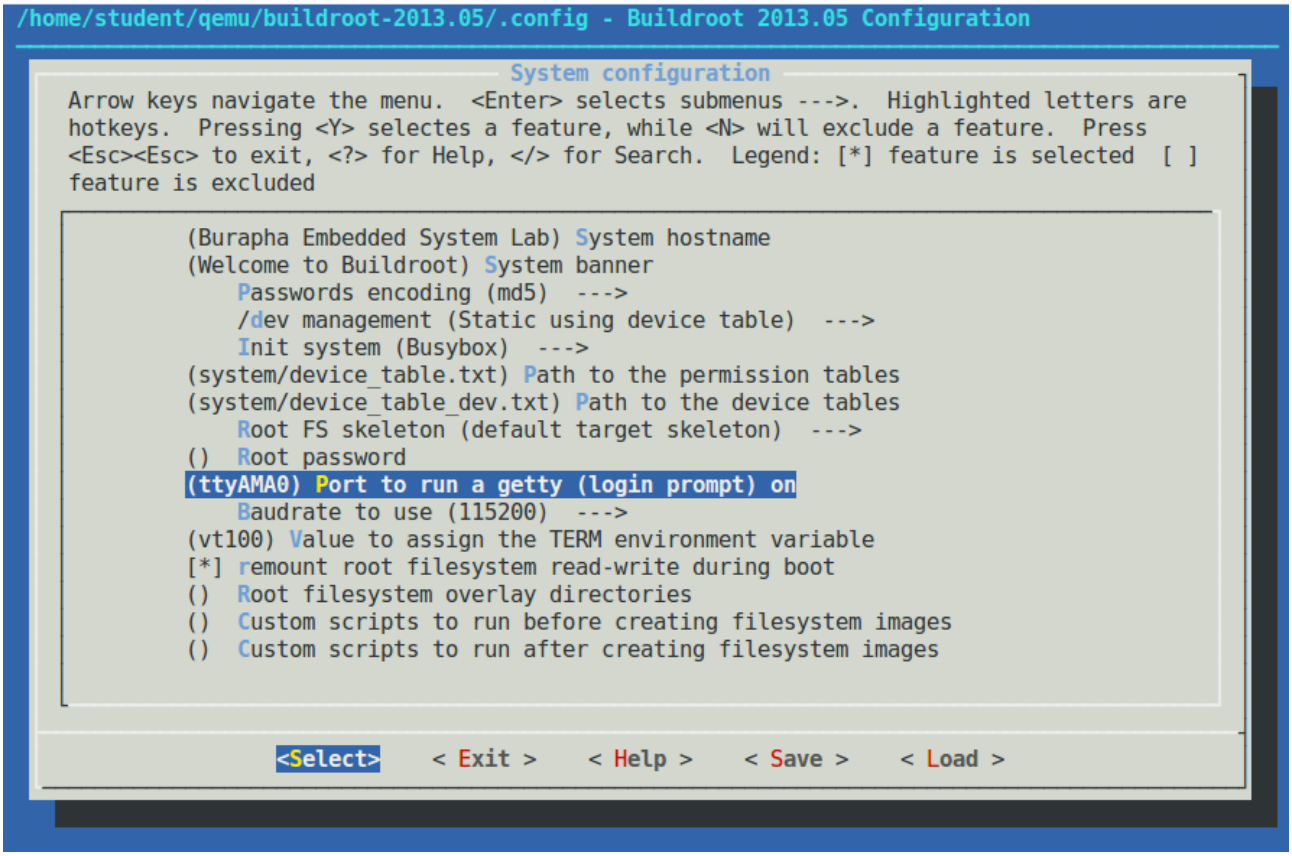
FileSystem Images ==> cpio the root filesystem และ tar the root filesystem

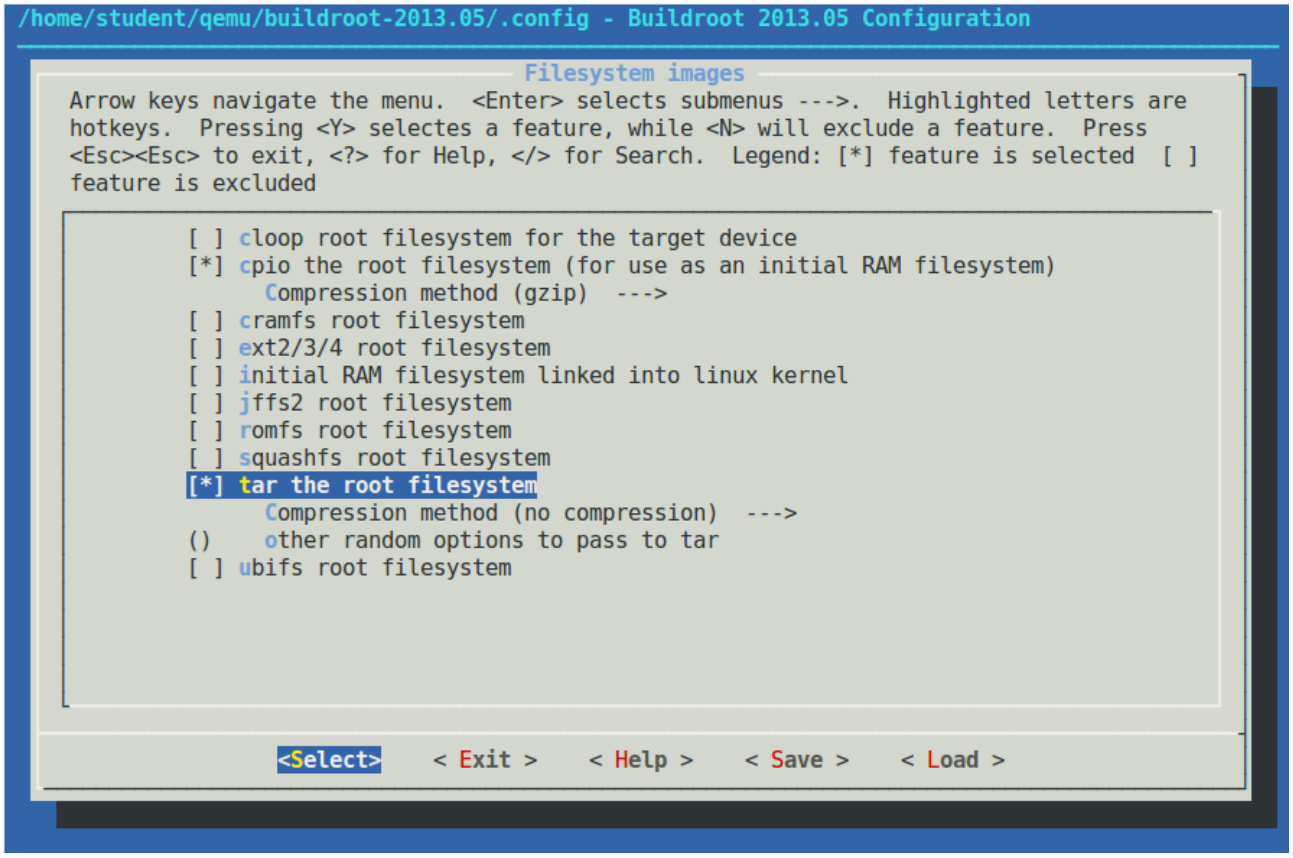
---

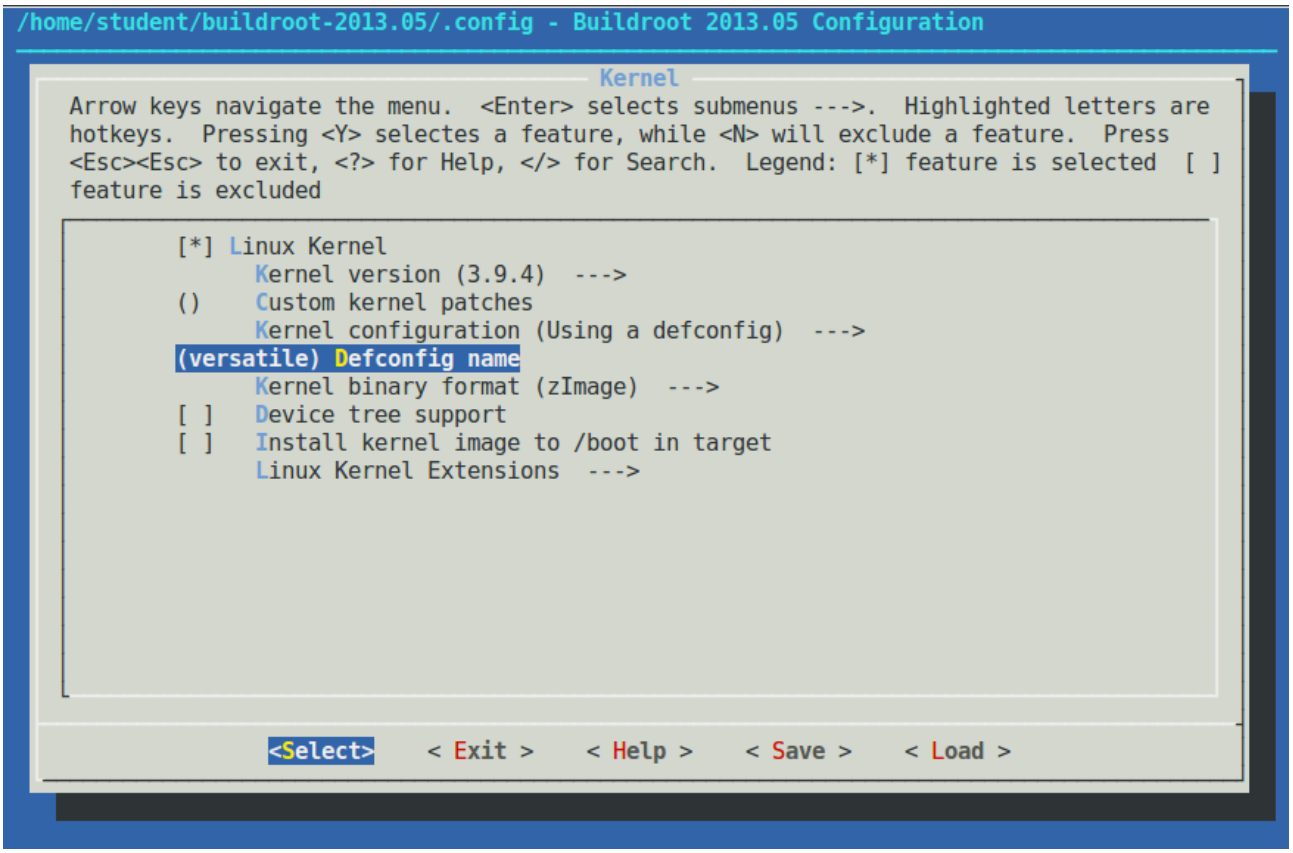
Kernel ==>ระบุชื่อ defconfig เป็น “versatile”

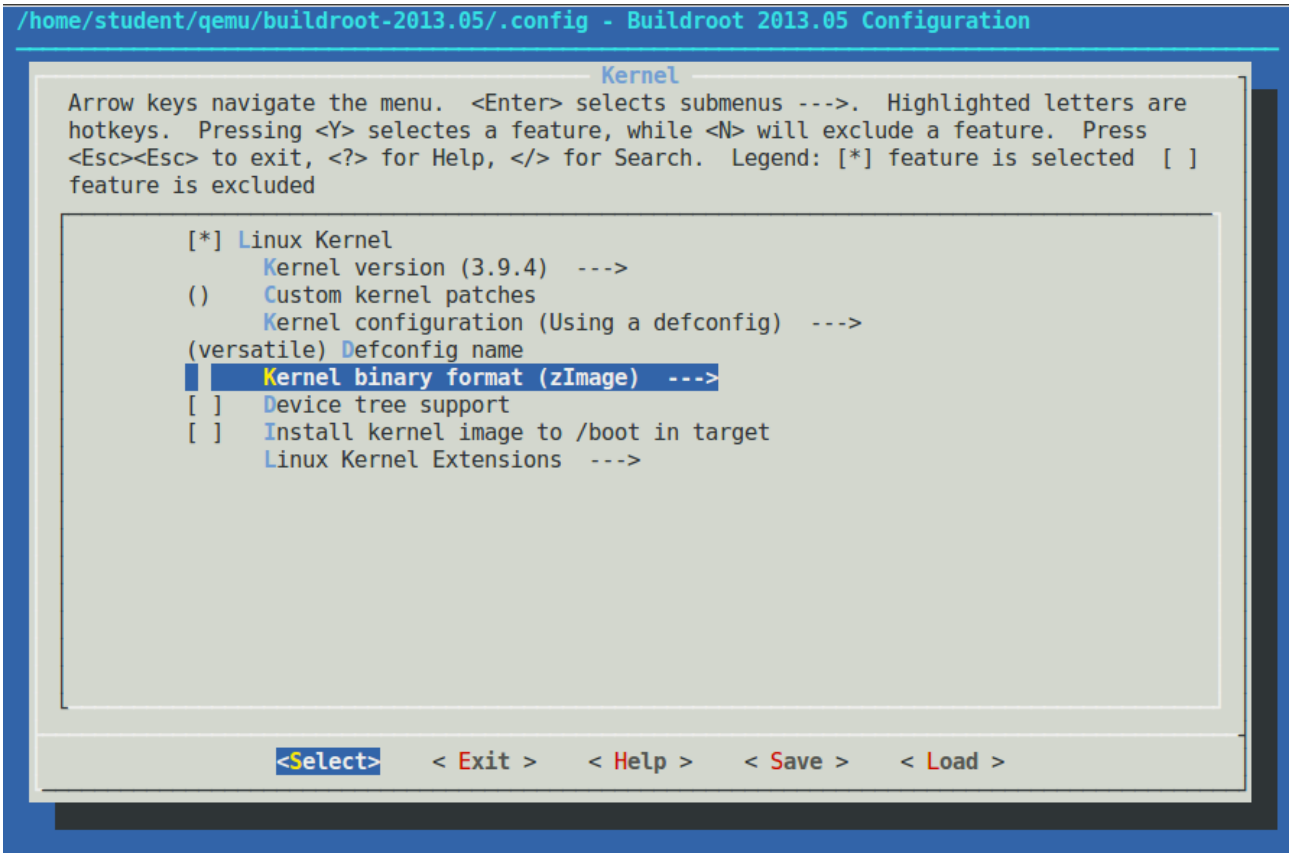
Kernel ==> Kernel binary format to zImage


/home/student/qemu/buildroot-2013.05/.config - Buildroot 2013.05 Configuration
Buildroot 2013.05 Configuration
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted letters are hotkeys. Pressing <Y> selectes a feature, while <N> will exclude a feature. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] feature is selected [ ] feature is excluded
Target Architecture (ARM (little endian)) --->
Target Architecture Variant (arm926t) --->
Build options --->
Toolchain --->
System configuration --->
Package Selection for the target --->
Host utilities --->
Filesystem images --->
Bootloaders --->
Kernel --->
Legacy config options --->
<Select> < Exit > < Help > < Save > < Load >


/home/student/qemu/buildroot-2013.05/.config - Buildroot 2013.05 Configuration
System configuration
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted letters are hotkeys. Pressing <Y> selectes a feature, while <N> will exclude a feature. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] feature is selected [ ] feature is excluded
(Burapha Embedded System Lab) System hostname
(Welcome to Buildroot) System banner
Passwords encoding (md5) --->
/dev management (Static using device table) --->
Init system (Busybox) --->
(system/device_table.txt) Path to the permission tables
(system/device_table_dev.txt) Path to the device tables
Root FS skeleton (default target skeleton) --->
() Root password
(ttyAMA0) Port to run a getty (login prompt) on
Baudrate to use (115200) --->
(vt100) Value to assign the TERM environment variable
[*] remount root filesystem read-write during boot
() Root filesystem overlay directories
() Custom scripts to run before creating filesystem images
() Custom scripts to run after creating filesystem images
<Select> < Exit > < Help > < Save > < Load >


/home/student/qemu/buildroot-2013.05/.config - Buildroot 2013.05 Configuration
Filesystem images
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted letters are hotkeys. Pressing <Y> selectes a feature, while <N> will exclude a feature. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] feature is selected [ ] feature is excluded
[ ] cloop root filesystem for the target device
[*] cpio the root filesystem (for use as an initial RAM filesystem)
Compression method (gzip) --->
[ ] cramfs root filesystem
[ ] ext2/3/4 root filesystem
[ ] initial RAM filesystem linked into linux kernel
[ ] jffs2 root filesystem
[ ] romfs root filesystem
[ ] squashfs root filesystem
[*] tar the root filesystem
Compression method (no compression) --->
() other random options to pass to tar
[ ] ubifs root filesystem
<Select> < Exit > < Help > < Save > < Load >


/home/student/buildroot-2013.05/.config - Buildroot 2013.05 Configuration
Kernel
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted letters are hotkeys. Pressing <Y> selectes a feature, while <N> will exclude a feature. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] feature is selected [ ] feature is excluded
[*] Linux Kernel
Kernel version (3.9.4) --->
() Custom kernel patches
Kernel configuration (Using a defconfig) --->
(versatile) Defconfig name
Kernel binary format (zImage) --->
[ ] Device tree support
[ ] Install kernel image to /boot in target
Linux Kernel Extensions --->
<Select> < Exit > < Help > < Save > < Load >




```
$ make
$ ls -al output/images/
total 6712
drwxr-xr-x 2 student student    4096 2013-07-22 11:45 .
drwxr-xr-x 8 student student    4096 2013-07-22 21:03 ..
-rw-r--r-- 1 student student 1922560 2013-07-22 21:03 rootfs.cpio
-rw-r--r-- 1 student student  858843 2013-07-22 21:03 rootfs.cpio.gz
-rw-r--r-- 1 student student 2181120 2013-07-22 21:03 rootfs.tar
-rw-r--r-- 1 student student 1895000 2013-07-22 11:44 uImage

$ cp output/images/uImage ~/qemu/
```

วาง root filesystem ไปยัง ไดเรกทอรี NFS ชื่อว่า ~/nfsroot ที่เครื่อง Host ได้เปิดแชร์ไว้

```
$ sudo tar -xvf rootfs.tar ~/nfsroot/
$ sudo chown -R student:student ~/nfsroot
```

ขั้นตอนการสร้าง tun/tap device เพื่อกำหนดวงของระบบเครือข่ายท้องถิ่นภายในระหว่าง QEMU และเครื่อง Host เพื่อให้ระบบปฏิบัติการลีนุกซ์แบบฝังตัวภายใน QEMU สามารถ mount ไดเรกทอรี NFS ที่เปิดแชร์ไว้บนเครื่อง Host ได้ โดยตั้งค่า IP Address ให้กับเครื่อง Host เป็น 192.168.1.1 และตั้งค่า IP Address สำหรับอุปกรณ์ target ซึ่งในที่นี้ก็คือ 192.168.1.101
ขั้นตอนต่อไปทำการติดตั้งโปรแกรม uml-utilities (ที่มีคำสั่ง tunctl) เพื่อให้สามารถจำลองสัญญาณระบบเครือข่าย (network traffic)

```
$ sudo apt-get install uml-utilities
$ sudo tunctl -u $(whoami) -t tap1
$ sudo ifconfig tap1 192.168.1.1
$ sudo route add -net 192.168.1.0 netmask 255.255.255.0 dev tap1
$ sudo sh -c "echo  1 > /proc/sys/net/ipv4/ip_forward"
```

ขั้นตอนสุดท้ายทำการรันโปรแกรม QEMU เพื่อจำลองการทำงานบอร์ดสมองกลฝังตัว (Versatile Board) และทำการโหลด root filesystem จากเครื่อง Host

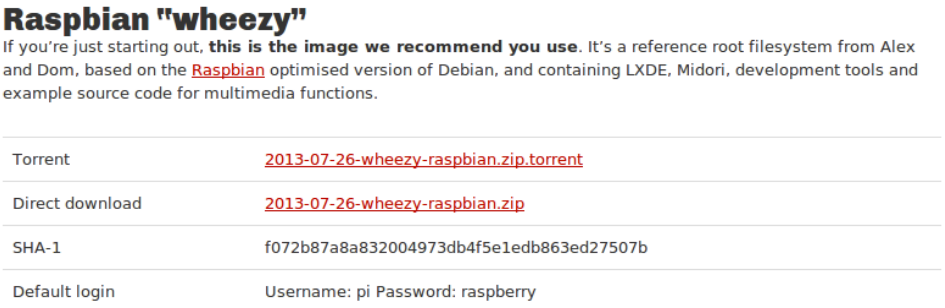
```
$ qemu-system-arm -M versatilepb -m 128M -kernel ~/uImage -append
"console=ttyAMA0 root=/dev/nfs rw nfsroot=192.168.1.1:~/nfsroot
ip=192.168.1.101" -net nic -net tap,ifname=tap1,script=no -nographic
Set 'tap1' persistent and owned by uid 1000
SIOCADDRT: File exists
Uncompressing Linux... done, booting the kernel.
Booting Linux on physical CPU 0x0
Linux version 3.9.4 (student@marabuntu) (gcc version 4.7.3 (Buildroot 2013.05) ) #1
Mon Jul 22 11:44:32 PDT 2013
CPU: ARM926EJ-S [41069265] revision 5 (ARMv5TEJ), cr=00093177
CPU: VIVT data cache, VIVT instruction cache
Machine: ARM-Versatile PB
Memory policy: ECC disabled, Data cache writeback
sched_clock: 32 bits at 24MHz, resolution 41ns, wraps every 178956ms
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 32512
Kernel command line: console=ttyAMA0 root=/dev/nfs rw
nfsroot=192.168.1.1:/home/student/nfsroot ip=192.168.1.101
PID hash table entries: 512 (order: -1, 2048 bytes)
Dentry cache hash table entries: 16384 (order: 4, 65536 bytes)
Inode-cache hash table entries: 8192 (order: 3, 32768 bytes)
__ex_table already sorted, skipping sort
Memory: 128MB = 128MB total
Memory: 126192k/126192k available, 4880k reserved, 0K highmem
Virtual kernel memory layout:
    vector  : 0xffff0000 - 0xffff1000   (   4 kB)
    fixmap  : 0xfff00000 - 0xfffe0000   ( 896 kB)
    vmalloc : 0xc8800000 - 0xff000000   ( 872 MB)
    lowmem  : 0xc0000000 - 0xc8000000   ( 128 MB)
    modules : 0xbf000000 - 0xc0000000   (  16 MB)
      .text : 0xc0008000 - 0xc0342850   (3307 kB)
      .init : 0xc0343000 - 0xc035f87c   ( 115 kB)
      .data : 0xc0360000 - 0xc0385440   ( 150 kB)
       .bss : 0xc0385440 - 0xc039f78c   ( 105 kB)
NR_IRQS:224
VIC @f1140000: id 0x00041190, vendor 0x41
FPGA IRQ chip 0 "SIC" @ f1003000, 13 irqs
Console: colour dummy device 80x30
Calibrating delay loop... 300.44 BogoMIPS (lpj=1502208)
pid_max: default: 32768 minimum: 301
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok
Setting up static identity map for 0xc027e748 - 0xc027e7a0
NET: Registered protocol family 16
DMA: preallocated 256 KiB pool for atomic coherent allocations
Serial: AMBA PL011 UART driver
dev:f1: ttyAMA0 at MMIO 0x101f1000 (irq = 44) is a PL011 rev1
console [ttyAMA0] enabled
dev:f2: ttyAMA1 at MMIO 0x101f2000 (irq = 45) is a PL011 rev1
```

```
dev:f3: ttyAMA2 at MMIO 0x101f3000 (irq = 46) is a PL011 rev1
...
Console: switching to colour frame buffer device 80x60
brd: module loaded
physmap platform flash device: 04000000 at 34000000
physmap-flash physmap-flash.0: map_probe failed
smc91x.c: v1.1, sep 22 2004 by Nicolas Pitre <nico@fluxnic.net>
eth0: SMC91C11xFD (rev 1) at c8800000 IRQ 57 [nowait]
eth0: Ethernet addr: 52:54:00:12:34:56
mousedev: PS/2 mouse device common for all mice
TCP: cubic registered
NET: Registered protocol family 17
VFP support v0.3: implementor 41 architecture 1 part 10 variant 9 rev 0
eth0: link up
IP-Config: Guessing netmask 255.255.255.0
IP-Config: Complete:
     device=eth0, hwaddr=52:54:00:12:34:56, ipaddr=192.168.1.101,
mask=255.255.255.0, gw=255.255.255.255
     host=192.168.1.101, domain=, nis-domain=(none)
     bootserver=255.255.255.255, rootserver=192.168.1.1, rootpath=
input: AT Raw Set 2 keyboard as /devices/fpga:06/serio0/input/input0
input: ImExPS/2 Generic Explorer Mouse as /devices/fpga:07/serio1/input/input1
VFS: Mounted root (nfs filesystem) on device 0:9.
Freeing init memory: 112K
Starting logging: OK
Initializing random number generator... done.
Starting network...
ip: RTNETLINK answers: File exists

Welcome to Burapha Embedded System Lab
Burapha login:
# ls
# ls -al
total 20
drwxr-xr-x    2 default  default       4096 Jul 23  2013 .
drwxr-xr-x   17 default  default       4096 Jul 23  2013 ..
-rw-------    1 root     root            10 Jul 23  2013 .ash_history
-rw-r--r--    1 default  default           0 May 31  2013 .bash_history
-rw-r--r--    1 default  default         175 May 31  2013 .bash_logout
-rw-r--r--    1 default  default         161 May 31  2013 .bash_profile
#
```

ตัวอย่างการสร้างระบบจำลองเสมือนของบอร์ด Raspberry Pi บน QEMU

ดาวน์โหลดไฟล์ image ของบอร์ด Raspberry PI จาก http://www.raspberrypi.org/downloads และให้ทำการเปลี่ยนชื่อไฟล์เป็น `wheezy-raspbian.img` ดังขั้นตอนข้างล่างนี้



Raspbian "wheezy"

If you're just starting out, **this is the image we recommend you use**. It's a reference root filesystem from Alex and Dom, based on the Raspbian optimised version of Debian, and containing LXDE, Midori, development tools and example source code for multimedia functions.

Torrent	2013-07-26-wheezy-raspbian.zip.torrent
Direct download	2013-07-26-wheezy-raspbian.zip
SHA-1	f072b87a8a832004973db4f5e1edb863ed27507b
Default login	Username: pi Password: raspberry

```
$ unzip 2013-07-26-wheezy-raspbian.zip
$ mv 2013-07-26-wheezy-raspbian.img ~/qemu/wheezy-raspbian.img
```

ใช้คำสั่ง file เพื่อดูรายละเอียดพาร์ทิชั่นภายในไฟล์ `wheezy-raspbian.img`

```
$ file wheezy-raspbian.img
wheezy-rasbian.img: x86 boot sector; partition 1: ID=0xc, starthead 130,
startsector 8192, 114688 sectors; partition 2: ID=0x83, starthead 165,
startsector 122880, 3665920 sectors, code offset 0xb8
```

นำค่า `startsector` (122880) จาก `partition 2` มาคูณกับค่า 512 จะได้ค่า `offset` เท่ากับ 62914560 แล้วให้นำค่า `offset` การใช้ในการ mount ไฟล์ชื่อ `wheezy-raspbian.img` เพื่อให้ชี้ไปยังไดเรกทอรี `/mnt` ดังคำสั่งข้างล่างนี้

```
$ sudo mount wheezy-raspbian.img -o offset=62914560 /mnt
```

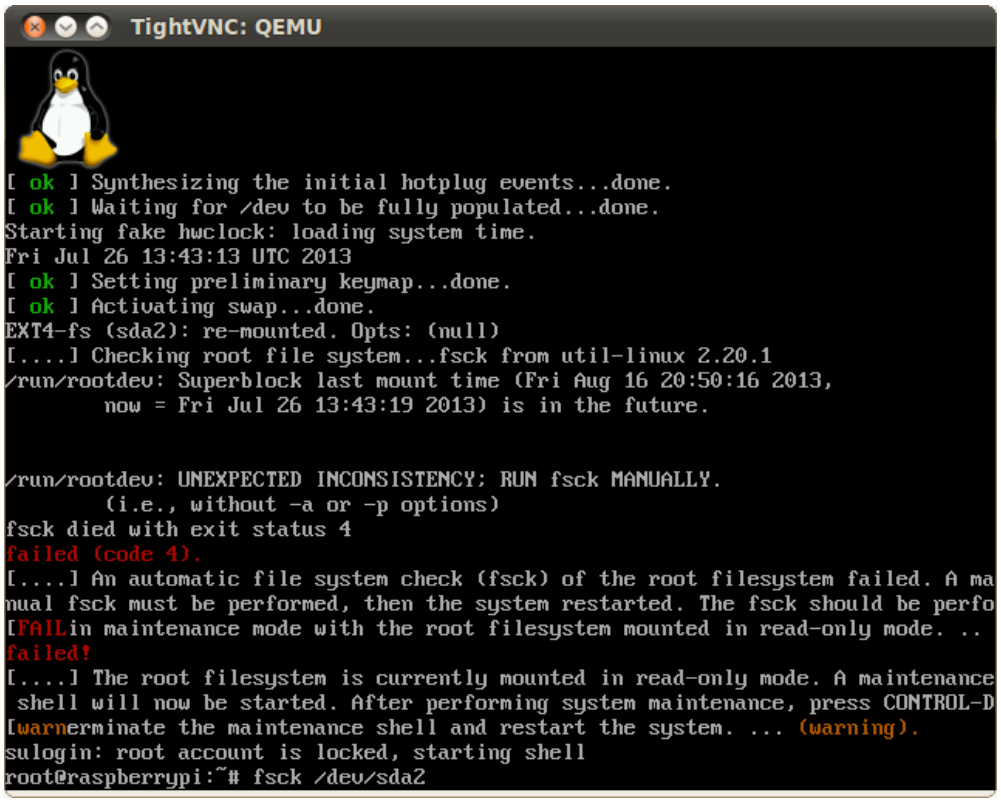
แก้ไขไฟล์ `/mnt/etc/ld.so.preload` แล้วปิดบรรทัดคำสั่งด้วยเครื่องหมาย # เพื่อไม่ให้โหลดไลบรารี ดังข้างล่างนี้

```
...
# /usr/lib/arm-linux-gnueabihf/libcofi_rpi.so
...
```

```
$ sudo umount /mnt
```

ก่อนรันโปรแกรม QEMU จำเป็นต้องดาวน์โหลดลีนุกซ์คอร์เนลเฉพาะสำหรับบอร์ด Raspberry Pi

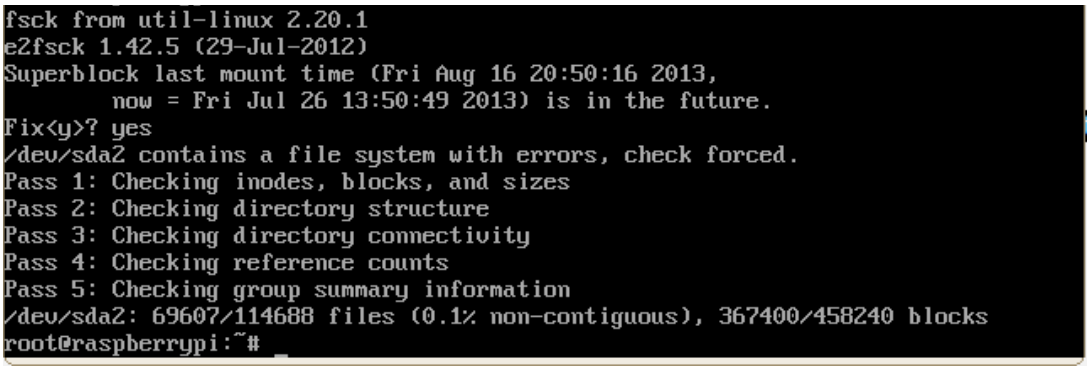
```
$ cd ~/qemu/
$ wget http://xecdesign.com/downloads/linux-qemu/kernel-qemu
$ qemu-system-arm -kernel ~/qemu/kernel-qemu -cpu arm1176 -m 256 -M
versatilepb -no-reboot -serial stdio -append "root=/dev/sda2 panic=1"
-hda ~/qemu/wheezy-raspbian.img
```



จากหน้าต่างข้างบนจะสังเกตเห็นว่ามีการแสดงข้อความล้มเหลวจากการอ่านระบบไฟล์ (filesystem) เนื่องจากไม่ใช่สิทธิ์ root ดังนั้นจะต้องทำการซ่อมระบบไฟล์ด้วยคำสั่งในฐานะ root อีกครั้ง

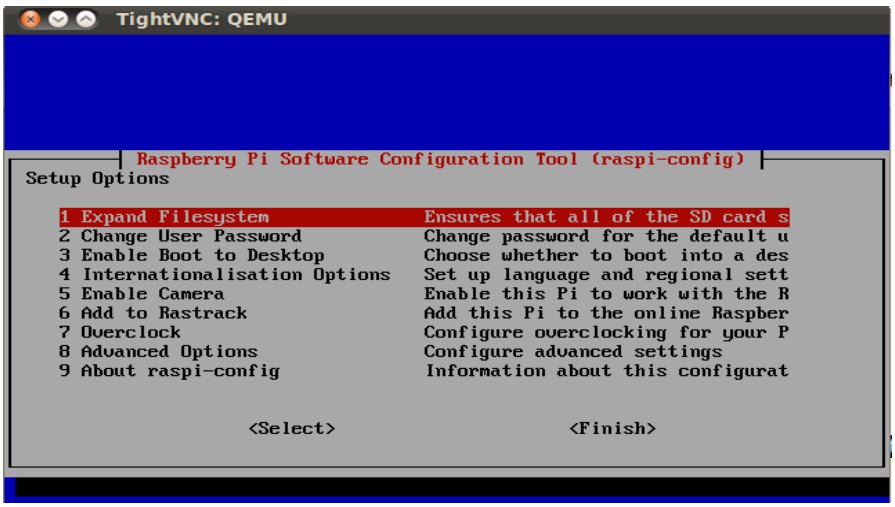
```
$ sudo fsck /dev/sda2
```

คำสั่ง fsck ก็จะทำการตรวจสอบและซ่อมระบบไฟล์ดังแสดงในรูปข้างล่าง

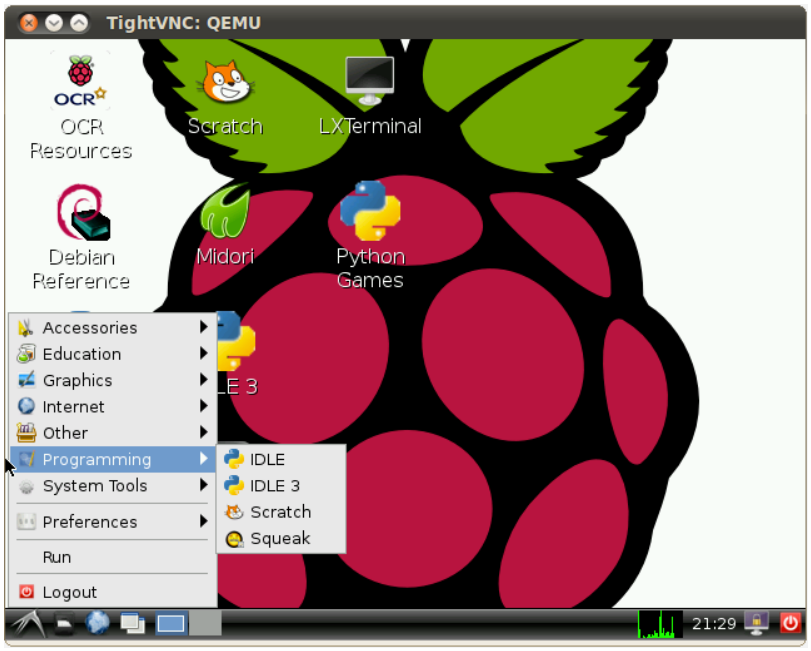


ในกรณีต้องการตั้งค่าสำหรับบอร์ด Raspberry Pi โดยเฉพาะ สามารถรันคำสั่งดังนี้

```
root@raspberrypi# raspi-config
```



จากเมนูด้านบนสามารถตั้งค่าอื่นตามที่ต้องการได้เช่น ต้องการให้บูทเข้าสู่ GUI Mode ดังนั้นให้เลือกข้อที่ 3 (Enable Boot to Desktop) และเริ่มบูทเข้าระบบใหม่อีกครั้งก็จะเข้าสู่ GUI Mode ดังแสดงข้างล่าง



รูปที่ 3.29 แสดงการเข้าสู่ระบบปฏิบัติการภายใน Raspberry Pi บน QEMU

ตัวอย่างการสร้างระบบจำลองเสมือนของบอร์ด Friendly ARM (Mini2440) บน QEMU

บอร์ด FriendlyARM Mini2440 หรือนิยมเรียกสั้นๆว่าบอร์ด Mini2440 ได้ถูกออกแบบในลักษณะเป็นบอร์ดชนิด SBC (Single-Board Computer) ที่ใช้หน่วยประมวลผลกลาง ARM9 ซึ่งเป็น System-on-Chip รุ่น Samsung S3C2440



รูปที่ 3.30 บอร์ด Friendly ARM Mini2440

นักพัฒนาสามารถเรียนรู้การทำงานของระบบปฏิบัติการภายในบอร์ด Mini2440 โดยเล่นอยู่บนโปรแกรม QEMU ที่จำลองการทำงานของบอร์ด โดยทำการติดตั้งโปรแกรมและแพ็กเก็ตที่เกี่ยวข้องดังต่อไปนี้ *(อ้างอิงมาจาก http://project4fun.com/node/33)*

```
$ sudo apt-get -y update
$ sudo apt-get -y install build-essential
$ sudo apt-get -y install git-core
$ sudo apt-get -y install zlib1g-dev
$ sudo apt-get -y install libsdl-console-dev
$ sudo apt-get -y install ia32-libs
$ sudo apt-get -y install bridge-utils
$ sudo apt-get -y install uml-utilities
$ sudo apt-get -y install qemu-utils
$ sudo apt-get -y install nbd-client
$ sudo apt-get -y install kpartx
```

ติดตั้ง cross-compiling toolchains

```
# cd /opt
# wget
http://dl-12.one2up.com/onetwo/content/2012/12/24/614ca76f9a53c060bd8cfa7effd541f6.tgz
# mv 614ca76f9a53c060bd8cfa7effd541f6.tgz arm920t-eabi.tgz
```

```
# tar xzvf arm920t-eabi.tgz -C /
# export PATH=\$PATH:/opt/toolchains/arm920t-eabi/bin
# export CROSS_COMPILE=arm-angstrom-linux-gnueabi-
```

คัดลอกข้อมูลจาก git://repo.or.cz/qemu/mini2440.git ดังนี้

```
$ mkdir ~/eeburapha/qemu
$ cd ~/eeburapha/qemu
$ git clone git://repo.or.cz/qemu/mini2440.git
...
Receiving objects: 100% (xxxxx/xxxxx), xx.xx MiB | xxxx KiB/s, done.
Resolving deltas: 100% (xxxxx/xxxxx), done.
```

คอมไพล์ไบนารี qemu ที่ซัพพอร์ต Mini2440

```
$ cd mini2440
$ wget
http://dl-10.one2up.com/onetwo/content/2013/9/10/fde6b403a54066a6aa138
38f7588b91e.patch
$ mv fde6b403a54066a6aa13838f7588b91e.patch bug_SIGFPE.patch
$ patch –p0 < bug_SIGFPE.patch
$ ./configure --target-list=arm-softmmu
$ make
```

แก้ไขไฟล์ mini2440_start.sh ให้มีรายละเอียดดังข้างล่างนี้

```
$ vim ~/eeburapha/qemu/mini2440/mini2440/mini2440_start.sh
echo Starting in $base
name_nand="$base/mini2440_nand.bin"
name_sd="$base/sd.img"
if [ ! -f "$name_nand" ]; then
echo $0 : creating NAND empty image : "$name_nand"
dd if=/dev/zero of="$name_nand" bs=528 count=131072
fi
cmd="$base/../arm-softmmu/qemu-system-arm \
-M mini2440 $* \
-serial stdio \
-mtdblock "$name_nand" \
-sd $name_sd \
-show-cursor \
-usb -usbdevice keyboard -usbdevice mouse \
-net nic,vlan=0 \
-net tap,vlan=0,ifname=tap1,script=/etc/qemu-ifup \
-monitor telnet::5555,server,nowait"
echo $cmd
$cmd
```

สร้างไฟล์ network.sh

```
$ vim network.sh
```

```
#!/bin/sh
sudo chown root.users /dev/net/tun
sudo chmod g+rw /dev/net/tun
sudo brctl addbr br0
sudo ifconfig eth0 0.0.0.0 promisc
sudo brctl addif br0 eth0
sudo dhclient br0
sudo tunctl -t tap1 -u `whoami`
```

```
$ chmod 755 network.sh
```

สร้างไฟล์ qemu-ifup ไว้ในไดเรกทอรี /etc/

```
$ vim qemu-ifup
#!/bin/sh
echo "Executing /etc/qemu-ifup"
echo "Bringing up $1 for bridged mode..."
sudo /sbin/ifconfig $1 0.0.0.0 promisc up
echo "Adding $1 to br0..."
sudo /sbin/brctl addif br0 $1
sleep 2
```

```
$ chmod 755 qemu-ifup
```

สร้างไฟล์ u-boot.bin

```
$ cd ~/eeburapha/qemu/mini2440/mini2440
$ mkdir uboot
$ cd uboot
$ git clone git://repo.or.cz/u-boot-openmoko/mini2440.git
$ cd mini2440
$ make mini2440_config
$ make all
dd if=u-boot.bin of=u-boot-nand2k.bin bs=2K conv=sync
119+1 records in
120+0 records out
245760 bytes (246 kB) copied, 0.000734641 s, 335 MB/s
dd if=u-boot.bin of=u-boot-nand16k.bin bs=16K conv=sync
14+1 records in
15+0 records out
245760 bytes (246 kB) copied, 0.000435712 s, 564 MB/s
```

แล้วให้ทำการคัดลอกไฟล์ u-boot.bin กลับไปยังไดเรกทอรี

~/eeburapha/qemu/mini2440/mini2440

สร้างไฟล์ sd image

```
$ cd eeburapha/qemu/mini2440/
```

```
$ qemu-img create sd.img 256M
Formatting 'sd.img', fmt=raw size=268435456
$ sudo modprobe nbd
$ sudo modprobe dm-mod
$ qemu-nbd -p 24561 sd.img &
$ sudo nbd-client localhost 24561 /dev/nbd0
Negotiation: ..size = 256MB
bs=1024, sz=268435456 bytes
$ sudo fdisk -u=cylinders /dev/nbd0
Device contains neither a valid DOS partition table, nor Sun, SGI or OSF
disklabel
Building a new DOS disklabel with disk identifier 0xe5f5bdf3.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by
w(rite)

WARNING: cylinders as display units are deprecated. Use command 'u' to
         change units to sectors.

Command (m for help): o
Building a new DOS disklabel with disk identifier 0x1999a406.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by
w(rite)

WARNING: cylinders as display units are deprecated. Use command 'u' to
         change units to sectors.

Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1): 1
First cylinder (1-32, default 1):
Using default value 1
Last cylinder, +cylinders or +size{K,M,G} (1-32, default 32): +50MB

Command (m for help): t
Selected partition 1
Hex code (type L to list codes): b
Changed system type of partition 1 to b (W95 FAT32)

Command (m for help): n
Partition type:
   p   primary (1 primary, 0 extended, 3 free)
   e   extended
Select (default p): p
Partition number (1-4, default 2): 2
First cylinder (7-32, default 7):
Using default value 7
```

```
Last cylinder, +cylinders or +size{K,M,G} (7-32, default 32):
Using default value 32

Command (m for help): p

Disk /dev/nbd0: 268 MB, 268435456 bytes
255 heads, 63 sectors/track, 32 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x1999a406

     Device Boot      Start         End      Blocks   Id  System
/dev/nbd0p1               1           7       55203    b  W95 FAT32
/dev/nbd0p2               7          32      200813   83  Linux

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: If you have created or modified any DOS 6.x
partitions, please see the fdisk manual page for additional
information.
Syncing disks.
```

```
$ sudo kpartx -a /dev/nbd0
$ sudo mkfs.vfat /dev/mapper/nbd0p1
$ sudo mkfs.ext3 /dev/mapper/nbd0p2
$ mkdir disk
$ sudo mount /dev/mapper/nbd0p2 ./disk
$ mkdir -p disk/boot
$ sudo mount /dev/mapper/nbd0p1 ./disk/boot
$ wget
http://mini2440.googlecode.com/files/emdebian-grip-090306-armel-lenny-
installed.tar.bz2
$ (cd disk; sudo tar jxf
../emdebian-grip-090306-armel-lenny-installed.tar.bz2)
$ sudo umount disk/boot disk
$ sudo nbd-client -d /deb/nbd0
```

แล้วให้ทำการคัดลอกไฟล์ sd.img ไปวางไว้ที่ ~/eeburapha/qemu/mini2440/mini2440 และเริ่มการรัน mini2440 ด้วยคำสั่ง

```
$ cd ~/eeburapha/qemu/mini2440/mini2440
$ ./network.sh
Set 'tap1' persistent and owned by uid 1000
$ ./ mini2440_start.sh

*** Warning - bad CRC or NAND, using default environment
```

```
USB:   S3C2410 USB Deviced
ERROR: usbd_device_event_irq(), 613: (3401ea64,1) NULL device or device->bus
ERROR: usbd_device_event_irq(), 613: (3401ea64,2) NULL device or device->bus
ERROR: usbd_device_event_irq(), 613: (3401ea64,3) NULL device or device->bus
In:    serial
Out:   serial
Err:   serial
MAC: 08:08:11:18:12:27
Hit any key to stop autoboot:  0
MINI2440 #
```

ทำการฟอร์แมท nand

```
# nand scrub
NAND scrub: device 0 whole chip
Warning: scrub option will erase all factory set bad blocks!
         There is no reliable way to recover them.
         Use this command only for testing purposes if you
         are sure of what you are doing!

Really scrub this NAND flash? <y/N>
Erasing at 0x3ffc000 -- 100% complete.
Bad block table not found for chip 0
Bad block table not found for chip 0
OK
```

สร้าง bad block table

```
# nand createbbt
Create BBT and erase everything ? <y/N>
Skipping bad block at  0x03ff0000
Skipping bad block at  0x03ff4000
Skipping bad block at  0x03ff8000
Skipping bad block at  0x03ffc000

Creating BBT. Please wait ...Bad block table not found for chip 0
Bad block table not found for chip 0
Bad block table written to 0x03ffc000, version 0x01
Bad block table written to 0x03ff8000, version 0x01
```

สร้างที่เก็บข้อมูลตัวแปรระบบ (env)

```
# dynenv set 40000
device 0 offset 0x40000, size 0x3fc0000
45 4e 56 30 - 00 00 04 00
```

ขั้นตอนสุดท้ายทำการ reset จะได้ emulator ที่พร้อมใช้งานโดยสามารถคัดลอกข้อมูลใส่ sd.img ได้

```
# reset
S3C: CLK=240 HCLK=240 PCLK=240 UCLK=57
QEMU mini2440_reset: loaded default u-boot from NAND
```

```
QEMU mini2440_reset: loaded override u-boot (size 3c000)
S3C: CLK=240 HCLK=240 PCLK=240 UCLK=48
S3C: CLK=304 HCLK=304 PCLK=304 UCLK=48
S3C: CLK=304 HCLK=101 PCLK=50 UCLK=48
S3C: CLK=304 HCLK=76 PCLK=38 UCLK=48
S3C: CLK=304 HCLK=76 PCLK=38 UCLK=48
S3C: CLK=405 HCLK=101 PCLK=50 UCLK=48

U-Boot 1.3.2-moko12 (Jun 26 2009 - 18:16:16)

I2C:   ready
DRAM:  64 MB
Flash:  2 MB
NAND:  64 MiB
Found Environment offset in OOB..
USB:   S3C2410 USB Deviced
In:    serial
Out:   serial
Err:   serial
MAC: 08:08:11:18:12:27
Hit any key to stop autoboot:  0
MINI2440 #
```

บทที่ 4 พื้นฐานการเขียนโปรแกรมภาษา C/C++ และ QT สำหรับนักพัฒนา

พื้นฐานการเขียนโปรแกรมภาษา C/C++ สำหรับการพัฒนาบนระบบสมองกลฝังตัว

ในมุมมองการพัฒนาโปรแกรมภายใต้ระบบปฏิบัติการลีนุกซ์ที่อยู่บนระบบสมองกลฝังตัวนั้นจะไม่แตกต่างจากการพัฒนาโปรแกรมภายใต้ระบบปฏิบัติการลีนุกซ์บนเครื่องคอมพิวเตอร์ทั่วไป ดังนั้นนักพัฒนาที่คุ้นเคยกับการเขียนโปรแกรมพื้นฐานไม่ว่าจะเป็นโปรแกรมภาษาซี/ซีพลัสพลัส ไม่จำเป็นต้องเพิ่มเติมทักษะอะไรมากมายเป็นพิเศษ เนื่องจากไลบรารีหรือโปรแกรมที่พัฒนาจากนักพัฒนาคนอื่นๆเกือบทั้งหมดนั้น จะยังคงสามารถนำมาคอมไพล์และทำงานได้เช่นเดิม แต่อย่างไรก็ตามสิ่งที่นักพัฒนาโปรแกรมภายใต้ระบบสมองกลฝังตัวต้องพึงระวังคือข้อจำกัดทางด้านทรัพยากรภายในระบบสมองกลฝังตัวไม่ว่าจะเป็นความเร็วในการประมวลผลกลาง พื้นที่หน่วยความจำ พื้นที่ตัวเก็บข้อมูลแบบชั่วคราวหรือถาวร รวมทั้งการใช้พลังงานไฟฟ้าจากแบตเตอรี่ เป็นต้น

ภาษาโปรแกรมที่เหมาะสมสำหรับระบบสมองกลฝังตัวคงปฏิเสธไม่ได้ที่จะเริ่มต้นด้วยภาษาซี (C Language) เนื่องจากเป็นภาษาพื้นฐานที่ใช้ในการพัฒนาโปรแกรมระบบภายในระบบปฏิบัติการลีนุกซ์และเป็นไลบรารีมากกว่าร้อยละ 80 ดังนั้นเมื่อนักพัฒนาติดตั้งระบบปฏิบัติการลีนุกซ์พื้นฐานลงไปในเครื่องคอมพิวเตอร์แล้วก็สามารถเริ่มต้นเขียนโปรแกรมได้ทันที รวมทั้งยังสามารถเรียกใช้ไลบรารีพื้นฐานต่างๆได้เช่น การจัดการระบบไฟล์ การจัดการอุปกรณ์ I/O การจัดการระบบเครือข่าย ระบบจัดการโปรเซส เป็นต้น นอกจากนั้นภาษาซีพลัสพลัส (C++) ซึ่งมีพื้นฐานของการเขียนโปรแกรมเชิงวัตถุ (Object oriented Programming - OOP) เหมาะกับนักพัฒนาที่ต้องการพัฒนาโปรแกรมประยุกต์ที่มีความซับซ้อนและมีขนาดใหญ่รวมถึงการใช้แนวคิดในการพัฒนาโปรแกรมแนวใหม่ด้วยวิธีการแบบ OOP นักพัฒนาจะต้องติดตั้งตัวคอมไพล์ (Compiler) เครื่องมือ (C++ Tools) และไลบรารี (C++ library) เพิ่มเติมเพื่อให้สามารถรองรับการเขียนโปรแกรมภาษาซีพลัสพลัสได้อย่างสมบูรณ์

พื้นฐานการสร้าง MAKEFILE

Makefile (GNU automake system) คือภาษาโปรแกรมประเภทหนึ่งที่เป็นการรวมเอาคำสั่งหรือสคริปท์ที่ช่วยให้การคอมไพล์โค้ดโปรแกรมเพื่อให้สะดวกขึ้น โดยขั้นตอนการคอมไพล์นั้นจะใช้คำสั่ง make ซึ่งเป็นคำสั่งที่สามารถถูกเรียกใช้งานได้อยู่บนเชลล์ โดยจะทำการอ่านรายละเอียดที่อยู่ภายในไฟล์ชื่อ “Makefile” ซึ่งบรรจุการตั้งค่าการคอมไพล์ต่างๆไว้ เช่น การอ้างอิงไลบรารีที่เกี่ยวข้อง โค้ดไฟล์เสริมที่เกี่ยวข้อง เป็นต้น องค์ประกอบภายใน Makefile ประกอบไปด้วย

```
(Macros)
     VARIABLE DECLARATIONS

(Targets)
     TARGET:   DEPENDENCIES
          RULES TO BUILD TARGET
```

โดยที่ *(Macros)* จะถูกเขียนอยู่ในรูปแบบการกำหนดค่าตัวแปร ตัวอย่างเช่น

```
CC=gcc
OPT=-g
```

เมื่อถูกเรียกภายใน *(Targets)* จะเขียนตามรูปแบบข้างล่างนี้

```
$(CC) a_source_file.c
```

นอกจากนั้นยังสามารถระบุได้มากกว่าหนึ่งตัวแปรเพื่อให้กลายเป็นตัวแปรใหม่ ดังตัวอย่างข้างล่าง

```
COMP = $(CC) $(OPT)
```

รายการ *(Targets)* คือรายการที่ระบุไว้ว่าให้ทำการคอมไพล์ไฟล์ที่ระบุไว้ (dependencies) ซึ่งวิธีการเขียนนั้นค่อนข้างมีรายละเอียดในการจัดรูปแบบโดยเฉพาะการใช้ whitespace (spacebar และ tab) ถ้าไม่ได้จัดตามรูปแบบดังแสดงข้างล่างคำสั่ง make ก็อาจจะไม่สามารถอ่านสคริปท์เพื่อใช้คอมไพล์ได้ ดังแสดงข้างล่าง

```
       tab             space       space
      |                |           |
target:   dependency1 dependency2 dependency3
     rule1
     rule2
     rule3
^^^^^
|
tab
```

ตัวอย่างไฟล์ Makefile อย่างง่ายแบบที่ 1

```
all: hello

hello: main.o factorial.o hello.o
	g++ main.o factorial.o hello.o -o hello
main.o: main.cpp
	g++ -c main.cpp
factorial.o: factorial.cpp
	g++ -c factorial.cpp
hello.o: hello.cpp
	g++ -c hello.cpp
clean:
	rm -rf *o hello
```

ตัวอย่างไฟล์ Makefile อย่างง่ายแบบที่ 2

ใช้ macro เพื่อสร้างตัวแปร CC และ CFLAGS

```
# the compiler to use
CC=g++
# options passed to the compiler
CFLAGS=-c -Wall

all: hello

hello: main.o factorial.o hello.o
	$(CC) main.o factorial.o hello.o -o hello
main.o: main.cpp
	$(CC) $(CFLAGS) main.cpp
factorial.o: factorial.cpp
	$(CC) $(CFLAGS) factorial.cpp
hello.o: hello.cpp
	$(CC) $(CFLAGS) hello.cpp
clean:
	rm -rf *o hello
```

ตัวอย่างไฟล์ Makefile อย่างง่ายแบบที่ 3

สร้างตัวแปร SOURCES, OBJECTS และ EXECUTABLE เพื่อรวมไฟล์โปรแกรมเข้าด้วยกัน

```
CC=g++
CFLAGS=-c -Wall
LDFLAGS=
SOURCES=main.cpp hello.cpp factorial.cpp
OBJECTS=$(SOURCES:.cpp=.o)
EXECUTABLE=hello

all: $(SOURCES) $(EXECUTABLE)
```

```
$(EXECUTABLE): $(OBJECTS)
	$(CC) $(LDFLAGS) $(OBJECTS) -o $@
.cpp.o:
	$(CC) $(CFLAGS) $< -o $@
clean:
	rm -rf $(OBJECTS) $(EXECUTABLE)
```

ตัวอย่างไฟล์ Makefile อย่างง่ายแบบที่ 4

ในกรณีที่มีการอ้างอิง (include) ไฟล์ header (.h) เพิ่มหรือไลบรารีภายนอกเพิ่มเติม จะต้องใส่ option -I และ -l เพิ่มเข้าไป

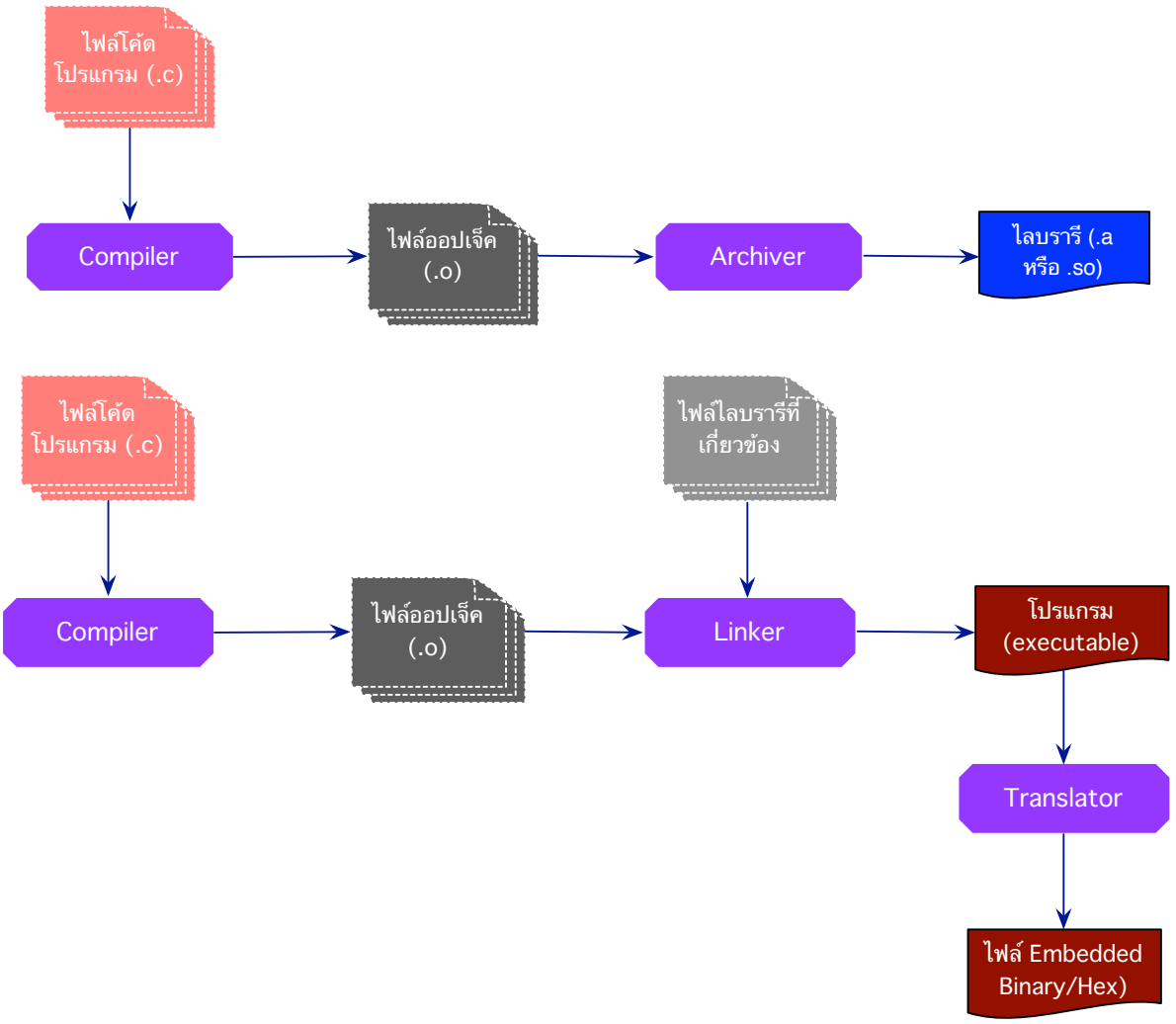
```
CC=g++
CFLAGS=-c -Wall
LDFLAGS=
SOURCES=main.cpp hello.cpp factorial.cpp
OBJECTS=$(SOURCES:.cpp=.o)
MY_INCLUDES=/home/student/project/myLib/
MY_LIBRARIES=zlib fltk
EXECUTABLE=hello

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
	$(CC) -I$(MY_INCLUDES) $(LDFLAGS) $(OBJECTS) -o $@ -l$(MY_LIBRARIES)
@echo "Compile hello completed..."
.cpp.o:
	$(CC) $(CFLAGS) $< -o $@
@echo "Compile all object files completed..."
clean:
	rm -rf $(OBJECTS) $(EXECUTABLE)
@echo "Clean all object files and hello program done..."
```

การสร้างและอ้างอิงไลบรารี

ขบวนการคอมไพล์ในระบบปฏิบัติการลีนุกซ์จากโค้ดโปรแกรม (source code) จนกลายมาเป็นโปรแกรมที่สามารถใช้งานหรือรันได้ (executable file) นั้น ประกอบไปด้วยหลายขั้นตอนดังแสดงในรูปข้างล่าง โดยมีขั้นตอนคือเมื่อมีการคอมไพล์โค้ดโปรแกรม (.c) จะได้ไฟล์ออกมาเป็นชนิดไบนารีไฟล์ ที่เรียกว่า ออพเจ็คโค้ด (object code) ซึ่งมีนามสกุลไฟล์เป็น ".o" ไฟล์ออพเจ็คโค้ดเหล่านี้สามารถถูกรวมกันโดยใช้โปรแกรม ar (archiver) เพื่อให้กลายเป็นไฟล์ไลบรารีหรือจะถูกแปลงให้เป็นโปรแกรมที่สามารถรันได้ (executable file) ด้วยโปรแกรม ld (linker)



รูปที่ 4.1 แสดงขั้นตอนการสร้างไฟล์ไลบรารีและโปรแกรมสำหรับเขียนลงบอร์ดสมองกลฝังตัว

ในกรณีที่เป็นระบบคอมพิวเตอร์ทั่วไป ตัวไลบรารีและไฟล์ออพเจ็คโค้ดนั้นจะถูกเชื่อมรวมกันจนกลายเป็นไฟล์โปรแกรม (executable file) แต่ในกรณีระบบสมองกลฝังตัวที่ใช้ไมโครคอนโทรเลอร์นั้นจะต้องนำไฟล์โปรแกรมมาแปลงให้เป็นไฟล์แบบไบนารีหรือไม่ก็เป็นไฟล์สกุล .hex เสียก่อนเพื่อที่จะสามารถถูกโหลดเข้าไปในหน่วยความจำแบบแฟลชของบอร์ดสมองกลฝังตัวได้

จากตัวอย่างข้างล่างเป็นตัวอย่างพื้นฐานในการเขียนโปรแกรมที่จะต้องมีการสร้างไลบรารีและพัฒนาโปรแกรมเพื่อเรียกใช้ไลบรารี ในตารางข้างล่างแสดงโค้ดโปรแกรมที่มีอยู่ 2 ไดเรกทอรีย่อยได้แก่ ไดเรกทอรี `src/` และ ไดเรกทอรี `inc/`

ไดเรกทอรี SRC/	ไดเรกทอรี INC/
`main.c`	
`func1.c`	`func1.h`
`func2.c`	`func2.h`

ไดเรกทอรี SRC/	ไดเรกทอรี INC/
func3.c	func3.h

รายละเอียดในการสร้างไลบรารีนั้นจะมีด้วยกัน 2 แบบคือ สแตติกไลบรารี (static library) และไดนามิกไลบรารี (dynamic library) ดังตัวอย่างข้างล่าง

ตัวอย่างการสร้างไลบรารีชนิด static

```
$ gcc -c src/func1.c src/func2.c src/func3.c -I./inc
$ ar libmine.a func1.o func2.o func3.o
```

ตัวอย่างการสร้างไลบรารีชนิด dynamic

```
$ gcc -shared -o libmine.so func1.o func2.o func3.o -I./inc
```

เมื่อสร้างไลบรารีเรียบร้อยแล้วสามารถนำไลบรารีเหล่านี้ไปใช้กับโปรเจคอื่นๆได้ ดังตัวอย่างข้างล่าง

```
$ gcc src/main.c libmine.a -I/inc -o myprogram
```

หรือ

```
$ gcc src/main.c -lmine -L./ -I/inc -o myprogram
```

เพื่อความสะดวกในการคอมไพล์โปรแกรมและอ้างอิงไลบรารีต่างๆ สามารถสร้างไฟล์ `Makefile` ดังตัวอย่างข้างล่าง

```
CC=gcc
CFLAGS=-c -Wall
LDFLAGS=
SOURCES=main.c
OBJECTS=$(SOURCES:.c=.o)
MY_INCLUDES=/home/student/project/inc/
# MY_STATIC_LIBRARIES=libmine.a
MY_DYNAMIC_LIBRARIES=mine
EXECUTABLE=myprogram

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
	$(CC) -I$(MY_INCLUDES) $(LDFLAGS) $(OBJECTS) -o $@ -l$(MY_DYNAMIC_LI-
BRARIES)
@echo “Compile hello completed...”

.c.o:
	$(CC) $(CFLAGS) $< -o $@
```

```
@echo “Compile all object files completed...”

clean:
      rm -rf $(OBJECTS) $(EXECUTABLE)
@echo “Clean all object files and hello program done...”
```

การพัฒนาโปรแกรมเพื่อเข้าถึงระบบไฟล์

ในระบบสมองกลฝังตัวนักพัฒนามีโอกาสในการที่จะต้องทำการบันทึกข้อมูลที่ได้จากการรับค่าสถานะจากเซนเซอร์ต่างๆเพื่อเก็บลงไปในไฟล์หรือจะทำการเปิดอ่านไฟล์ที่ระบบปฏิบัติการได้บันทึกข้อมูลไว้แล้ว ดังนั้นการพัฒนาโปรแกรมเพื่อเข้าถึงระบบไฟล์จึงเป็นพื้นฐานแรกๆของการเรียนรู้ของนักพัฒนา โดยการเรียกฟังก์ชันที่เตรียมไว้ให้ภายในไลบรารี stdio.h ดังตัวอย่างข้างล่าง

```
#include <stdio.h>

FILE *fopen(const char *path, const char *mode);
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
int fprintf(FILE *stream, const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);
int fflush(FILE *stream);
int fclose(FILE *stream);
```

การเข้าถึงระบบไฟล์นั้นมีอยู่ 3 โหมดได้แก่ อ่านได้เท่านั้น (O_RDONLY) เขียนได้เท่านั้น (O_WRONLY) และ อ่านและแก้ไขได้ (O_RDWR)

ตัวอย่างโปรแกรมสำหรับการอ่านไฟล์และแก้ไขไฟล์

```
// file_manipulation.c
#include <stdio.h>
#include <string.h>

int   usage(){

   printf("\n");
   printf("Usage : file Manipulation [OPTION] [FILE]\n");
   printf("Demo for text file operation, read and write. [Jamp II]\n");
   printf("  -r \t\t read operation\n");
   printf("  -w \t\t write operation\n");
   printf("\n");

   return 0;
}

int   check_operation(char *opt){
```

```
  if(!strcmp(opt, "-r")){     // read operation
      return 0;
  }
  else if(!strcmp(opt, "-w")){      // write operation
      return 1;
  }
  else{                       // error
      return -1;
  }
}

int   main(int argc, char *argv[]){

   FILE     *file;
   int      opt;

   // check for argument
   if(argc != 3){
      usage();

      return 0;
   }

   // check for getting operation
   if((opt = check_operation(argv[1])) < 0){
      usage();

      return 0;
   }

   // start file operation
   if(file = fopen(argv[2], (argv[1]+1))){
      char dataBuf[256];

      if(opt == 0){           // READ
         printf("****************** READ ******************\n");

         while(fgets(dataBuf, 256, file) != 0){
            printf("%s", dataBuf);

            memset(dataBuf, 0, 256);
         }
         printf("******************************************\n");
      }
      else{             // WRITE
         printf("***************** WRITE ******************\n");

         while(1){
            memset(dataBuf, 0, 256);

            gets(dataBuf);
```

```
            if(!strcmp(dataBuf, ":exit"))
               break;

            fprintf(file, "%s\n", dataBuf);
         }

         printf("****************************************\n");
      }

      fclose(file);
   }
   else{
      printf("Can not open file\n");
   }

   return 0;
}
```

ตัวอย่างไฟล์ Makefile สำหรับโปรแกรม file_manipulation.c เพื่อทำการคอมไพล์ด้วยโปรแกรมคำสั่ง arm-linux-gcc สำหรับสถาปัตยกรรม ARM

```
#
# Makefile : File Manipulation sample Program
#
# Macro
# CROSS_COMPILE=
CROSS_COMPILE=arm-linux-
CC=$(CROSS_COMPILE)gcc
CFLAGS=-Os
LDFLAGS=-Os -static

# Targets
all: file_manipulation

fileOpt: file_manipulation.o
      $(CC) $(LDFLAGS) $< -o $@

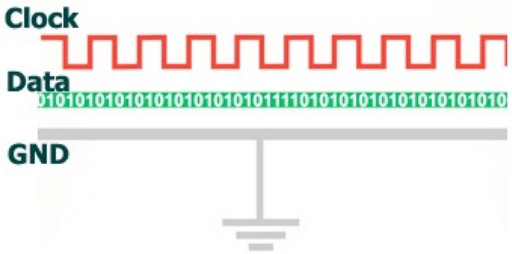
fileOpt.o: file_manipulation.c
      $(CC) $(CFLAGS) -c -o $@ $<

clean:
      rm -rf *.o file_manipulation
```

การพัฒนาโปรแกรมติดต่อผ่านพอร์ตอนุกรม

การสื่อสารข้อมูลแบบอนุกรมเป็นรูปแบบการสื่อสารข้อมูลรากฐานตัวหนึ่งที่ใช้กับอุปกรณ์คอมพิวเตอร์ผ่านพอร์ตอนุกรมเช่น โมเดม, เมาส์ หรือ คีย์บอร์ด โดยรูปแบบการสื่อสารของพอร์ตชนิดนี้จะเป็นการส่งข้อมูลแบบทีละบิต (bit-by-bit) บนมาตรฐานการส่งข้อมูลแบบ RS-232 ซึ่งจะต่างจากพอร์ตขนาน (parallel port) ที่จะมีการส่งข้อมูลพร้อมกันทีละหลายๆบิต (8-bit) โดยความเร็วของการส่งข้อมูลแบบอนุกรมจะขึ้นอยู่กับความถี่ที่เลือกใช้ในการส่งข้อมูล ข้อดีของการสื่อสารข้อมูลแบบอนุกรมคือสามารถส่งข้อมูลได้ในระยะทางที่ไกลกว่าและใช้สายสัญญาณที่น้อยกว่าการสื่อสารข้อมูลแบบขนาน ประเภทของการสื่อสารแบบอนุกรมแบ่งตามลักษณะสัญญาณในการส่งได้ 2 แบบ ได้แก่

1. **การสื่อสารแบบซิงโครนัส (Synchronous)** เป็นการสื่อสารข้อมูลโดยใช้สัญญาณนาฬิกาในการควบคุมจังหวะของการรับส่งสัญญาณ



รูปที่ 4.2 แสดงการส่งข้อมูลแบบซิงโครนัส

2. **การสื่อสารแบบอะซิงโครนัส (Asynchronous)** เป็นการสื่อสารที่ใช้สายข้อมูลเพียงตัวเดียว จะใช้รูปแบบของการส่งข้อมูล (Bit Pattern) เป็นตัวกำหนดว่าส่วนไหนเป็นส่วนเริ่มต้นข้อมูล ส่วนไหนเป็นตัวข้อมูล ส่วนไหนจะเป็นตัวตรวจสอบความถูกต้องของข้อมูล และส่วนไหนเป็นส่วนปิดท้ายของข้อมูล โดยต้องกำหนดให้สัญญาณนาฬิกาเท่ากันทั้งภาคส่งและภาครับ



รูปที่ 4.3 แสดงการส่งข้อมูลแบบอะซิงโครนัส

ระบบสื่อสารภายในระบบสมองกลฝังตัวส่วนใหญ่จะเป็นแบบอะซิงโครนัส ซึ่งมีความสามารถในการทำหน้าที่รับและส่งข้อมูลแบบไม่ต้องมีสัญญาณนาฬิกาพ่วงไปด้วย เรียกว่า UART (Universal Asynchronous Receiver Transmitter) โดยส่วนใหญ่จะใช้มาตรฐานการสื่อสารแบบ RS-232 ถ้าติดตั้งบนคอมพิวเตอร์ทั่วไปจะเรียกว่า COMPORT เมื่อเข้ามาตรฐาน RS-232 แล้วระดับแรงดันต้องปรับเปลี่ยนให้เป็นไฟบวกลบเพื่อลดสัญญาณรบกวนและสามารถส่งไปได้ระยะทางหลายสิบเมตร สำหรับในไมโครคอนโทรลเลอร์ เช่น PIC, MCS-51 หรือ AVR ส่วนใหญ่จะมีโมดูล UART ที่มีอยู่ภายในเพื่อช่วยในการสื่อสารข้อมูลระหว่างตัวไมโครคอนโทรลเลอร์และคอมพิวเตอร์แต่จะมีระดับสัญญาณเป็นระดับสัญญาณ TTL (5V) หากต้องการเชื่อมต่อกับคอมพิวเตอร์โดยตรงจะต้องต่อวงจรเพิ่มเติมเพื่อทำการแปลงระดับแรงดันด้วยไอซี MAX232

ขาสัญญาณที่ใช้ในการสื่อสารส่วนใหญ่จะมี 2 ขาหลักคือขารับข้อมูล (RxD) และ ขาส่งข้อมูล (TxD) โดยมาตรฐานการสื่อสารจะต้องตรงกันทั้งตัวรับและตัวส่ง เช่นบอดเรต (baudrate) 9600 บิตต่อวินาที จำนวนบิตเริ่ม บิตหยุด บิตพาริตี้ และจำนวนบิตข้อมูล

ตาราง 4.1 ไฟล์ชนิด character ต่างๆที่อยู่ภายในไดเรกทอรี /dev

อุปกรณ์	ชื่อไฟล์	MAJOR	MINOR
Parallel port 0	/dev/lp0 หรือ /dev/par0	6	0
Parallel port 1	/dev/lp1 หรือ /dev/par1	6	1
1st Serial port	/dev/ttyS0	4	64
2nd Serial port	/dev/ttyS1	4	65
IDE tape drive	/dev/ht0	37	0
1st SCSI tape drive	/dev/st0	9	0
2nd SCSI tape drive	/dev/st1	9	1
System Console	/dev/console	5	1
1st Virtual Terminal	/dev/tty1	4	1
2nd Virtual Terminal	/dev/tty2	4	2
Process current terminal	/dev/tty	5	0
Sound card	/dev/audio	14	4

ตัวอย่างการเขียนโปรแกรมเพื่อติดต่อผ่านพอร์ตอนุกรม (**/dev/ttyS*x*** หรือ **/dev/ttyUSB*x***) ภายใต้ระบบปฏิบัติการลีนุกซ์

```
// serial_thread.c
#include <stdio.h>   /* Standard input/output definitions */
#include <string.h>  /* String function definitions */
#include <unistd.h>  /* UNIX standard function definitions */
#include <fcntl.h>   /* File control definitions */
#include <errno.h>   /* Error number definitions */
#include <termios.h> /* POSIX terminal control definitions */
#include <pthread.h> /* Create pthread to run reading prog in background */

struct serial_data {
      int fd, nbytes, portsite; /* device file */
      char *bufptr; /* set pointer of buffer */
      char buffer[255], data[255]; /* target buffer to get from pointer */
};

struct serial_data ffuart;

void *ffuart_thread() {
      ffuart.portsite = 1;
```

```
	ffuart.fd = open("/dev/ttyS1", O_RDWR | O_NOCTTY | O_NDELAY);
	if (ffuart.fd == -1) {
		/* Could not open the port. */
		perror("open_port: Unable to open /dev/ttyS1 - ");
	} else {
		struct termios options;
		fcntl(ffuart.fd, F_SETFL, 0);
		tcgetattr(ffuart.fd, &options);
		cfsetispeed(&options, B9600);
		cfsetospeed(&options, B9600);
		options.c_cflag |= (CLOCAL | CREAD);
		options.c_cflag |= PARENB;
		options.c_cflag &= ~PARODD;
		options.c_cflag &= ~CSTOPB;
		options.c_cflag &= ~CSIZE;
		options.c_cflag |= CS8;
		options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
		options.c_oflag &= ~OPOST;
		options.c_cc[VMIN] = 0;
		options.c_cc[VTIME] = 10;
		tcsetattr(ffuart.fd, TCSANOW, &options);
		printf("Port is scanning. ^^\n");
		ffuart.bufptr = ffuart.buffer; /* set buffer to get from pointer */
		printf("Hey! You Reading yet.\n");
		printf("Now working in ST Port.\n");
		while (1) {
			printf("ReadingFF\n");
			ffuart.nbytes = read(ffuart.fd, ffuart.buffer, 255);
			if (ffuart.nbytes != 0) {
				ffuart.buffer[8] = '\0';
				printf("From %d side : %s\n", ffuart.portsite,
ffuart.buffer);
			}
		}
	}
	close(ffuart.fd);
}

int main() {
	pthread_t ff_tid;
	int rc;
	printf("FF_Thread creating!!\n");
	rc = pthread_create(&ff_tid, NULL, ffuart_thread, (void *) NULL);
	if (rc) {
		printf("ERROR; return code from pthread_create() is %d\r\n", rc);
		exit(-1);
	}
	printf("FF_Thread created!!\t[Complete!!]\n");
	pthread_join(ff_tid, NULL);
	printf("Created all!!\t[Complete!!]\n");
	pthread_exit(NULL);
}
```

ตัวอย่างไฟล์ Makefile สำหรับโปรแกรม serial_thread.c ที่ทำงานได้บนสถาปัตยกรรม ARM

```
#
# Makefile      : Serial communication Sample Program
#

CROSS_COMPILE=arm-linux-uclibc-
CC=$(CROSS_COMPILE)gcc
SOURCES=serial_thread.c
OBJECTS=$(SOURCES:.c=.o)
CFLAGS=-Os
LDFLAGS=-Os -lm -static
LIBRARIES=pthread
EXECUTABLE=serial_thread

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
     $(CC) $(LDFLAGS) $(OBJECTS) -o $@ -l$(LIBRARIES)

.c.o:
     $(CC) $(CFLAGS) $< -o $@

clean:
     rm -rf $(OBJECTS) $(EXECUTABLE)
```

ตัวอย่างถัดไปคือตัวอย่างการสื่อสารผ่านพอร์ตอนุกรมระหว่างเครื่องคอมพิวเตอร์ 2 เครื่อง

```
// serial_chat.c
#include <termios.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/signal.h>
#include <sys/types.h>

#define BAUDRATE B38400
#define MODEMDEVICE "/dev/ttyS1"
#define _POSIX_SOURCE 1         //POSIX compliant source
#define FALSE 0
#define TRUE 1

volatile int STOP=FALSE;

void signal_handler_IO (int status);    //definition of signal handler
int wait_flag=TRUE;                     //TRUE while no signal received
char devicename[80];
long Baud_Rate = 38400;         // default Baud Rate (110 through 38400)
long BAUD;                      // derived baud rate from command line
long DATABITS;
long STOPBITS;
```

```
long PARITYON;
long PARITY;
int Data_Bits = 8;              // Number of data bits
int Stop_Bits = 1;              // Number of stop bits
int Parity = 0;                 // Parity as follows:
                  // 00 = NONE, 01 = Odd, 02 = Even, 03 = Mark, 04 = Space
int Format = 4;
FILE *input;
FILE *output;
int status;

main(int Parm_Count, char *Parms[])
{
   char version[80] = "       POSIX compliant Communications test program version
1.00 4-25-1999\r\n";
   char version1[80] = "          Copyright(C) Mark Zehner/Peter Baumann 1999\r\n";
   char version2[80] = " This code is based on a DOS based test program by Mark Zeh-
ner and a Serial\r\n";
   char version3[80] = " Programming POSIX howto by Peter Baumann, integrated by
Mark Zehner\r\n";
   char version4[80] = " This program allows you to send characters out the speci-
fied port by typing\r\n";
   char version5[80] = " on the keyboard.  Characters typed will be echoed to the
console, and \r\n";
   char version6[80] = " characters received will be echoed to the console.\r\n";
   char version7[80] = " The setup parameters for the device name, receive data for-
mat, baud rate\r\n";
   char version8[80] = " and other serial port parameters must be entered on the
command line \r\n";
   char version9[80] = " To see how to do this, just type the name of this program.
\r\n";
   char version10[80] = " This program is free software; you can redistribute it
and/or modify it\r\n";
   char version11[80] = " under the terms of the GNU General Public License as pub-
lished by the \r\n";
   char version12[80] = " Free Software Foundation, version 2.\r\n";
   char version13[80] = " This program comes with ABSOLUTELY NO WARRANTY.\r\n";
   char instr[100] ="\r\nOn the command you must include six items in the following
order, they are:\r\n";
   char instr1[80] ="   1.  The device name      Ex: ttyS0 for com1, ttyS1 for com2,
etc\r\n";
   char instr2[80] ="   2.  Baud Rate            Ex: 38400 \r\n";
   char instr3[80] ="   3.  Number of Data Bits  Ex: 8 \r\n";
   char instr4[80] ="   4.  Number of Stop Bits  Ex: 0 or 1\r\n";
   char instr5[80] ="   5.  Parity               Ex: 0=none, 1=odd, 2=even\r\n";
   char instr6[80] ="   6.  Format of data received:  1=hex, 2=dec, 3=hex/asc,
4=dec/asc, 5=asc\r\n";
   char instr7[80] =" Example command line:  serial_chat ttyS0 38400 8 0 0 4 \r\n";
   char Param_strings[7][80];
   char message[90];

   int fd, tty, c, res, i, error;
```

```
   char In1, Key;
//place for old and new port settings for serial port
   struct termios oldtio, newtio;
//place tor old and new port settings for keyboard teletype
   struct termios oldkey, newkey;
   struct sigaction saio;               //definition of signal action
   char buf[255];                       //buffer for where data is put

   input = fopen("/dev/tty", "r");      //open the terminal keyboard
   output = fopen("/dev/tty", "w");     //open the terminal screen

   if (!input || !output)
   {
      fprintf(stderr, "Unable to open /dev/tty\n");
      exit(1);
   }

   error=0;
   fputs(version,output);               //display the program introduction
   fputs(version1,output);
   fputs(version2,output);
   fputs(version3,output);
   fputs(version4,output);
   fputs(version5,output);
   fputs(version6,output);
   fputs(version7,output);
   fputs(version8,output);
   fputs(version9,output);
   fputs(version10,output);
   fputs(version11,output);
   fputs(version12,output);
   fputs(version13,output);
   //read the parameters from the command line
   if (Parm_Count==7)
//if there are the right number of parameters on the command line
   {
      for (i=1; i<Parm_Count; i++)  // for all wild search parameters
      {
         strcpy(Param_strings[i-1],Parms[i]);
      }
      i=sscanf(Param_strings[0],"%s",devicename);
      if (i != 1) error=1;
      i=sscanf(Param_strings[1],"%li",&Baud_Rate);
      if (i != 1) error=1;
      i=sscanf(Param_strings[2],"%i",&Data_Bits);
      if (i != 1) error=1;
      i=sscanf(Param_strings[3],"%i",&Stop_Bits);
      if (i != 1) error=1;
      i=sscanf(Param_strings[4],"%i",&Parity);
      if (i != 1) error=1;
      i=sscanf(Param_strings[5],"%i",&Format);
      if (i != 1) error=1;
```

```
      sprintf(message,"Device=%s, Baud=%li\r\n",devicename, Baud_Rate);
//output the received setup parameters
      fputs(message,output);
      sprintf(message,"Data Bits=%i  Stop Bits=%i  Parity=%i  Format=%i\r\n",D-
ata_Bits, Stop_Bits, Parity, Format);
      fputs(message,output);
   }  //end of if param_count==7
   if ((Parm_Count==7) && (error==0))  //if the command line entrys were correct
   {                                   //run the program
      tty = open("/dev/tty", O_RDWR | O_NOCTTY | O_NONBLOCK);
//set the user console port up
      tcgetattr(tty,&oldkey);
// save current port settings
//so commands are interpreted right for this program
// set new port settings for non-canonical input processing  //must be NOCTTY
      newkey.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
      newkey.c_iflag = IGNPAR;
      newkey.c_oflag = 0;
      newkey.c_lflag = 0;       //ICANON;
      newkey.c_cc[VMIN]=1;
      newkey.c_cc[VTIME]=0;
      tcflush(tty, TCIFLUSH);
      tcsetattr(tty,TCSANOW,&newkey);

      switch (Baud_Rate)
      {
         case 38400:
         default:
            BAUD = B38400;
            break;
         case 19200:
            BAUD  = B19200;
            break;
         case 9600:
            BAUD  = B9600;
            break;
         case 4800:
            BAUD  = B4800;
            break;
         case 2400:
            BAUD  = B2400;
            break;
         case 1800:
            BAUD  = B1800;
            break;
         case 1200:
            BAUD  = B1200;
            break;
         case 600:
            BAUD  = B600;
            break;
         case 300:
```

```
            BAUD  = B300;
            break;
         case 200:
            BAUD  = B200;
            break;
         case 150:
            BAUD  = B150;
            break;
         case 134:
            BAUD  = B134;
            break;
         case 110:
            BAUD  = B110;
            break;
         case 75:
            BAUD  = B75;
            break;
         case 50:
            BAUD  = B50;
            break;
      }  //end of switch baud_rate
      switch (Data_Bits)
      {
         case 8:
         default:
            DATABITS = CS8;
            break;
         case 7:
            DATABITS = CS7;
            break;
         case 6:
            DATABITS = CS6;
            break;
         case 5:
            DATABITS = CS5;
            break;
      }  //end of switch data_bits
      switch (Stop_Bits)
      {
         case 1:
         default:
            STOPBITS = 0;
            break;
         case 2:
            STOPBITS = CSTOPB;
            break;
      }  //end of switch stop bits
      switch (Parity)
      {
         case 0:
         default:                        //none
            PARITYON = 0;
```

```
            PARITY = 0;
            break;
         case 1:                        //odd
            PARITYON = PARENB;
            PARITY = PARODD;
            break;
         case 2:                        //even
            PARITYON = PARENB;
            PARITY = 0;
            break;
      }  //end of switch parity

  //open the device(com port) to be non-blocking (read will return immediately)
      fd = open(devicename, O_RDWR | O_NOCTTY | O_NONBLOCK);
      if (fd < 0)
      {
         perror(devicename);
         exit(-1);
      }

   //install the serial handler before making the device asynchronous
      saio.sa_handler = signal_handler_IO;
      sigemptyset(&saio.sa_mask);   //saio.sa_mask = 0;
      saio.sa_flags = 0;
      saio.sa_restorer = NULL;
      sigaction(SIGIO,&saio,NULL);

   // allow the process to receive SIGIO
      fcntl(fd, F_SETOWN, getpid());
      // Make the file descriptor asynchronous (the manual page says only
      // O_APPEND and O_NONBLOCK, will work with F_SETFL...)
      fcntl(fd, F_SETFL, FASYNC);

      tcgetattr(fd,&oldtio); // save current port settings
      // set new port settings for canonical input processing
      newtio.c_cflag = BAUD | CRTSCTS | DATABITS | STOPBITS | PARITYON | PARITY |
CLOCAL | CREAD;
      newtio.c_iflag = IGNPAR;
      newtio.c_oflag = 0;
      newtio.c_lflag = 0;       //ICANON;
      newtio.c_cc[VMIN]=1;
      newtio.c_cc[VTIME]=0;
      tcflush(fd, TCIFLUSH);
      tcsetattr(fd,TCSANOW,&newtio);

     // loop while waiting for input. normally we would do something useful here
      while (STOP==FALSE)
      {
         status = fread(&Key,1,1,input);
         if (status==1)  //if a key was hit
         {
            switch (Key)
```

```
            { /* branch to appropriate key handler */
               case 0x1b: /* Esc */
                  STOP=TRUE;
                  break;
               default:
                  fputc((int) Key,output);
//                  sprintf(message,"%x ",Key);  //debug
//                  fputs(message,output);
                  write(fd,&Key,1);          //write 1 byte to the port
                  break;
            }  //end of switch key
         }  //end if a key was hit
 // after receiving SIGIO, wait_flag = FALSE, input is available and can be read
         if (wait_flag==FALSE)  //if input is available
         {
            res = read(fd,buf,255);
            if (res>0)
            {
               for (i=0; i<res; i++)  //for all chars in string
               {
                  In1 = buf[i];
                  switch (Format)
                  {
                     case 1:         //hex
                        sprintf(message,"%x ",In1);
                        fputs(message,output);
                        break;
                     case 2:         //decimal
                        sprintf(message,"%d ",In1);
                        fputs(message,output);
                        break;
                     case 3:         //hex and asc
                        if ((In1<32) || (In1>125))
                        {
                           sprintf(message,"%x",In1);
                           fputs(message,output);
                        }
                        else fputc ((int) In1, output);
                        break;
                     case 4:         //decimal and asc
                     default:
                        if ((In1<32) || (In1>125))
                        {
                           sprintf(message,"%d",In1);
                           fputs(message,output);
                        }
                        else fputc ((int) In1, output);
                        break;
                     case 5:         //asc
                        fputc ((int) In1, output);
                        break;
                  }  //end of switch format
```

```
               }  //end of for all chars in string
            }  //end if res>0
//            buf[res]=0;
//            printf(":%s:%d\n", buf, res);
//            if (res==1) STOP=TRUE; /* stop loop if only a CR was input */
            wait_flag = TRUE;      /* wait for new input */
         }  //end if wait flag == FALSE

      }  //while stop==FALSE
      // restore old port settings
      tcsetattr(fd,TCSANOW,&oldtio);
      tcsetattr(tty,TCSANOW,&oldkey);
      close(tty);
      close(fd);        //close the com port
   }  //end if command line entrys were correct
   else  //give instructions on how to use the command line
   {
      fputs(instr,output);
      fputs(instr1,output);
      fputs(instr2,output);
      fputs(instr3,output);
      fputs(instr4,output);
      fputs(instr5,output);
      fputs(instr6,output);
      fputs(instr7,output);
   }
   fclose(input);
   fclose(output);
}  //end of main

/***************************************************************************
* signal handler. sets wait_flag to FALSE, to indicate above loop that     *
* characters have been received.                                           *
***************************************************************************/

void signal_handler_IO (int status)
{
//    printf("received SIGIO signal.\n");
   wait_flag = FALSE;
}
```

ตัวอย่างไฟล์ Makefile สำหรับโปรแกรม serial_thread.c ที่ทำงานบนสถาปัตยกรรม x86 โดยถูกตั้งค่าการคอมไพล์ให้ใช้โปรแกรมคำสั่ง gcc

```
#
# Makefile     : Serial chat sample Program
#

# CROSS_COMPILE=arm-linux-uclibc-
CC=$(CROSS_COMPILE)gcc
```

```
SOURCES=serial_chat.c
OBJECTS=$(SOURCES:.c=.o)
CFLAGS=-Os
LDFLAGS=-Os -lm -static
LIBRARIES=
EXECUTABLE=serial_chat

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
	$(CC) $(LDFLAGS) $(OBJECTS) -o $@ -l$(LIBRARIES)

.c.o:
	$(CC) $(CFLAGS) $< -o $@

clean:
	rm -rf $(OBJECTS) $(EXECUTABLE)
```

จากโปรแกรมข้างต้นแสดงตัวอย่างการสื่อสารข้อมูลระหว่างคอมพิวเตอร์ 2 เครื่องผ่านพอร์ตอนุกรม (COM1) โดยการรันโปรแกรมเพื่อเริ่มสื่อสารดังต่อไปนี้

```
$ serial_chat /dev/ttyS0 38400 8 1 0 4
```

โดยที่:

- ผ่านพอร์ต `/dev/ttyS0`
- baud rate ที่ 38,400 bps
- ขนาดข้อมูล 8 บิต
- ขนาดบิตหยุด (stop bit) 1 บิต
- ไม่มี parity bit
- แสดงเป็นเลขฐานสิบ

การพัฒนาโปรแกรมสื่อสารระหว่างโปรเซส

การสื่อสารระหว่างโปรเซส (Interprocess Communication - IPC) เป็นวิธีการแลกเปลี่ยนข้อมูลระหว่างโปรเซสที่อยู่ภายในคอมพิวเตอร์เดียวกันหรือโปรเซสที่อาจจะอยู่ต่างคอมพิวเตอร์แต่ถูกเชื่อมต่อผ่านระบบเครือข่ายเดียวกัน ตัวอย่างเช่น การเรียกคำสั่งผ่านเชลล์โดยผลลัพธ์จากโปรเซสแรกจะส่งไปเป็นอินพุทให้กับโปรเซสถัดไป

```
$ ls -al | grep "source"
drwxr-xr-x 24 root root     4096 2013-09-25 04:51 linux-source-2.6.32
```

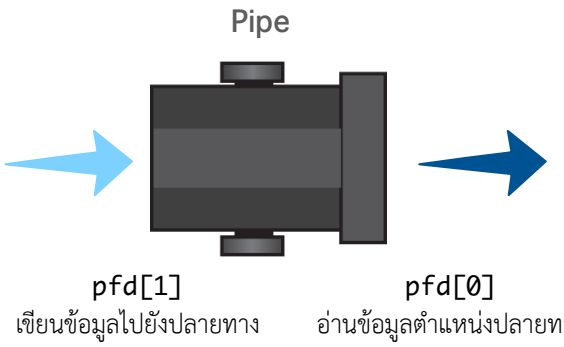
จากคำสั่งข้างต้นจะมีการใช้สัญลักษณ์ไปป์ (Pipe) เพื่อเป็นท่อเชื่อมในการนำส่งข้อมูลจากคำสั่งด้านซ้ายไปยังคำสั่งด้านขวา ซึ่งขบวนการในการสื่อสารระหว่างโปรเซสนั้นมีอยู่ด้วยกันหลายวิธีดังแสดงในตารางข้างล่าง

ตาราง 4.2 วิธีการสื่อสารในแบบต่างๆของ IPC

METHOD	DESCRIPTION
File	บันทึกเก็บไว้ในดิสก์ โดยที่โปรแกรมอื่นสามารถเข้าถึงได้
Signal	เป็นการส่งสัญญาณของระบบในระดับล่างไปขัดจังหวะของโปรเซส เพื่อบอกให้โปรเซสทำตามวัตถุประสงค์ของสัญญาณนั้นๆ หรืออาจจะใช้เป็นการส่งสัญญาณระหว่างโปรเซสด้วยกัน
Socket	ส่งข้อมูลไปยังเครือข่ายเน็ทเวิร์คบนคอมพิวเตอร์เครื่องเดียวกันหรือไปยังคอมพิวเตอร์เครื่องอื่น
Pipe	สตรีมข้อมูลระหว่างโพรเซสชนิดทางเดียว หรือ half duplex
Message queue	สตรีมข้อมูลที่ไม่ระบุคล้ายกับไปป์ แต่เก็บและเรียกข้อมูลจากในแพ็กเกจ
Named pipe	ดำเนินการผ่านไฟล์บนระบบไฟล์แทนมาตรฐาน input และ output
Semaphore	โครสร้างมาตรฐานที่ประสาน thread หรือกระบวนการที่กระทำกับทรัพยากรที่ใช้ร่วมกัน
Shared memory	เป็นวิธีการส่งข้อมูลไปไว้ในส่วนของหน่วยความจำที่แชร์ให้กับโปรเซสอื่นๆ ให้สามารถเข้าถึงข้อมูลนั้นได้
Message passing (shared nothing)	เหมือนกับ message queue
Memory-mapped file	แมพไฟล์ไปยัง RAM และสามารถแก้ไขได้โดยการเปลี่ยนที่อยู่หน่วยความจำโดยตรง

Pipe

ไปป์ (Pipe) เป็นกระบวนการที่ใช้ในการติดต่อระหว่างโปรเซสที่ง่ายที่สุดคือการนำผลลัพธ์ที่ได้จากโปรแกรมหนึ่งไปเป็นอินพุทของอีกโปรแกรมหนึ่ง โดยไปป์จะเป็นตัวกลางในการส่งข้อมูลระหว่างโปรเซส ซึ่งการส่งข้อมูลจะเป็นแบบทิศทางเดียว (unidirectional)



รูปที่ 4.4 แสดงการส่งข้อมูลแบบทิศทางเดียวด้วย Pipe

การพัฒนาโปรแกรมสื่อสารด้วยวิธีการสตรีมข้อมูลระหว่างโพรเซสชนิดทางเดียวนี้จะใช้ฟังก์ชัน pipe ที่อยู่ภายในไฟล์ไลบรารีชื่อว่า unistd.h

```
#include    <unistd.h>
int pipe ( int pipefd[2] );  <---  pipefd  - ตัวแปร array สำหรับเก็บค่า file descriptors
                                   pipefd[0] - file descriptor สำหรับการอ่าน
                                   pipefd[1] - file descriptor สำหรับการเขียน
```

สถานะการทำงานของ pipe

0 - ดำเนินการได้สำเร็จ, -1 - การดำเนินการล้มเหลว

จากรูปข้างบนแสดงการทำงานของฟังก์ชัน pipe ซึ่งจะสร้างช่องทางสื่อสารชนิดทางเดียวระหว่างโปรเซส โดยจะคืนค่า file descriptor ทั้งสองฝั่ง ผ่านตัวแปร pipefd[] กล่าวคือ ปลายหนึ่งสำหรับอ่าน (pipefd[0]) และอีกปลายหนึ่งสำหรับเขียน (pipefd[1]) ค่าของ file descriptor เป็นชนิดจำนวนเต็ม (int) ที่ระบบปฏิบัติการลีนุกซ์จะใช้ในการอ้างอิงถึงแฟ้มที่มีการเปิดใช้งานดังนั้นเมื่อเรียกใช้งานฟังก์ชัน pipe จะต้องส่งอาเรย์ของจำนวนเต็มที่มีสมาชิก 2 ตัวให้แก่ ฟังก์ชัน pipe()

```
  int  pipefd[2];       // file descriptor ของ pipe
```

ตัวอย่างโปรแกรม pipe

```
// pipe.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(int argc, char *argv[]){
      pid_t pid;
      int mypipefd[2];
      int ret;
      char buf[20];

      ret = pipe(mypipefd);
      if(ret == -1) {
            perror("pipe");
            exit(1);
      }
      pid = fork();
      if(pid == 0) {
            /* Child Process */
            printf("Child Process\n");
            write(mypipefd[1], "Hello there!", 12);
}else{
            /* Parent Process */
            printf("Parent Process\n");
            read(mypipefd[0], buf, 15);
```

```
            printf("buf : %s\n", buf);
        }
        return 0;
}
```

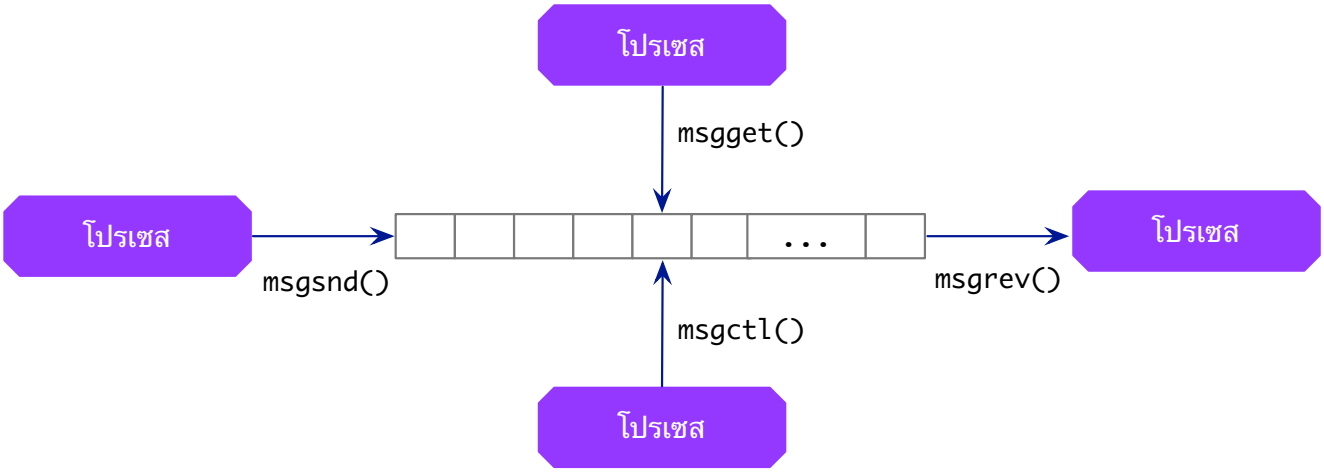
คอมไพล์โปรแกรม pipe.c และทดสอบการส่งข้อมูลระหว่างโปรเซสทั้งสอง (โปรเซส parent และโปรเซส child)

```
$ gcc -o pipe pipe.c
$ ./pipe
Parent Process
Child Process
buf : Hello there!
```

Message Queues

จะมีลักษณะคล้ายกับไฟล์ pipe คือสามารถส่งบล็อกข้อมูลจากโปรเซสหนึ่งไปยังอีกโปรเซสหนึ่งผ่านลีนุกซ์คอร์เนลเช่นเดียวกัน โดยแต่ละบล็อกข้อมูลจะมีหมายเลขอ้างอิงกำหนดไว้ให้เพื่อให้โปรเซสอื่นๆสามารถอ้างอิงหมายเลขข้อมูลที่ต้องการนั้นได้ ซึ่งจะไม่เหมือนกันการส่งข้อมูลผ่าน pipe ซึ่งทั้งสองโปรเซสจะต้องรอซึ่งกันและกัน ดังนั้นขั้นตอนการส่งข้อมูลแบบ message queue จะมีขั้นตอนคือโปรเซสที่ทำการเขียนข้อมูลไปยัง queue (ด้วยคำสั่ง **msgsnd**) แล้ว ก็สามารถจบการทำงานได้ทันที ถ้าโปรเซสใดต้องการอ่าน ก็สามารถเข้าไปอ่านข้อมูลใน queue ได้ภายหลัง (ด้วยคำสั่ง **msgrcv**) ดังแสดงในรูปข้างล่าง

แต่อย่างไรก็ตาม message queue ก็ยังมีข้อจำกัดเรื่องขนาดบล็อกข้อมูล ที่จะถูกจำกัดขนาดไว้ รวมทั้งจำนวนบล็อกทั้งหมดที่มีบนระบบก็จะถูกจำกัดไว้เช่นกัน ซึ่งภายในระบบปฏิบัติการลีนุกซ์ตัวแปรบล็อกข้อมูลจะประกอบไปด้วย 2 ส่วนคือ MSGMAX(4096) และ MSGMNB(16384) ซึ่งตัวแรกจะหมายถึงความจุของแต่ละบล็อกข้อความ และตัวที่สองจะหมายถึงความจุทั้งหมดของ queue แต่ในบางระบบค่าเหล่านี้อาจจะแตกต่างกันไป หรืออาจจะไม่ใช้ค่านี้เลยก็เป็นได้



รูปที่ 4.5 แสดงการส่งข้อมูลแบบ message queue

รายการฟังก์ชันภายในไลบรารี sys/msg.h ที่ใช้ใน Message Queues

```
#include<sys/msg.h>

int msgct1(int msqid, int cmd, struct msqid_ds *buf);
int megget1(key_t key, int ms_gflg);
int msgrcv(int msqid, void *msg_ptr, size_t msg_sz, long int msgtype, int msgflg);
int msgsnd(int msqid, const void *msg_ptr, size_t msg_sz, int msgflg);
```

ตัวอย่างโปรแกรมสำหรับส่งบล๊อกข้อมูลเข้าไปใน Message Queues

```
/*
** msg_sender.c -- writes to a message queue
*/
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

struct my_msgbuf {
    long mtype;
    char mtext[200];
};

int main(void)
{
    struct my_msgbuf buf;
    int msqid;
    key_t key;

    if ((key = ftok("msg_sender.c", 'B')) == -1) {
        perror("ftok");
        exit(1);
    }

    if ((msqid = msgget(key, 0644 | IPC_CREAT)) == -1) {
        perror("msgget");
        exit(1);
    }
    printf("Enter lines of text, ^D to quit:\n");

    buf.mtype = 1; /* we don't really care in this case */
    while(gets(buf.mtext), !feof(stdin)) {
        if (msgsnd(msqid, (struct msgbuf *)&buf, sizeof(buf), 0) == -1)
            perror("msgsnd");
    }

    if (msgctl(msqid, IPC_RMID, NULL) == -1) {
        perror("msgctl");
```

```
        exit(1);
    }

    return 0;
}
```

ตัวอย่างโปรแกรมสำหรับอ่านบล๊อกข้อมูล ภายใน Message Queues

```
/*
** msg_receiver.c -- reads from a message queue
*/

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

struct my_msgbuf {
    long mtype;
    char mtext[200];
};

int main(void)
{
    struct my_msgbuf buf;
    int msqid;
    key_t key;

    if ((key = ftok("msg_sender.c", 'B')) == -1) {
      /* same key as msg_sender.c */
        perror("ftok");
        exit(1);
    }
    if ((msqid = msgget(key, 0644)) == -1) { /* connect to the queue */
        perror("msgget");
        exit(1);
    }
    printf("spock: ready to receive messages, captain.\n");

    for(;;) { /* Spock never quits! */
        if (msgrcv(msqid, (struct msgbuf *)&buf, sizeof(buf), 0, 0) == -1) {
            perror("msgrcv");
            exit(1);
        }
        printf("spock: \"%s\"\n", buf.mtext);
    }

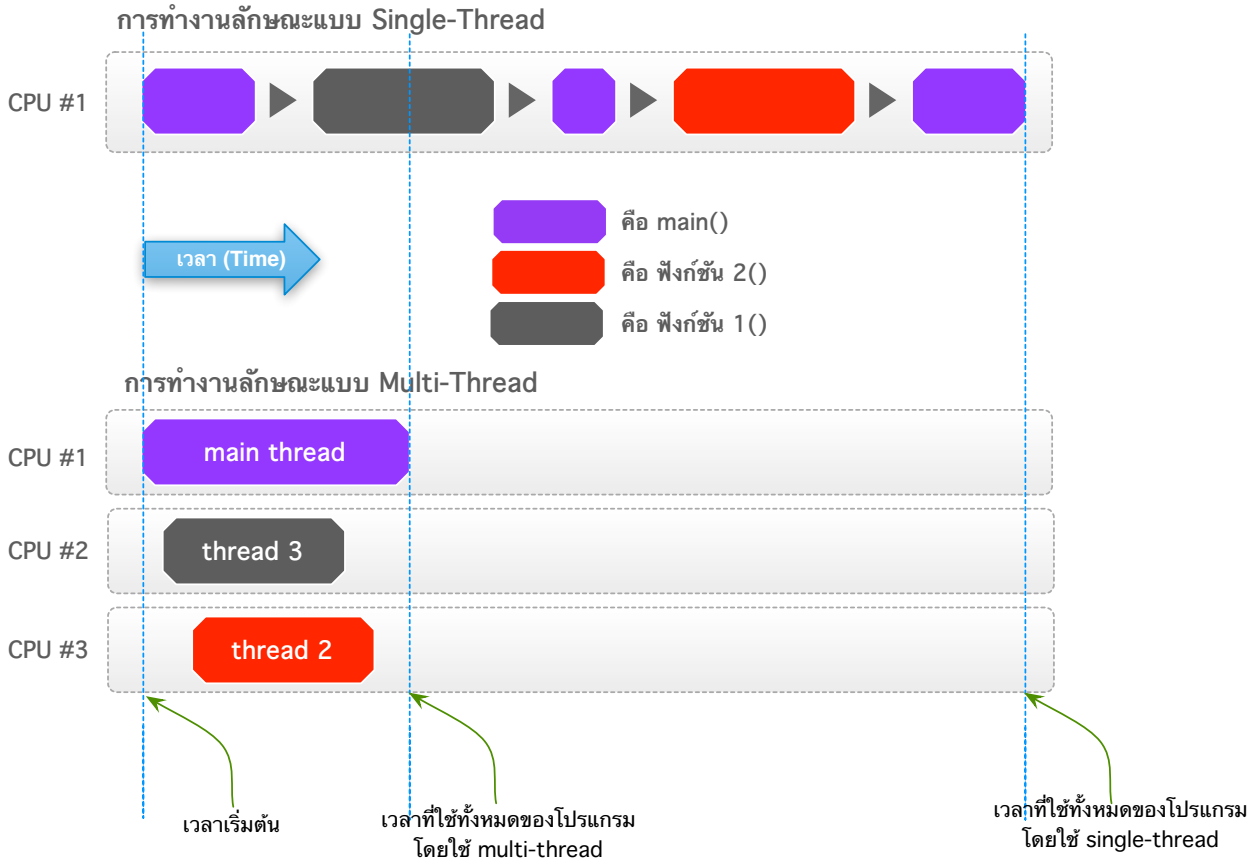
    return 0;
}
```

Thread

นักพัฒนาสามารถพัฒนาโปรแกรมในลักษณะ multi-tasking programming โดยการสร้างเทรด (thread) เพื่อเพิ่มประสิทธิภาพการทำงานของระบบปฏิบัติการลีนุกซ์ให้สามารถรองรับการทำงานแบบหลายงานได้ ซึ่งเทรดจะถูกเรียกได้อีกชื่อหนึ่งว่า “lightweight processes”

เนื่องจากโปรเซสทั่วไปภายในระบบปฏิบัติการลีนุกซ์ประกอบด้วยสถานะของหน่วยประมวลผล (เช่นค่าภายในตัวรีจิสเตอร์ AX,BX,DX), รายละเอียดของการจองหน่วยความจำ (สำหรับเก็บ code, globals, heap และ stack) และรายละเอียดที่เกี่ยวข้องกับระบบปฏิบัติการ (เช่น การเปิดไฟล์, หมายเลขโปรเซส) ซึ่งเทรดก็จะคล้ายกัน แต่จะต่างกันตรงที่ multiple processes จะแยกการเก็บสถานะกันอย่างชัดเจน ในขณะที่ multiple threads จะมีการใช้ code, globals และ heap ร่วมกัน

เทรดจะสามารถทำงานได้ดีในกรณีที่เครื่องคอมพิวเตอร์นั้นมีมากกว่าหนึ่งหน่วยประมวลผล โดยในรูปข้างล่างแสดงการทำงานของเทรดแต่ละตัว (ได้แก่ main(), function1() และ function2()) ซึ่งถ้าเครื่องมีหน่วยประมวลผลเพียงตัวเดียวการทำงานของโปรแกรมนี้ก็จะทำตามลำดับของคำสั่งแต่ละบรรทัด (program counter) ซึ่งอาจจะใช้เวลาทั้งสิ้น 2 นาที แต่ถ้าเครื่องมีหน่วยประมวลผลจำนวน 3 ตัวทั้งสามส่วนก็จะแยกกันทำงานไปแต่ละหน่วยประมวลผลซึ่งอาจจะใช้เวลาทั้งสิ้นเพียง 50 วินาทีเท่านั้น



รูปที่ 4.6 เปรียบเทียบการประมวลผลโปรแกรมแบบ Single-Thread และ Multi-Thread

การสื่อสารกันระหว่างเทรดจะสามารถติดต่อผ่านกันได้ทางหน่วยความจำที่ใช้งานร่วมกันอยู่ได้ทันที ในขณะที่การสื่อสารระหว่างโปรเซสจะสามารถติดต่อผ่านกันได้ต้องผ่านทางระบบปฏิบัติการเท่านั้น เช่น ผ่านไฟล์, ผ่านไปป์ หรือผ่านซ็อกเก็ต เป็นต้น

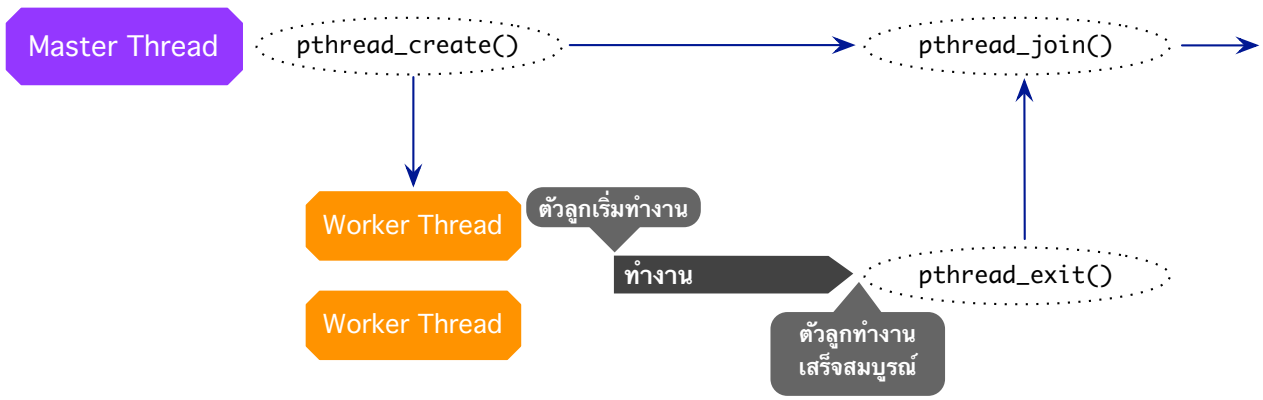
POSIX threads

ไลบรารี POSIX thread ถือว่าเป็นตัวมาตราฐานหลักในการเขียนเทรดสำหรับโปรแกรมภาษา C/C++ ซึ่งเป็นไลบรารีที่จัดเตรียมฟังก์ชันต่างๆเกี่ยวกับการจัดการเทรดและจะทำงานได้มีประสิทธิภาพมากสำหรับระบบคอมพิวเตอร์ที่มีหลายหน่วยประมวลผลกลาง (multi-processor หรือ multi-core systems) เนื่องจากมีฟังก์ชันในการจัดตารางการทำงานของโปรเซสบนแต่ละหน่วยประมวลผลกลางที่อยู่ภายในระบบคอมพิวเตอร์ตัวเดียวกัน และเทรดทุกตัวที่อยู่ภายในโปรเซสเดียวกันนั้นก็จะใช้พื้นที่ในหน่วยความจำเดียวกัน (address space) ซึ่งแตกต่างจากเทคโนโลยีในการเขียนโปรแกรมแบบขนาน (Parallel programming) เช่น MPI และ PVM ที่ถูกใช้ในสภาพแวดล้อมการคำนวณแบบกระจาย (distributed computing) ไปยังระบบคอมพิวเตอร์เครื่องอื่นๆ ที่อยู่ไกลออกไปหรือต่างสถานที่กัน

การพัฒนาโปรแกรมเทรดโดยใช้ไลบรารี Posix threads (pthread.h) จะมีฟังก์ชันที่สำคัญดังนี้

```
#include <pthread.h>
int pthread_create(pthread_t *new_thread_ID,
                   const pthread_attr_t *attr,
                   void * (*start_func)(void *),
                   void *arg);
int pthread_join(pthread_t target_thread, void **status);
```

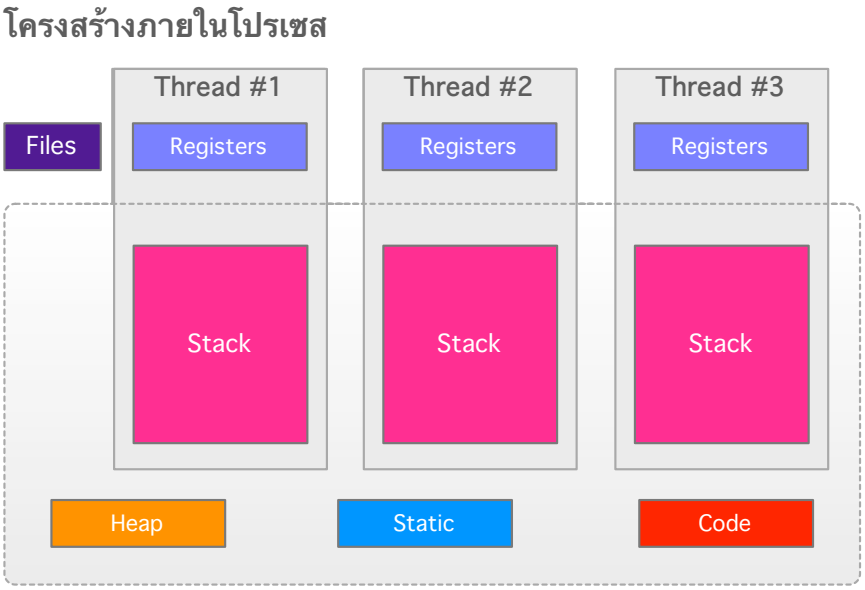
ขบวนการทำงานของเทรดนั้นจะประกอบไปด้วย thread creation, termination, thread synchronization (joins,blocking), scheduling, data management และ process interaction



รูปที่ 4.7 แสดงฟังก์ชันที่เกี่ยวข้องของการทำงานของเทรดลูกตั้งแต่เริ่มจนสิ้นสุด

พื้นที่ที่ใช้งานร่วมกันของเทรดทั้งหมดภายในโปรเซส จะมีประกอบไปด้วย

- ชุดคำสั่งโปรเซส (Process instructions)
- ค่า files descriptors ที่มีการเปิดไว้
- สัญญาณ (signals) และตัวดำเนินการ (signal handlers)
- ไดเรดทอรี่ปัจจุบัน (current working directory)
- หมายเลข User และ Group



รูปที่ 4.8 แสดงรายละเอียดของโครงสร้างภายในโปรเซส

โดยแต่ละเทรดจะมี:

- หมายเลขเทรด (Thread ID)
- กลุ่มตัวแปรรีจิสเตอร์ (set of registers) และ stack pointer
- สแต็กสำหรับเก็บค่าตัวแปร (local variables)
- signal mask
- ค่า priority
- ค่าสถานะที่ส่งกลับ: errno

และเมื่อต้องการคอมไพล์โปรแกรมจะต้องมีการอ้างอิงไลบรารี Posix threads ด้วย -lpthread ดังตัวอย่างข้างล่าง

```
$ gcc -o main main.c –lpthread
```

ตัวอย่างโปรแกรม simple_threads.c

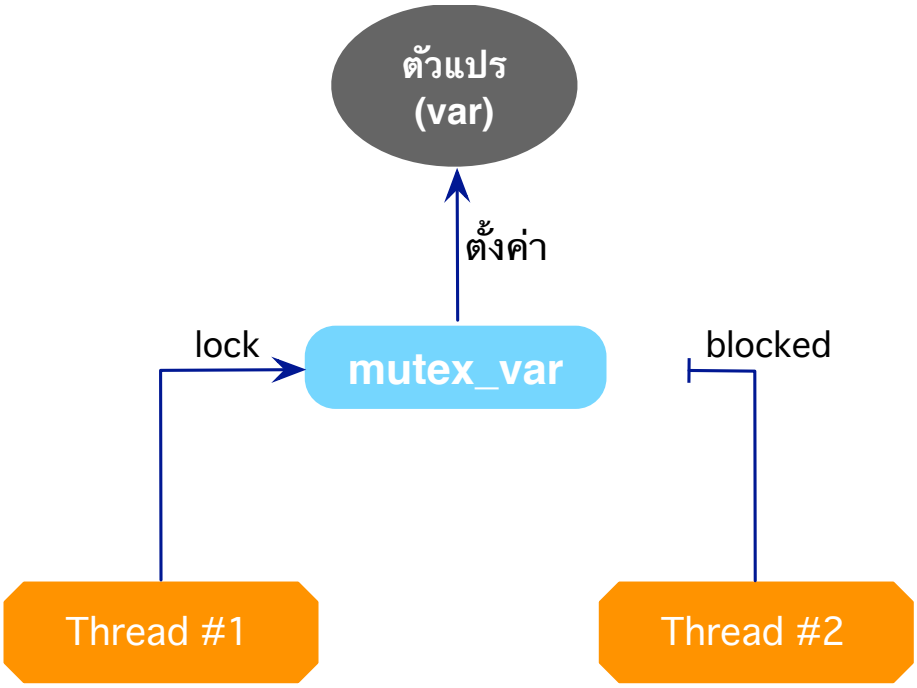
```
// simple_threads.c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void *print_message_function(void *ptr);
main()
{
      pthread_t thread1, thread2;
      const char *message1 = "Thread 1";
      const char *message2 = "Thread 2";
      int iret1, iret2;
/* Create independent threads each of which will execute function */
      iret1 = pthread_create( &thread1, NULL, print_message_function, (void*) mes-
sage1);
      iret2 = pthread_create( &thread2, NULL, print_message_function, (void*) mes-
sage2);
/* Wait till threads are complete before main continues. Unless we  */
/* wait we run the risk of executing an exit which will terminate   */
/* the process and all threads before the threads have completed.   */
      pthread_join( thread1, NULL);
      pthread_join( thread2, NULL);
      printf("Thread 1 returns: %d\n",iret1);
      printf("Thread 2 returns: %d\n",iret2);
      exit(0);
}
void *print_message_function(void *ptr) {
      char *message;
      message = (char *) ptr;
      printf("%s \n", message);
}
```

ทำการคอมไพล์ และทดสอบรันโปรแกรม

```
$ gcc -o simple_threads simple_threads.c –lpthread
$ ./simple_threads
Thread 2
Thread 1
Thread 1 returns: 0
Thread 2 returns: 0
```

Thread Synchronization

นักพัฒนาสามารถพัฒนาโปรแกรมโดยเรียกใช้ความสามารถในการจัดกลไกการทำงานประสานกันระหว่างเทรด (synchronization mechanisms) โดยไม่ให้เกิดปัญหาในกรณีที่เทรดแต่ละตัวจะต้องเข้าไปใช้ทรัพยากรตัวเดียวกัน ภายในเครื่องคอมพิวเตอร์เดียวกันโดยใช้ไลบรารี pthread



รูปที่ 4.9 แสดงการเข้าไปกำหนดค่าใน mutex ของเทรดที่ต้องการเข้าไปขอใช้ทรัพยากร

จากรูปด้านบนแสดงการอนุญาตเพียง thread #1 ตัวเดียวที่สามารถเข้าไปกำหนดค่า mutex (Mutual Exclusion Lock) เพื่อเข้าถึงทรัพยากรหรือตัวแปร โดยที่ thread #2 จะต้องรอ (block) คิวเพื่อเข้าถึงทรัพยากรหรือตัวแปรหลังจากที่ thread #1 ทำงานเสร็จเรียบร้อยแล้ว ตัว thread #2 จึงจะสามารถเข้าไปกำหนดค่า mutex เพื่อขอเข้าถึงทรัพยากรหรือตัวแปรนั้นต่อไป

Mutexes จะเป็นตัวการสำคัญในการช่วยป้องกันปัญหาข้อมูลเกิดความผิดพลาดหรือการไม่สอดคล้องกันของข้อมูล (data inconsistency) เนื่องจากสาเหตุการแย่งเข้าไปใช้ทรัพยากรส่วนกลางของเทรดหลายตัวพร้อมกัน เช่นหน่วยความจำกลาง (shared memory) หรือตัวแปรกลาง (global variable) ซึ่งเรียกพื้นที่ส่วนที่ถูกเข้าไปใช้งานนั้นว่า critical region แต่อย่างไรก็ตามการใช้ mutexes จะถูกใช้ได้เพียงในกรณีมีหลายเทรดในโปรเซสตัวเดียวกันเท่านั้น แต่ในกรณีที่มีหลายโปรเซสต้องการเข้าใช้ทรัพยากรเดียวกัน จะต้องแก้ไขโดยการใช้เทคนิค semaphores แทน mutexes

<table>
<tr><th colspan="2">กรณีไม่ใช้ MUTEX</th><th colspan="2">กรณีใช้ MUTEX</th></tr>
<tr><td colspan="2">

```
int counter=0;
void func()
{
   counter++;
}
```

</td><td colspan="2">

```
pthread_mutex_t  var = PTHREAD_MUTEX_INTIALIZER;
int counter=0;

void func()
{
   pthread_mutex_lock(&var);
   counter++;
   pthread_mutex_unlock(&var);
}
```

</td></tr>
<tr><th>Thread #1</th><th>Thread #2</th><th>Thread #1</th><th>Thread #2</th></tr>
<tr><td>counter=0</td><td>counter=0</td><td>counter=0</td><td>counter=0</td></tr>
<tr><td>counter=1</td><td>counter=1</td><td>counter=1</td><td>Thread #2 ถูกบล็อคไม่ให้เข้าถึงตัวแปร counter แต่ Thread #1 สามารถเข้าถึงตัวแปร counter ได้</td></tr>
<tr><td></td><td></td><td></td><td>counter=2</td></tr>
</table>

ซึ่งฟังก์ชันสำหรับ mutex ภายในไลบรารี pthread ได้แก่

- **`pthread_mutex_lock()`** ใช้สำหรับการเข้าไปล็อคค่าในตัวแปร mutex การเรียกฟังก์ชันนี้ก็จะถูกบล๊อคเอาไว้ไม่ให้เทรดอื่นเข้ามา จนกว่าเทรดตัวที่ใช้งานอยู่ทำการปล่อยตัว mutex เท่านั้น
- **`pthread_mutex_unlock()`** เพื่อปล่อยหรือปลดล็อคค่า mutex จากเทรดที่ใช้ mutex อยู่
- **`pthread_mutex_trylock()`** เพื่อใช้ตรวจสอบตัวแปร mutex ว่ากำลังถูกใช้ล๊อคใช้งานอยู่หรือไม่ ซึ่งจะมีประโยชน์ในการป้องกันไม่ให้เกิดเหตุการณ์ที่เทรดแต่ละตัวต่างฝ่ายต่างรอให้แต่ละตัวเสร็จ (deadlock conditions)

ตัวอย่างแสดงการสร้างเทรดจำนวน 10 เทรด โดยแต่ละตัวจะเข้าไปใช้งานตัวแปรกลางชื่อว่า counter แต่จะต้องเพิ่มค่า counter ได้เพียงครั้งละ 1 เทรดเท่านั้น

```
// simple_mutex.c
#include <stdio.h>
#include <pthread.h>
#define NTHREADS 10
void *thread_function(void *);
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
int counter = 0;

main()
{
      pthread_t thread_id[NTHREADS];
      int i, j;

      for(i=0; i < NTHREADS; i++)
```

```
	{
		pthread_create( &thread_id[i], NULL, thread_function, NULL );
	}

	for(j=0; j < NTHREADS; j++)
	{
		pthread_join( thread_id[j], NULL);
	}
// Now that all threads are complete I can print the final result.
// Without the join I could be printing a value before all the threads
// have been completed.

	printf("Final counter value: %d\n", counter);
}

void *thread_function(void *dummyPtr) {
	printf("Thread number %ld\n", pthread_self());
	pthread_mutex_lock(&mutex1);
	counter++;
	pthread_mutex_unlock(&mutex1);
}
```

คอมไพล์ และทดสอบรันโปรแกรม

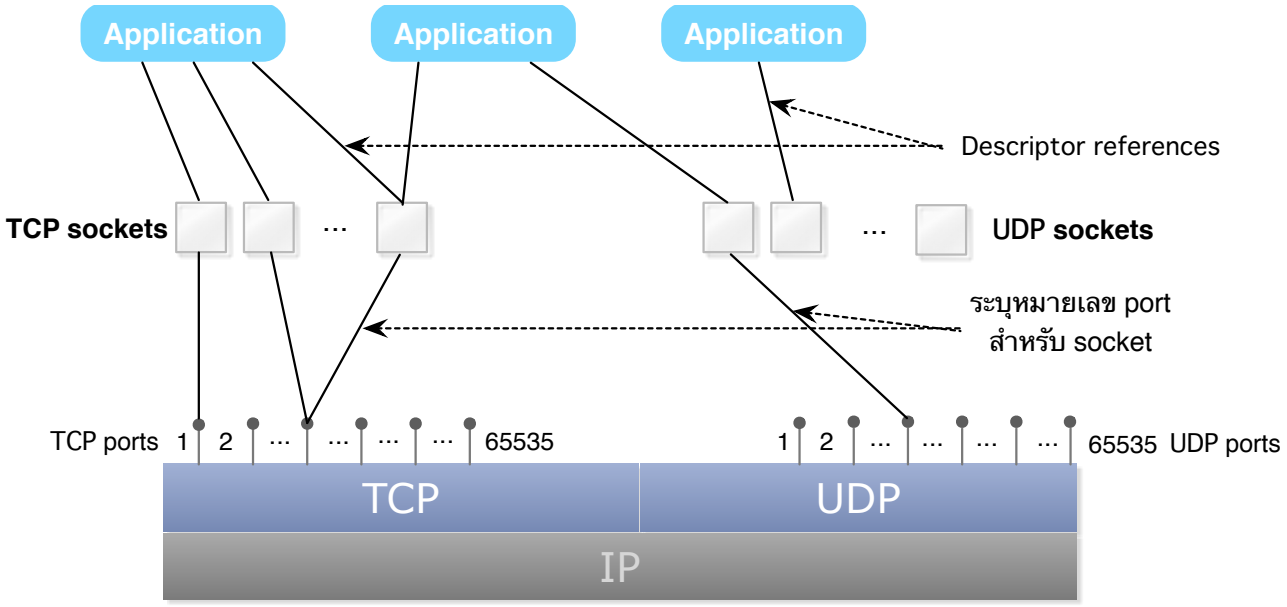
```
$ gcc -o simple_mutex simple_mutex.c –lpthread
$ ./simple_mutex
Thread number 139638417069824
Thread number 139638425462528
Thread number 139638442247936
Thread number 139638433855232
Thread number 139638408677120
Thread number 139638400284416
Thread number 139638391891712
Thread number 139638383499008
Thread number 139638375106304
Thread number 139638366713600
Final counter value: 10
```

การพัฒนาโปรแกรมสื่อสารบนระบบเครือข่าย

บอร์ดสมองกลฝังตัวส่วนใหญ่จะมีพอร์ตสำหรับเชื่อมต่อระบบเครือข่าย (Ethernet Port) เป็นพื้นฐาน รวมทั้งระบบปฏิบัติการลีนุกซ์แบบฝังตัวเองก็ได้จัดเตรียมไลบรารีและชุดโปรแกรมคำสั่งทางด้านระบบเครือข่ายเป็นพื้นฐานอยู่ภายในไว้ให้แล้วเช่นกัน ดังนั้นนักพัฒนาสามารถทำให้บอร์ดสมองกลฝังตัวติดต่อสื่อสารกับบอร์ดอื่นๆ คอมพิวเตอร์อื่นๆ หรือเครื่องแม่ข่ายที่อยู่ภายในระบบเครือข่ายเดียวกันได้ โดยบอร์ดสมองกลฝังตัวก็ยังสามารถบันทึกข้อมูลที่มีขนาดใหญ่ฝากไว้ในเครื่องคอมพิวเตอร์อื่นๆหรือ

เครื่องแม่ข่าย แทนที่จะบันทึกข้อมูลขนาดใหญ่นั้นเก็บไว้ในตัวเอง นอกจากนั้นผู้ใช้ก็สามารถเข้าถึงเพื่อควบคุมหรือดูสถานะการทำงานของบอร์ดสมองกลฝังตัวจากที่ไหนก็ได้

ดังนั้นการพัฒนาโปรแกรมจะต้องทำความเข้าใจวิธีการตั้งค่าหมายเลข IP Addresses แต่ละตัว ไม่ว่าจะเป็นหมายเลขเครือข่าย (Network ID) หมายเลขเครื่อง (Host ID) หมายเลข Netmask Address เมื่อโปรเซสทั้งสองที่อยู่ต่างเครื่องต้องการสื่อสารกัน จะต้องมีสถาปนาการเชื่อมต่อด้วยขั้นตอนตามเทคโนโลยี TCP/IP ซึ่งอยู่ในระดับ Transport Layer และจะต้องมีการระบุหมายเลขพอร์ต (Port Address) ไปยังโปรโตคอลแอพพิเคชั่น (Application Protocol) ที่โปรเซสแต่ละฝั่งใช้อยู่ด้วย ตัวอย่างเช่น โปรแกรมรับส่งไฟล์ที่ใช้โปรโตคอล FTP ในการกำหนดควบคุมวิธีการส่งไฟล์ระหว่างกัน เรียกการพัฒนาโปรแกรมทางด้านนี้ว่า Socket Programming



รูปที่ 4.10 แสดงการเชื่อมต่อระดับโปรแกรมประยุกต์และหมายเลขพอร์ตของ TCP/IP

ซ็อกเก็ต (socket) นั้นมีอยู่ด้วยกัน 3 ประเภทโดยยึดตามมาตราฐานของเทคโนโลยี TCP/IP ได้แก่

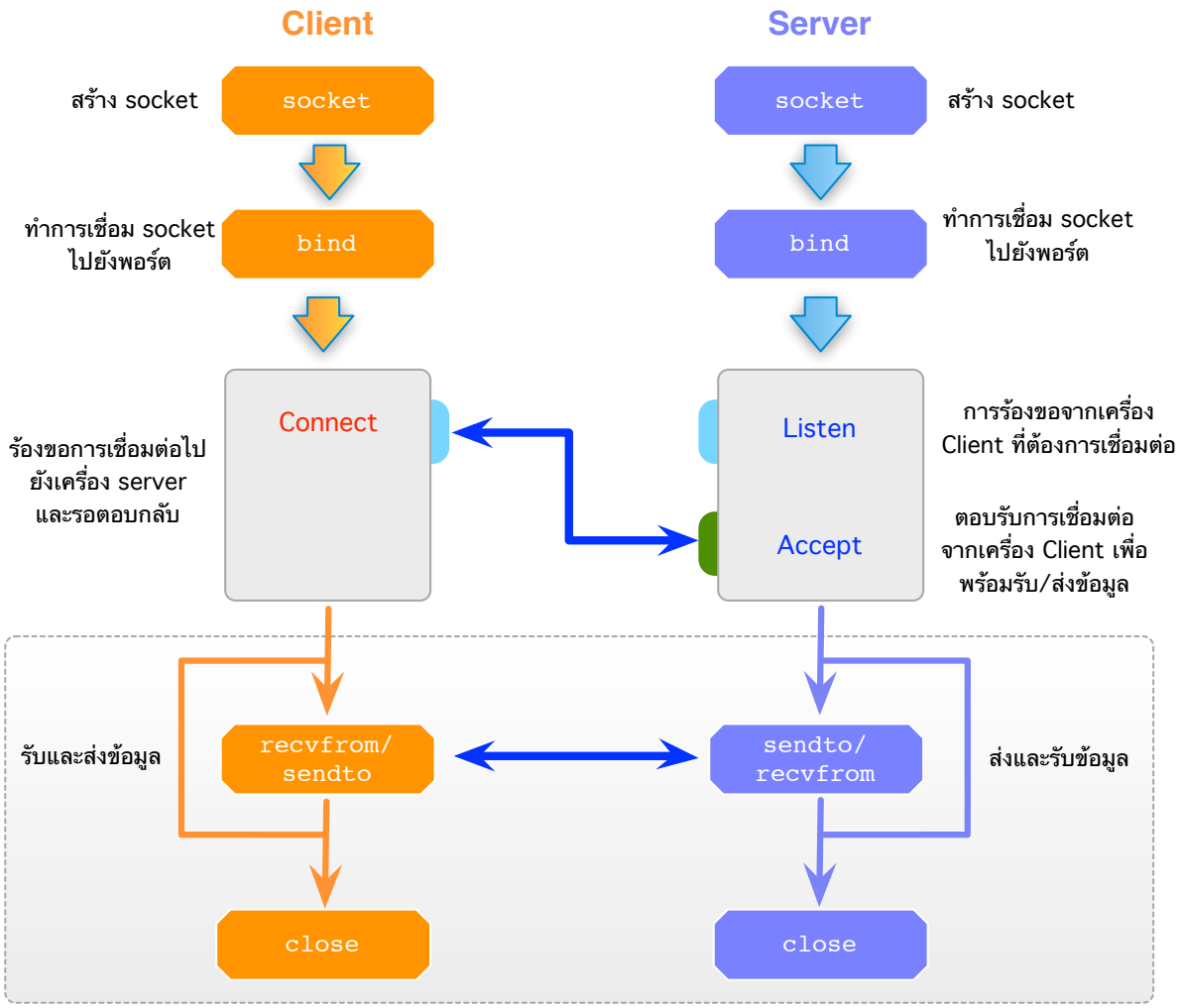
ตาราง 4.3 ประเภทของ Socket

SOCKET	คำอธิบาย
Datagram Socket	เรียกอีกชื่อหนึ่งว่า Connection less Socket ซึ่งใช้โปรโตคอล UDP (User Datagram Protocol) เป็นตัวกำหนดวิธีการสื่อสาร โดยข้อมูลหรือแพ็กเก็ต (packet) แต่ละตัวจะถูกส่งบน datagram socket ที่แยกเส้นทางกันออกไป ดังนั้นแพ็กเก็ตที่ส่งจากเครื่องต้นทางก็จะถูกลำเลียงกระจายออกไปในแต่ละเส้นทาง จนถึงเครื่องรับปลายทาง โดยแต่ละแพ็กเก็ตก็อาจจะมาถึงเครื่องปลายทางแบบไม่ได้เรียงตามลำดับตามที่ถูกส่งออกจากเครื่องต้นทางก่อนหน้านั้น

SOCKET	คำอธิบาย
Stream Socket	เรียกอีกชื่อหนึ่งว่า Connection-oriented Socket ซึ่งใช้โปรโตคอล TCP (Transport Control Protocol) เป็นตัวกำหนดวิธีการสื่อสาร ซึ่งจะมีการสถาปนาการเชื่อมต่อและการันตีการรับส่งแพ็กเก็ต ดังนั้นข้อมูลหรือแพ็กเก็ตแต่ละตัวจะถูกส่งบน stream socket และจะถูกลำเลียงส่งผ่านช่องทางที่ถูกสร้างขึ้นมานี้ไปยังเครื่องปลายทางจนครบถ้วนสมบูรณ์
Raw Socket	ส่วนใหญ่จะพบในอุปกรณ์เครือข่ายเช่น สวิทช์ (Switch) และ เราเตอร์ (Router) ซึ่งทำงานอยู่ในระดับ Internet Layer ที่มีการรับ-ส่งข้อมูลโดยไม่ได้มีการใช้โปรโตคอลเหมือน datagram และ stream socket ในการกำหนดมาตราฐานการสื่อสาร

ความสัมพันธ์ระหว่างโปรแกรม ซ็อกเก็ต โปรโตคอล และหมายเลขพอร์ตภายในเครื่องคอมพิวเตอร์ เพื่อใช้ในการสื่อสารข้อมูลระหว่างโปรเซสที่อยู่ต่างเครื่องกัน ประเด็นที่น่าสนใจที่นักพัฒนาควรรู้ได้แก่

- โปรแกรมหนึ่งโปรแกรมสามารถเปิดซ็อกเก็ตได้หลายซ็อกเก็ตเพื่อรับการเชื่อมต่อจากเครื่องภายนอกได้พร้อมกันในเวลาเดียวกัน
- โปรแกรมหลายโปรแกรมสามารถใช้ซ็อกเก็ตตัวเดียวกันในเวลาเดียวกันแต่ไม่ค่อยพบเห็นการใช้ในลักษณะนี้
- มากกว่าหนึ่งซ็อกเก็ตที่สามารถถูกเกี่ยวข้องและใช้งานพอร์ตอันเดียวกัน
- โปรแกรมประยุกต์แต่ละตัวจะมีการใช้ทั้ง TCP และ UDP เพื่อใช้ในการจัดการและรับส่งข้อมูลตามวัตถุประสงค์ที่แตกต่างกันไป เช่น ต้องการเน้นความเร็วในการส่งแม้ข้อมูลจะหายได้บ้างก็จะใช้ช่องทาง UDP หรือถ้าเน้นความถูกต้องของข้อมูลโดยที่ข้อมูลจะต้องไม่สูญหายก็จะใช้ช่องทาง TCP ในการรับส่งแทน
- ไฟล์ /etc/services ภายในระบบปฏิบัติการลีนุกซ์จะบอกรายละเอียดของโปรแกรมให้บริการและโปรโตคอลที่ใช้หมายเลขพอร์ตในการสื่อสารผ่านซ็อกเก็ตในระดับของ Transport layer



รูปที่ 4.11 แสดงขบวนการเชื่อมต่อระหว่างเครื่องภายในระบบเครือข่ายผ่าน Socket

ตัวอย่างโปรแกรมเชื่อมต่อผ่านซ็อกเก็ตตามขั้นตอนที่อธิบายในรูปข้างต้น ตัวลูก (Client) จะเชื่อมต่อไปยังเครื่องแม่ (Server) แล้วส่งข้อความ Hello World of Socket ไปยังเครื่องแม่ เมื่อเครื่องแม่ได้รับข้อความก็จะแสดงออกทางหน้าจอของเครื่องแม่ ดังนี้

โปรแกรมฝั่งเครื่องแม่ (Server):

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
	/* Variables */
	int sock;
	struct sockaddr_in server;
	int mysock;
	char buff[1024];
```

```
	int rval;

	/* Create socket */
	sock = socket(AF_INET, SOCK_STREAM, 0);
	if(sock < 0){
		perror("Failed to create socket");
		exit(1);
	}
	server.sin_family = AF_INET;
	server.sin_addr.s_addr = INADDR_ANY;
	server.sin_port = htons(5000);

	/* Call bind */
	if(bind(sock, (struct sockaddr *)&server, sizeof(server)))
	{
		perror("bind failed");
		exit(1);
	}

	/* Listen */
	listen(sock, 5);

	/* Accept */
	do {
		mysock = accept(sock, (struct sockaddr *) 0, 0);
		if(mysock == -1) {
			perror("accept failed");
		}else{
			memset(buff, 0, sizeof(buff));
			if((rval = recv(mysock, buff, sizeof(buff), 0)) < 0 )
				perror("reading stream message error");
			else if(rval == 0)
				printf("Ending connection\n");
			else
				printf("MSG: %s\n", buff);

			printf("Got the message (rval = %d)\n", rval);
			close(mysock);
		}
	}while(1);
	return 0;
}
```

โปรแกรมฝั่งเครื่องลูก (Client):

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
```

```c
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

#define DATA "Hello World of Socket"

int main(int argc, char *argv[])
{
	int sock;
	struct sockaddr_in server;
	struct hostent *hp;
	char buff[1024];
	sock = socket(AF_INET, SOCK_STREAM, 0);
	if(sock < 0)
	{
		perror("socket failed");
		exit(1);
	}

	server.sin_family = AF_INET;
	hp = gethostbyname(argv[1]);
	if(hp == 0)
	{
		perror("gethostbyname failed");
		close(sock);
		exit(1);
	}
	memcpy(&server.sin_addr, hp->h_addr, hp->h_length);
	server.sin_port = htons(5000);
	if(connect(sock, (struct sockaddr *) &server, sizeof(server)) < 0)
	{
		perror("connect failed");
		close(sock);
		exit(1);
	}

	if(send(sock, DATA, sizeof(DATA), 0) < 0 )
	{
		perror("send failed");
		close(sock);
		exit(1);
	}
	printf("Sent %s\n", DATA);
	close(sock);

	return 0;
}
```

คอมไพล์และรันโปรแกรมทั้งสองเครื่องดังนี้

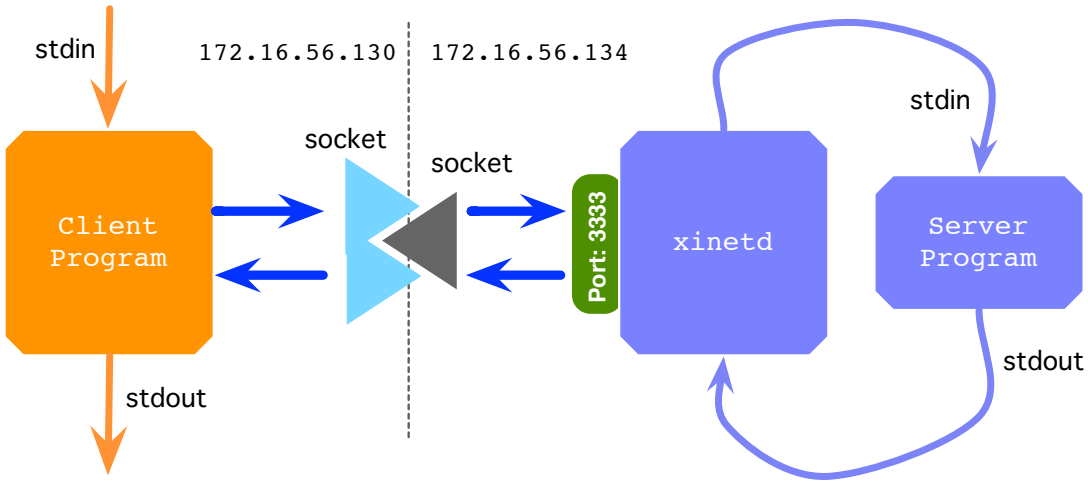
เครื่องแม่:

```
$ gcc -o server server.c
$ ./server
MSG: Hello World of Socket
Got the message (rval = 22)
```

เครื่องลูก:

```
$ gcc -o client client.c
$ ./client <IP เครื่องแม่>
Sent Hello World of Socket
```

ตัวอย่างถัดไปแสดงขั้นตอนการพัฒนาโปรแกรมบริการฝั่งเครื่องแม่ (Server) โดยมีการระบุชื่อโปรแกรมให้บริการในฝั่งของเครื่องแม่ข่ายอย่างชัดเจนภายในไฟล์ /etc/services ดังแสดงในรูปข้างล่าง



รูปที่ 4.12 แสดงขบวนการเชื่อมต่อเข้าสู่พอร์ต 3333 ของเครื่องแม่ข่าย

ขั้นตอนที่ 1 : สร้างโปรแกรมเชล์สคริปท์เพื่อเป็นโปรแกรมบริการฝั่งเครื่องแม่

ทำการสร้างไฟล์สคริปท์ เพื่อทำหน้าที่เป็นโปรแกรมฝั่งเครื่องแม่ข่าย

```
$ vim hello_server.sh
#!/bin/bash
/bin/echo -n "Hello World:" | /usr/bin/tee /tmp/log.log
/bin/date | /usr/bin/tee -a /tmp/log.log
```

เปลี่ยนสิทธิ์ให้ไฟล์สามารถถูกรันได้

```
$ chmod 755 hello_server.sh
$ ./hello_server.sh
Hello World:Mon Sep 30 01:44:24 PDT 2013
```

ขั้นตอนที่ 2 : กำหนดหมายเลขพอร์ตที่จะเปิดให้บริการ เพิ่มเติมในไฟล์ /etc/services

ทำการตรวจสอบหมายเลขพอร์ต 3333 ในไฟล์ /etc/services และการเช็คจากการเปิดบริการของโปรแกรม xinetd ก่อนว่าได้มีการเรียกใช้หรือไม่ ดังนี้

```
$ cat /etc/services | grep 3333
$ grep -r 3333 /etc/xinetd.d/*
```

เมื่อตรวจสอบแล้ว ปรากฏว่ายังไม่มีการเรียกใช้หมายเลขพอร์ต 3333 และชื่อบริการ “hello” ก็สามารถเข้าไปเพิ่มชื่อบริการใหม่ในไฟล์ /etc/services ดังคำสั่งข้างล่างนี้

```
$ sudo vim /etc/services
```

ทำการเพิ่มบรรทัดดังข้างล่างนี้ แล้วบันทึกไฟล์ /etc/services ให้เรียบร้อย

```
hello               3333/tcp        # sample hello service
```

หมายเหตุ:

- ให้ลบบรรทัดที่เพิ่มเข้าไปใน /etc/services ออกเมื่อได้ทดสอบ hello service แล้ว เพื่อไม่ให้เกิดปัญหากับการทำงานของโปรแกรม xinetd ในอนาคต

ขั้นตอนที่ 3 : ระบุรายละเอียดของ hello service เพิ่มเติมในไดเรกทอรี /etc/xinetd.d/

```
$ sudo vim /etc/xinetd.d/hello
```

เพิ่มบรรทัดดังข้างล่างนี้

```
# default: on
# description: Hello World socket server
service hello
{
      port  = 3333
      disable    = no
      socket_type      = stream
      protocol   = tcp
      user  = root
      wait  = no
      server     = /home/student/hello_server.sh
      log_on_success   += USERID
      log_on_failure   += USERID
}
```

ทำการเรียกโปรแกรม xinetd ใหม่ เพื่อให้อ่านการตั้งค่าระบบใหม่

```
$ sudo /etc/init.d/xinetd restart
 * Stopping internet superserver xinetd                          [ OK ]
 * Starting internet superserver xinetd                          [ OK ]
```

ขั้นตอนที่ 4 : ทดสอบการทำงานโดยการเรียกใช้โปรแกรม telnet

```
$ telnet 172.16.56.134 3333
Trying 172.16.56.134...
Connected to 172.16.56.134.
Escape character is '^]'.
Hello World:Mon Sep 30 02:30:39 PDT 2013
Connection closed by foreign host.
```

การพัฒนาโปรแกรมเก็บข้อมูลด้วย SQLite

การเก็บข้อมูลในระบบสมองกลฝังตัวมีด้วยกันหลายวิธี แต่ปัญหาที่เกิดขึ้นบ่อยในการพัฒนาระบบเก็บข้อมูลภายในบอร์ดสมองกลฝังตัวที่มีระบบปฏิบัติการลีนุกซ์ทำงานอยู่ คือถ้าเก็บข้อมูล log ในลักษณะ text file ธรรมดา ในกรณีที่ต้องการลบหรือแทรกข้อมูลในตำแหน่งที่ระบุไว้ในไฟล์ log จะมีโอกาสทำให้ไฟล์เสียหายได้สูง รวมทั้งการเขียนโปรแกรมเพื่อเข้าไปจัดการข้อมูลภายในไฟล์ log ก็จะมีความซับซ้อนมากยิ่งขึ้น

ดังนั้นทางออกสำหรับการเก็บข้อมูลให้มีระบบ และสามารถเข้าถึงได้อย่างมีประสิทธิภาพ ควรจะต้องให้อยู่ในลักษณะการเก็บข้อมูลแบบเดียวกันโครงสร้างฐานข้อมูล (Database) โปรแกรมตัวหนึ่งที่ได้รับความนิยมสูง และเป็นโปรแกรมที่กลายเป็นตัวมาตราฐานในการเก็บข้อมูลภายในอุปกรณ์สมองกลฝังตัวในปัจจุบันนี้ไม่ว่าจะเป็นแอนดรอยด์, iOS เป็นต้น ก็คือโปรแกรม SQLite ซึ่งเป็นโปรแกรมประเภท Open Source มีขนาดเล็ก รองรับหลายสถาปัตยกรรมโดยจะมีสกุลไฟล์ที่เก็บเป็น .db, .sdb, หรือ .s3db เป็นต้น

ขั้นตอนข้างล่างนี้แสดงตัวอย่างการติดตั้งโปรแกรม SQLite ลงในเครื่อง Host เพื่อเตรียมสภาพแวดล้อมในการพัฒนาโปรแกรมภาษา C/C++ ที่ใช้การเก็บข้อมูลด้วยไลบรารีของ SQLite ภายในระบบปฏิบัติการ Embedded Linux ที่จะถูกนำไปใช้ในบอร์ดสมองฝังตัวต่อไป

ขั้นตอนที่ 1 : ดาวน์โหลดโปรแกรม SQLite เวอร์ชันล่าสุด

ดาวน์โหลดโปรแกรม SQLite (sqlite-autoconf-3080002.tar.gz) จากลิงค์

http://www.sqlite.org/download.html

```
$ tar -xzvf sqlite-autoconf-3080002.tar.gz
$ cd sqlite-autoconf-3080002/
```

ขั้นตอนที่ 2 : ติดตั้งตัว cross compiler ลงใน Ubuntu

ในกรณีที่ใช้ Ubuntu เวอร์ชัน 10.10 ขึ้นไป

```
$ sudo apt-get install gcc-arm-linux-gnueabi.
```

ในกรณีที่ใช้ Ubuntu เวอร์ชันต่ำกว่า 10.10

```
$ sudo add-apt-repository ppa:linaro-maintainers/toolchain
$ sudo apt-get update
$ sudo apt-get install gcc-arm-linux-gnueabi
```

ขั้นตอนที่ 3 : ทำการคอมไพล์โปรแกรม SQLite เพื่อให้รองรับสถาปัตยกรรม ARM

```
$ mkdir ~/sqlite
$ ./configure --host=arm CC=arm-linux-gnueabi-gcc AR=arm-linux-
-gnueabi-ar STRIP=arm-linux-gnueabi-strip RANLIB=arm-linux-gnueabi-
-ranlib CFLAGS=" -mfpu=vfp -Os -lpthread -lrt" LDFLAGS=${LDFLAGS}
$ make
$ make install DESTDIR=~/sqlite
```

ไดเรกทอรีจากการคอมไพล์จะถูกติดตั้งอยู่ภายใต้ไดเรกทอรี ~/sqlite/ ดังนี้

```
usr/local/bin
usr/local/include
usr/local/lib
usr/local/share
```

ขั้นตอนที่ 4 : สร้างโปรแกรมที่เรียกใช้ไลบรารี SQLite และทำการคอมไพล์ด้วย cross compiler เพื่อให้รองรับสถาปัตยกรรม ARM

```
$ vim my_sqlite.c
```

```c
#include <stdio.h>
#include <sqlite3.h>

int main(int argc, char* argv[])
{
   sqlite3 *db;
   char *zErrMsg = 0;
   int rc;

   rc = sqlite3_open("test.db", &db);

   if( rc ){
      fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
      exit(0);
   }else{
      fprintf(stderr, "Opened database successfully\n");
   }
   sqlite3_close(db);
}
```

ทำการคอมไพล์โปรแกรมดังคำสั่งข้างล่างนี้

```
$ arm-linux-gnueabi-gcc -o my_sqlite my_sqlite.c -I~/sqlite/usr/local/
include e -L~/sqlite/usr/local/lib -lsqlite3 -lpthread -ldl
```

นักพัฒนาสามารถใช้ตัวเลือก "-lpthread" หรือ "-pthread" ก็ได้ แต่ถ้าใช้ "-pthread" ตัวคอมไพล์เลอร์ก็จะค้นหาตัวไลบรารี Posix Thread ที่เหมาะสมให้อัตโนมัติ
สำหรับตัวเลือก "-ldl" เป็นการระบุว่าใช่ไลบรารีที่บรรจุฟังก์ชัน dlsym, dlopen ที่มีใช้อยู่ในโปรแกรม

```
$ file my_sqlite
my_sqlite: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically
linked (uses shared libs), for GNU/Linux 2.6.31, not stripped
```

การดีบักโปรแกรมภาษา C/C++

การตรวจสอบการทำงานของโปรแกรมเพื่อหาความผิดพลาดหรือแม้กระทั่งตรวจสอบขั้นตอนการทำงานของโปรแกรมว่าเป็นไปตามเงื่อนไขที่ได้ออกแบบไว้หรือไม่นั้นก่อนที่จะปล่อยโปรแกรมไปอยู่ในระบบสมองกลฝังตัวเพื่อให้ทำงานตามที่ได้กำหนด จำเป็นอย่างยิ่งที่นักพัฒนาจะต้องตรวจสอบการทำงานของโปรแกรมและดูค่าภายในตัวแปรหรือภายในหน่วยความจำที่เกิดขึ้นในระหว่างโปรแกรมกำลังทำงานอยู่ (run-time) ซึ่งเรียกขบวนการนี้ว่าการดีบัก (Debugging)

สำหรับระบบปฏิบัติการลีนุกซ์นั้น ก็ได้เตรียมเครื่องมือสำหรับการดีบัก ซึ่งเรียกว่า GNU debugger ถูกพัฒนาโดย Richard Stallman บิดาแห่ง GPL ในปี ค.ศ. 1986 ซึ่งเป็นส่วนหนึ่งในระบบ GNU ของ Richard Stallman โดยการใช้กลุ่มคำสั่ง gdb kdb หรือ ddd เป็นต้น คำสั่ง gdb สามารถช่วยนักพัฒนาในการดีบักโปรแกรม ดังรายละเอียดข้างล่างนี้:

- เมื่อโปรแกรมเริ่มทำงานก็สามารถสั่งให้ทำงานตามคำสั่งบรรทัดต่อบรรทัด (line by line) ด้วยชุดคำสั่ง gdb
- สามารถทำให้โปรแกรมหยุดตามเงื่อนไขที่ระบุไว้ได้
- แสดงค่าของตัวแปร ณ ขณะนั้นในระหว่างที่ทำงานอยู่ (run-time)
- กำหนดจุดหยุดโปรแกรม (breakpoints) แล้วจึงตรวจสอบการทำงานแต่ละขั้นตอนแบบ step-by-step หรือสั่งให้โปรแกรมทำงานตามปกติต่อไปได้

โดยก่อนที่จะเริ่มการดีบักโปรแกรมนักพัฒนาจะต้องระบุตัวเลือก “-g” ต้องท้ายคำสั่ง gcc ในขณะที่กำลังคอมไพล์โปรแกรม ดังตัวอย่างข้างล่าง

```
$ gcc –g –c main.c
$ g++ -g –c function1.c
$ g++ -g –o my_program main.o function1.o
```

คอมไพลเลอร์จะเพิ่มข้อมูลบางอย่างที่จำเป็นต่อการทำงานของ gdb ลงไปในออปเจคไฟล์ (.o) และโปรแกรมที่ได้จากการคอมไพล์ (executable file) ซึ่ง gdb จะใช้ข้อมูลเหล่านั้นเพื่อคำนวณหาบรรทัดของโค้ดโปรแกรมเพื่อแสดงข้อมูลต่างๆออกมา คำสั่งพื้นฐานของ gdb ที่นิยมใช้กันได้แก่

ตาราง 4.4 ตัวอย่างคำสั่งการดีบักโปรแกรมด้วยโปรแกรม gdb

คำสั่งเต็ม	คำสั่งย่อ	คำอธิบาย
run	r	เริ่มโปรแกรม
kill	k	หยุดโปรแกรม
quit	q	ออกจาก GDB
continue	c	ทำงานต่อโดยหยุดที่ breakpoint ถัดไป
disassemble	disass	แสดง assembly code ของ function ที่ EIP อยู่
disassemble ADDR	disass ADDR	แสดง assembly code ที่ address ADDR (ใช้ชื่อ function ได้)
disassemble ADDR1 ADDR2	disass ADDR1 ADDR2	แสดง assembly code ที่ address ADDR1 ถึง ADDR2
info breakpoints	i b	แสดง breakpoint ทั้งหมด
info registers	i r	แสดงค่าของ CPU registers ทั้งหมด
info frame	i f	แสดงข้อมูลเกี่ยวกับ stack frame ปัจจุบัน
backtrace	bt	แสดง call stack
break *ADDR	b *ADDR	set breakpoint ที่ address ADDR (ถ้าใช้ชื่อ function ไม่ต้องมี *)
enable [NUM]	en [NUM]	enable breakpoint หมายเลขที่ NUM
disable [NUM]	dis [NUM]	disable breakpoint หมายเลขที่ NUM
delete [NUM]	d [NUM]	delete breakpoint หมายเลขที่ NUM
delete	d	delete breakpoint ทั้งหมด
nexti [num]	ni [num]	ทำงานคำสั่งถัดไป ไม่เข้าไปใน call
stepi [num]	si [num]	ทำงานคำสั่งถัดไป เข้าไปใน call
x/NFU ADDR		แสดงค่าของ address ADDR โดย N คือจำนวนที่จะแสดงผล F คือรูปแบบที่จะแสดงผล (ดูตารางถัดไป) U คือจำนวน byte มี b (byte), h (2 bytes), w (4 bytes), g (8 bytes)
display/F ADDR	disp/F ADDR	แสดงค่าของ address ADDR ทุกครั้งที่ถึงหยุดทำงานชั่วคราว
display	disp	แสดงค่าที่อยู่ใน display list ทั้งหมด

คำสั่งเต็ม	คำสั่งย่อ	คำอธิบาย
undisplay [NUM]	und [NUM]	ลบ display ที่เก็บไว้ที่ NUM
set ADDR=VAL		set ค่า VAL ไปที่ address ADDR

ตัวอย่างถัดไปเป็นการทดสอบการดีบักโปรแกรม factorial โดยเริ่มต้นให้ทำการสร้างโปรแกรมดังนี้

```
$ vim factorial.c
# include <stdio.h>
int main()
{
      int i=0, num=0, j=1;
      printf ("Enter the number: ");
      scanf ("%d", &num );

      for (i=1; i<num; i++)
            j=j*i;
      printf("The factorial of %d is %d\n",num,j);
}
```

ทำการคอมไพล์โปรแกรม factorial โดยใส่ตัวเลือก -g ดังแสดงในคำสั่งข้างล่างนี้

```
$ gcc -g factorial.c
$ ./a.out
Enter the number: 10
The factorial of 10 is 362880
```

เรียกโปรแกรม gdb ด้วยคำสั่ง

```
$ gdb ./a.out
GNU gdb (Ubuntu/Linaro 7.2-1ubuntu11~ppa1) 7.2
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/student/unix/a.out...done.
```

ทดสอบการใช้คำสั่งเพื่อตั้งตำแหน่ง breakpoint และ clear breakpoint ซึ่งมีรูปแบบการใช้งานได้ 2 แบบคือแบบที่ 1 หยุดในตำแหน่งบรรทัดที่ระบุ

```
break [file_name]:line_number
```

```
clear [file_name]:line_number
```

แบบที่ 2 คือหยุดในตำแหน่งชื่อฟังก์ชันที่ระบุ

```
break [file_name]:func_name
clear [file_name]:func_name
```

ตัวอย่างการใช้คำสั่งดีบักโปรแกรม

```
(gdb) break 3
Breakpoint 2 at 0x40059c: file factorial.c, line 3.
(gdb) break 7
Breakpoint 2 at 0x4005c3: file factorial.c, line 7.
(gdb) info break
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000000000040059c in main at factorial.c:3
2       breakpoint     keep y   0x00000000004005c3 in main at factorial.c:10
(gdb) run
Starting program: /home/student/unix/a.out
Breakpoint 2, main () at factorial.c:7
7           scanf ("%d", &num );
```

สามารถใช้คำสั่ง print [variables] เพื่อแสดงค่าภายในตัวแปรขณะที่โปรแกรมกำลังทำงานอยู่ได้

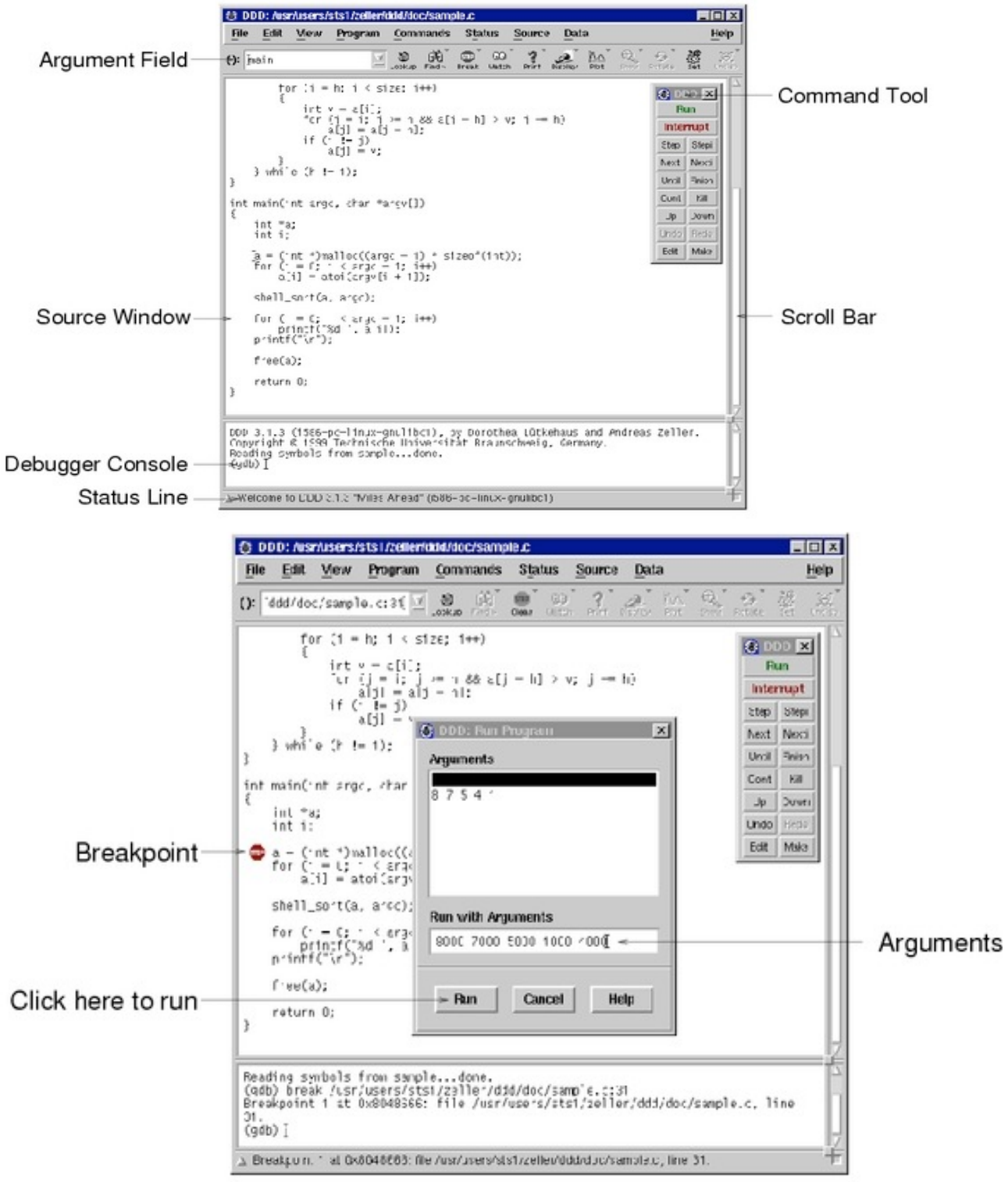
```
(gdb) print j
$1 = 1
(gdb) s
Enter the number: 3
9           for (i=1; i<num; i++)
(gdb) s
10              j=j*i;
(gdb) s
9           for (i=1; i<num; i++)
(gdb) s
10              j=j*i;
(gdb) print j
$2 = 1
(gdb) print i
$3 = 2
```

```
(gdb) break 10 if j==2
Note: breakpoint 1 also set at pc 0x4005e5.
Breakpoint 3 at 0x4005e5: file factorial.c, line 10.
(gdb) info break
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x00000000004005e5 in main at factorial.c:10
2       breakpoint     keep y   0x000000000040059c in main at factorial.c:3
3       breakpoint     keep y   0x00000000004005e5 in main at factorial.c:10
      stop only if j==2

(gdb) watch j==3
Hardware watchpoint 2: j == 3

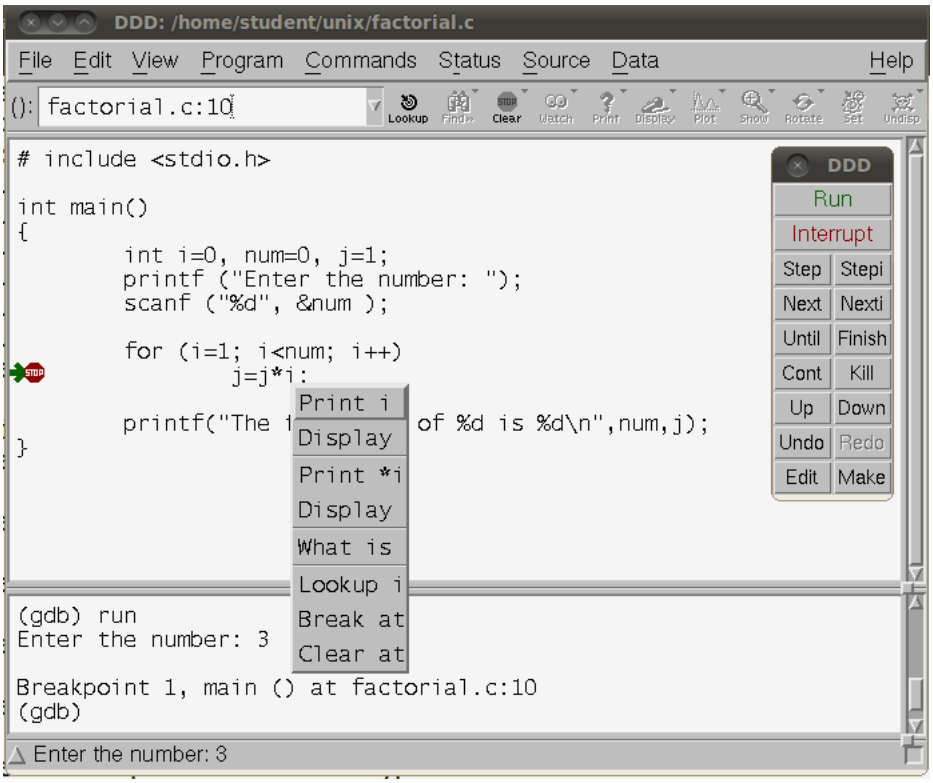
(gdb) disass main
Dump of assembler code for function main:
   0x0000000000400594 <+0>:push   %rbp
   0x0000000000400595 <+1>:mov    %rsp,%rbp
   0x0000000000400598 <+4>:sub $0x10,%rsp
   0x000000000040059c <+8>:movl$0x0,-0x4(%rbp)
   0x00000000004005a3 <+15>:     movl$0x0,-0x8(%rbp)
   0x00000000004005aa <+22>:     movl$0x1,-0xc(%rbp)
   0x00000000004005b1 <+29>:     mov $0x40070c,%eax
   0x00000000004005b6 <+34>:     mov    %rax,%rdi
   0x00000000004005b9 <+37>:     mov $0x0,%eax
   0x00000000004005be <+42>:     callq  0x400478 <printf@plt>
   0x00000000004005c3 <+47>:     mov $0x40071f,%eax
   0x00000000004005c8 <+52>:     lea    -0x8(%rbp),%rdx
   0x00000000004005cc <+56>:     mov    %rdx,%rsi
   0x00000000004005cf <+59>:     mov    %rax,%rdi
   0x00000000004005d2 <+62>:     mov $0x0,%eax
   0x00000000004005d7 <+67>:     callq  0x400498 <__isoc99_scanf@plt>
   0x00000000004005dc <+72>:     movl$0x1,-0x4(%rbp)
   0x00000000004005e3 <+79>:     jmp    0x4005f3 <main+95>
=> 0x00000000004005e5 <+81>:     mov    -0xc(%rbp),%eax
   0x00000000004005e8 <+84>:     imul   -0x4(%rbp),%eax
   0x00000000004005ec <+88>:     mov    %eax,-0xc(%rbp)
   0x00000000004005ef <+91>:     addl$0x1,-0x4(%rbp)
...
   0x0000000000400608 <+116>:    mov    %rax,%rdi
   0x000000000040060b <+119>:    mov $0x0,%eax
   0x0000000000400610 <+124>:    callq  0x400478 <printf@plt>
   0x0000000000400615 <+129>:    leaveq
   0x0000000000400616 <+130>:    retq
End of assembler dump.
```

ในกรณีที่ใช้โปรแกรม ddd ซึ่งเป็นโปรแกรมดีบักในรูปแบบกราฟฟิกซึ่งง่ายต่อการใช้งาน เพราะมีไอคอนที่สามารถอ้างอิงถึงคำสั่งต่างๆภายใน gdb ได้ทั้งหมดดังตัวอย่างหน้าตาของโปรแกรมในรูปข้างล่าง



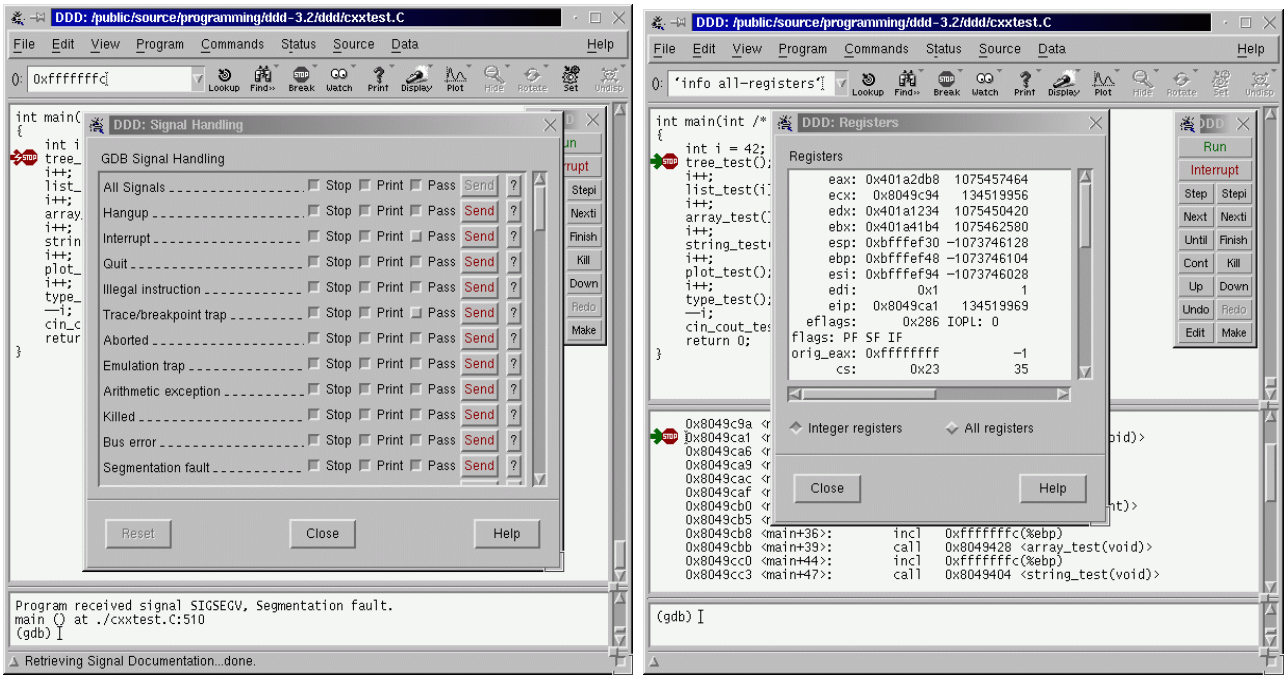
รูปที่ 4.13 หน้าตาโปรแกรม DDD และองค์ประกอบเครื่องมือต่างๆ

การตั้งตำแหน่ง breakpoint และดูค่าภายในตัวแปร สามารถทำได้ง่าย ดังตัวอย่างข้างล่าง



รูปที่ 4.14 ตัวอย่างการตั้งจุด Break Point

นอกจากนั้นสามารถส่งสัญญาณ (Signal) แบบต่างๆ และดูโค้ดโปรแกรมในรูปแบบภาษาแอสเซมบลี้ได้เช่นกัน

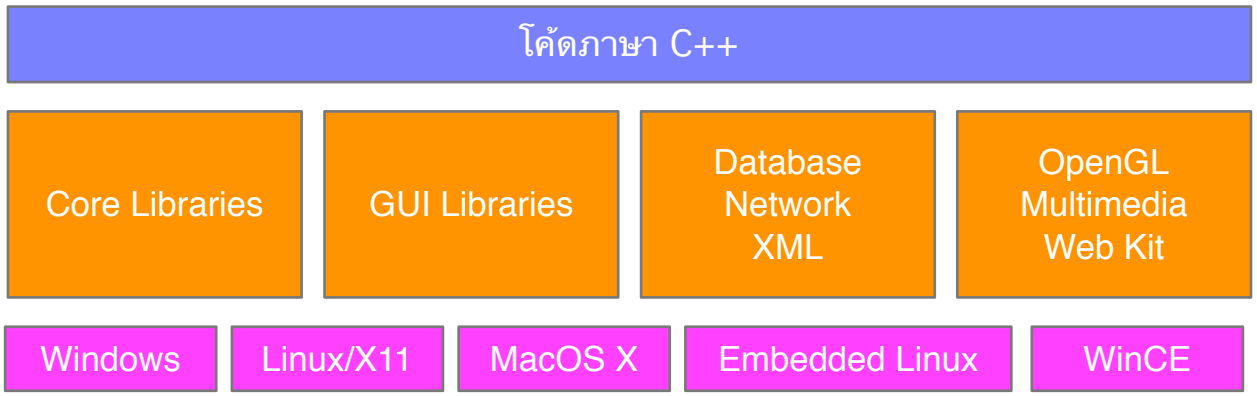


รูปที่ 4.15 หน้าต่างโปรแกรมในส่วนของการตั้งค่าในการส่งสัญญาณและดูภาษาแอสเซมบลี้

การเขียนโปรแกรมภาษา C++ ด้วย Qt

โปรแกรม Qt ได้รับการยอมรับว่าเป็น cross-platform application framework ที่ดีที่สุดตัวหนึ่งในปัจจุบันนี้ ซึ่งมีการเตรียมไลบรารีและฟังก์ชัน API จำนวนมากสำหรับการเขียนโปรแกรมภาษาซีพลัสพลัส ซึ่งโปรแกรม Qt ได้มีการปรับปรุงการทำงานหลายๆด้านภายในไลบรารีของภาษาซีพลัสพลัสเอง ตัวอย่างเช่น การจัดการหน่วยความจำ การติดต่อกับพอร์ตการสื่อสาร การจัดการโปรเซสและเทรด ให้มีความยืดหยุ่น มีความเสถียรภาพ และมีประสิทธิภาพสูงมากขึ้น เพื่อให้เหมาะกับสถาปัตยกรรมที่แตกต่างกันไป

Qt จะช่วยให้นักพัฒนาสามารถออกแบบและเขียนโปรแกรมง่ายขึ้นในการพัฒนาโปรแกรมประยุกต์ด้านต่างๆ ตัวอย่างเช่น งานด้านกราฟฟิก งานด้านระบบเครือข่าย งานด้านระบบฐานข้อมูล งานด้านมัลติมีเดีย การจัดการไฟล์ XML หรืองานด้านเว็บโปรแกรม เป็นต้น นอกจากนั้นโปรแกรมที่พัฒนาขึ้นมาภายใต้ Qt จะสามารถคอมไพล์และถูกรองรับให้สามารถทำงานได้หลายสถาปัตยกรรมและหลายแพลตฟอร์ม ได้แก่ ระบบปฏิบัติการลีนุกซ์ ระบบปฏิบัติการ MacOS X ระบบปฏิบัติการวินโดส์ ระบบปฏิบัติการวินโดส์ซีอี ระบบปฏิบัติการมีโก้ และระบบปฏิบัติการลีนุกซ์สำหรับระบบสมองกลฝังตัว



รูปที่ 4.16 โครงสร้างสถาปัตยกรรม Qt framework

จากรูปข้างบนแสดงโครงสร้างของ Qt framework โดยชั้นบนสุดคือโค้ดโปรแกรมภาษาซีพลัสพลัส ชั้นรองลงมาจะเป็นส่วนของ Qt classes สำหรับสร้าง GUI, Webkit, Database และอื่นๆ และชั้นสุดท้ายคือระบบปฏิบัติการที่รองรับการทำงานของโปรแกรม

การติดตั้ง Qt สำหรับแต่ละระบบปฏิบัติการ

ติดตั้ง Qt สำหรับแต่ละระบบปฏิบัติการ จะมีการเตรียมชุดโปรแกรมสำหรับการพัฒนาที่เรียกว่า Qt Installer ซึ่งได้รวม Qt Libraries และ Qt Creator สำหรับเป็นโปรแกรมเพื่อให้พัฒนาโค้ดโปรแกรมและกราฟฟิกที่ใช้ในโปรแกรม โดยการเข้าไปดาวน์โหลดได้จาก http://qt-project.org/downloads หลังจากนั้นก็สามารถติดตั้งได้ทันที

สำหรับระบบปฏิบัติการ Embedded Linux

ทำการดาวน์โหลด Qt source code ลงเครื่องคอมพิวเตอร์ และแตกไฟล์ ดังคำสั่งข้างล่างนี้

```
$ wget
http://download.qt-project.org/official_releases/qt/4.8/4.8.5/qt-every
where-opensource-src-4.8.5.tar.gz
$ tar -xzvf qt-everywhere-opensource-src-4.8.5.tar.gz
```

กำหนดการตั้งค่า เพื่อให้รองรับสถาปัตยกรรมที่ต้องการ เช่น ARM แล้วทำการคอมไพล์ตามลำดับ

```
$ cd ~/qt-everywhere-opensource-src-4.8.5
$ ./configure -embedded [architecture]
```

โดยที่:

[architecture] คือสถาปัตยกรรมของบอร์ด target เช่น สถาปัตยกรรม arm, x86 หรือ mips

หรือสามารถระบุรายละเอียดพิเศษอื่นๆในระหว่างการใช้คำสั่ง ./configure เพื่อให้เหมาะสมกับสภาพแวดล้อมของสถาปัตยกรรมที่ใช้งาน ดังคำสั่งข้างล่างนี้

```
$ sudo /usr/local/qt-embedded-4.8.5
$ ./configure -prefix /usr/local/qt-embedded-4.8.5 \
              -embedded -qt-gfx-linuxfb -qt-gfx-qvfb \
              -qt-gfx-vnc -no-largefile -exceptions -no-accessibility \
              -no-qt3support -qt-zlib -no-gif -no-libtiff \
              -qt-libpng -no-libmng -qt-libjpeg \
              -no-nis -no-cups -depths 8,16,18,32 \
              -qt-kbd-tty -qt-kbd-qvfb -qt-kbd-linuxinput \
              -qt-mouse-linuxinput -qt-mouse-qvfb -qt-mouse-pc
$ make
```

หลังจากเสร็จสิ้นการคอมไพล์แล้ว จะได้ไฟล์ไลบรารีที่จำเป็นสำหรับการนำไปติดตั้งเพิ่มลงในบอร์ดสมองกลฝังตัว ที่ใช้ระบบปฏิบัติการ Embedded Linux ดังนี้

```
-libQtCore.so          <--- ไลบรารีหลัก
-libQtGui.so           <--- ในกรณีที่ต้องการแสดงส่วนกราฟฟิกติดต่อผู้ใช้ (GUI)
-libQtNetwork.so       <--- ในกรณีที่ต้องการใช้งานทางด้านระบบเครือข่าย
-libQtSql.so           <--- ในกรณีที่ต้องการใช้งานในการเก็บข้อมูลด้วย SQlite
-libQtXml.so           <--- ในกรณีที่ต้องการใช้งานในการจัดการไฟล์ XML
```

ซึ่งนักพัฒนาสามารถเลือกไฟล์ไลบรารีที่ต้องการ แล้วทำการคัดลอกไฟล์ไลบรารีไปยังไดเรกทอรี /lib ของ root file system ภายในบอร์ดสมองกลฝังตัว

ในการเริ่มต้นพื้นฐานการพัฒนาโปรแกรมด้วย Qt นั้น สามารถสร้างโปรแกรม “Hello World” อย่างง่าย ได้ดังนี้

```
$ vim main.cpp

// main.cpp

#include<QtCore>
int main(){
      qDebug() << "Hello World\n";
}
```

ก่อนจะทำการคอมไพล์ จะต้องมีการสร้างไฟล์ทั้ง 2 ไฟล์ขึ้นมา ได้แก่ ไฟล์โปรเจค (.pro) และไฟล์ Makefile ด้วยคำสั่งดังนี้

```
$ qmake -project
```

ไฟล์โปรเจค (.pro) จะถูกสร้างขึ้นมาโดยอัตโนมัติ ดังรายละเอียดภายในไฟล์ข้างล่างนี้

```
#######################################################################
# Automatically generated by qmake
#######################################################################

TEMPLATE = app
TARGET =
DEPENDPATH += .
INCLUDEPATH += .

# Input
SOURCES += main.cpp
```

ถัดไปเป็นการสร้าง Makefile โดยรันคำสั่ง qmake โดยที่ไม่ต้องใส่พารามิเตอร์

```
$ qmake
$ cat Makefile
####### Compile
main.o: main.cpp
      $(CXX) -c $(CXXFLAGS) $(INCPATH) -o main.o main.cpp

####### Install
install:   FORCE

uninstall:   FORCE

FORCE:
```

ทำการคอมไพล์โปรแกรม

```
$ make
g++ -c -pipe -O2 -Wall -W -D_REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB
-DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++ -I. -I/usr/
include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -I. -I. -o
main.o main.cpp
g++ -Wl,-O1 -o qt main.o    -L/usr/lib -lQtGui -lQtCore -lpthread
```

ทดสอบเรียกโปรแกรม

```
$ ./qt
Hello World
```

ตัวอย่างโปรแกรม Qt ในรูปแบบกราฟฟิกที่ติดต่อกับผู้ใช้ (GUI program) สามารถพัฒนาผ่านโปรแกรม Qt Creator หรือสามารถเขียนโค้ดโปรแกรมด้วย Text editor ทั่วไป ดังแสดงข้างล่างนี้

```
$ vim qt_gui.cpp
// qt_gui.cpp
#include<QApplication>
#include<QLabel>

int main(int argc, char *argv[])
{
  QApplication app(argc, argv);
  QLabel* hello = new QLabel("<h1>Hello, World</h1>");
  hello->show();
  hello->setWindowTitle("Welcome to Embedded System World");
  return app.exec();
}
```

ทำการสร้างไฟล์โปรเจ็ค (.pro) และไฟล์ Makefile ดังคำสั่งข้างล่างนี้

```
$ qmake -project
$ qmake
```

ไฟล์ทั้งสองจะถูกสร้างขึ้นดังรายละเอียดดังนี้

```
$ ls
gui.pro  Makefile  qt_gui.cpp

$ cat gui.pro
#############################################################
# Automatically generated by qmake (2.01a) Mon Sep 30 23:09:58 2013
#############################################################

TEMPLATE = app
```

```
TARGET =
DEPENDPATH += .
INCLUDEPATH += .

# Input
SOURCES += qt_gui.cpp
```

```
$ cat Makefile
#############################################################################
# Makefile for building: gui
# Generated by qmake (2.01a) (Qt 4.6.2) on: Mon Sep 30 23:10:00 2013
# Project:  gui.pro
# Template: app
# Command: /usr/bin/qmake -unix -o Makefile gui.pro
#############################################################################

####### Compiler, tools and options

CC            = gcc
CXX           = g++
DEFINES       = -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED
CFLAGS        = -pipe -O2 -Wall -W -D_REENTRANT $(DEFINES)
CXXFLAGS      = -pipe -O2 -Wall -W -D_REENTRANT $(DEFINES)
...
compiler_uic_clean:
compiler_yacc_decl_make_all:
compiler_yacc_decl_clean:
compiler_yacc_impl_make_all:
compiler_yacc_impl_clean:
compiler_lex_make_all:
compiler_lex_clean:
compiler_clean:

####### Compile

qt_gui.o: qt_gui.cpp
      $(CXX) -c $(CXXFLAGS) $(INCPATH) -o qt_gui.o qt_gui.cpp

####### Install

install:   FORCE

uninstall:   FORCE

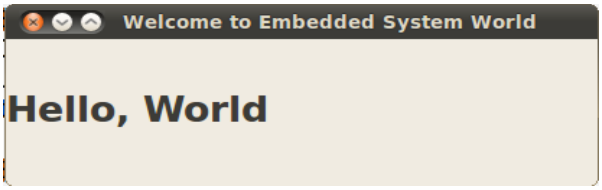
FORCE:
```

ทำการคอมไพล์โปรแกรม

```
$ make
g++ -c -pipe -O2 -Wall -W -D_REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB
-DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++ -I. -I/usr/
include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -I. -I. -o
qt_gui.o qt_gui.cpp
g++ -Wl,-O1 -o gui qt_gui.o    -L/usr/lib -lQtGui -lQtCore -lpthread
```

ทดสอบเรียกโปรแกรม Hello World ดังข้างล่าง

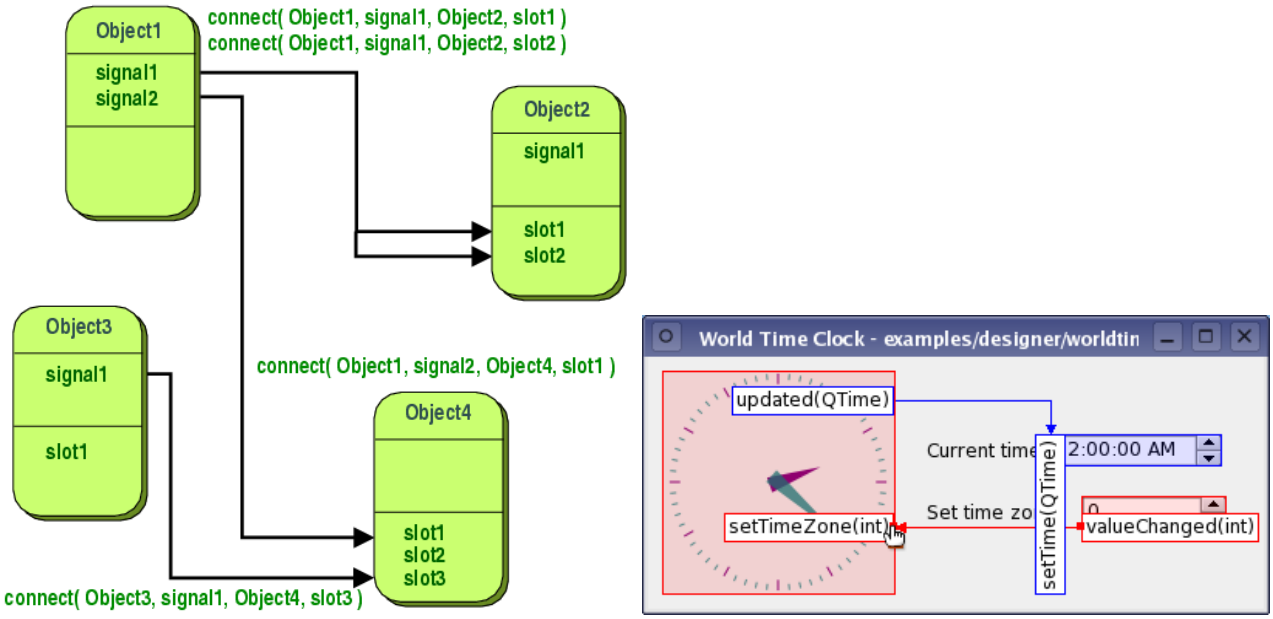
```
$ ./gui
```



รูปที่ 4.17 ผลลัพธ์การรันโปรแกรม Hello World

กลไกการทำงานของ SIGNAL และ SLOT

Signals และ slots ใช้สำหรับการสื่อสารระหว่างวัตถุ (object) ซึ่งเป็นกลไกหลักในการนำไปพัฒนาโปรแกรม Qt โดยเฉพาะทางด้านกราฟฟิกติดต่อกับผู้ใช้ (GUI) ดังแสดงในรูปข้างล่าง



รูปที่ 4.18 กลไกการเชื่อมระหว่าง Signals และ Slots

ฟังก์ชันในการเชื่อมระหว่าง signals กับ slots นั้นจะมีรูปแบบการเขียนโปรแกรมดังนี้

connect(*sender*, SIGNAL(*signal*), *receiver*, SLOT(*slot*));

ตัวอย่างโปรแกรมประยุกต์การทำงานของ signals และ slots

```
#include<QApplication>
```

```
#include<QPushButton>

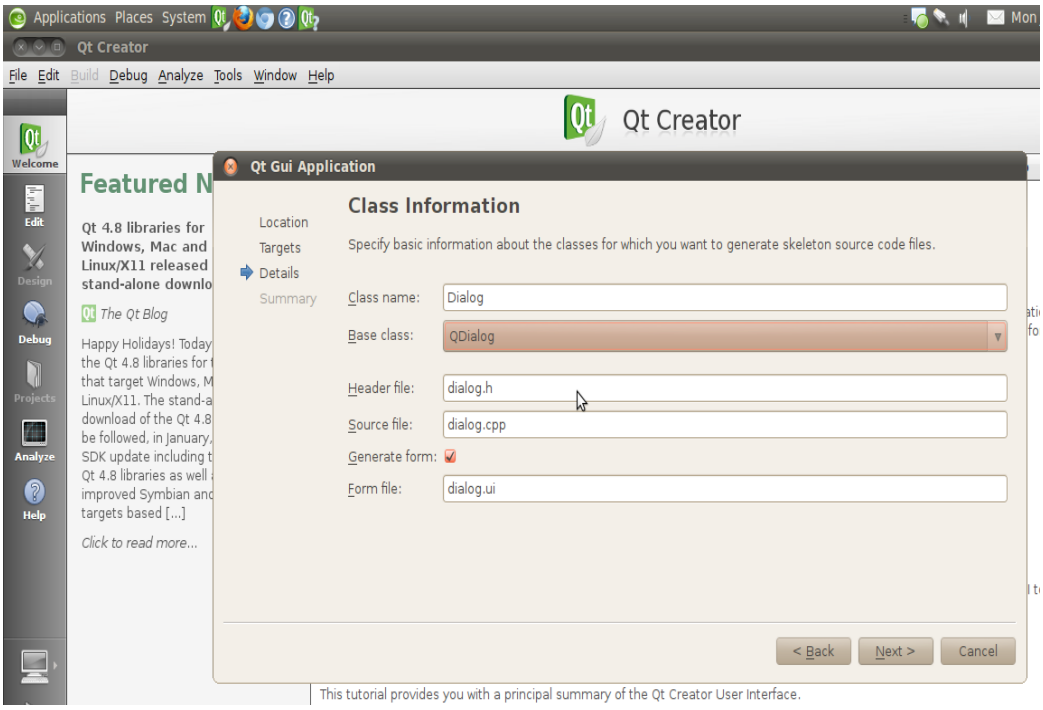
int main(int argc, char *argv[])
{
  QApplication app(argc, argv);
  QPushButton* hello = new QPushButton("Quit");
  hello->resize(150, 75);
  QObject::connect(hello, SIGNAL(clicked()), &app, SLOT(quit()));
  hello->show();
  hello->setWindowTitle("Hi");
  return app.exec();
}
```

จากโค้ดโปรแกรมข้างบนเป็นการเชื่อม signal ของปุ่มที่เกิดจากการถูกกด (clicked()) ไปยังตัว slot ที่ชื่อฟังก์ชัน quit() ซึ่งทำการออกจากโปรแกรม ดังนั้นการทำงานของโปรแกรมคือเมื่อมีการกดปุ่มก็จะออกจากโปรแกรมทันที โดยข้อสังเกตในการใช้ signal และ slot มีดังนี้

1. หนึ่ง signal สามารถเชื่อมได้หลาย slots
2. หลาย signals สามารถเชื่อมกับ slot เดียวกันได้
3. สามารถยกเลิกการเชื่อมต่อ (connection) ได้

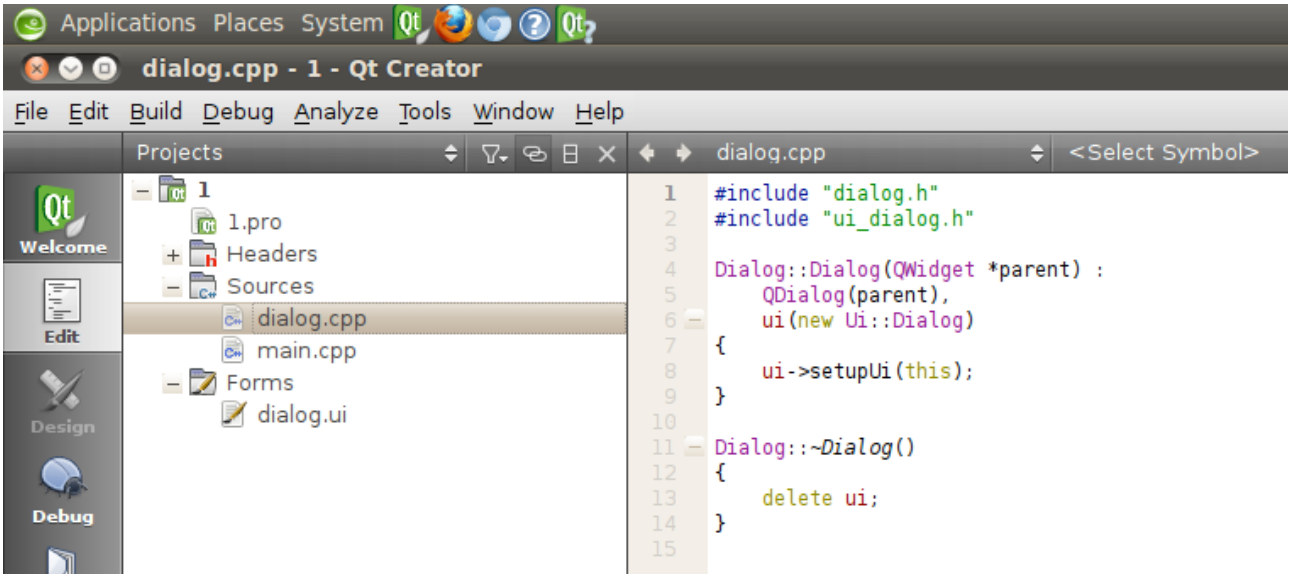
ตัวอย่างการพัฒนาโปรแกรมด้วย Qt Creator

เปิดโปรแกรม Qt Creator แล้วทำการสร้างโปรเจคใหม่ โดยเลือกชนิด Qt GUI Application แล้วเลือก Base class ให้เป็นแบบ QDialog ดังแสดงในรูปข้างล่าง



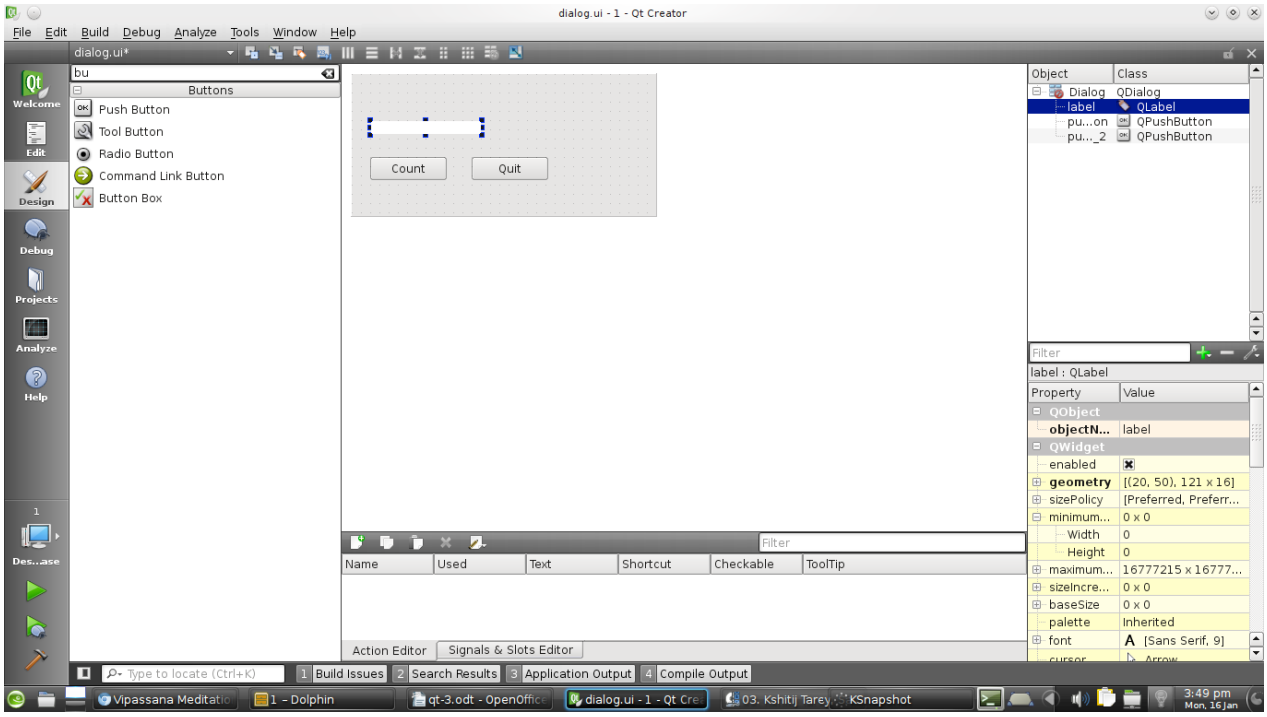
รูปที่ 4.19 หน้าต่างส่วนการตั้งค่ารายละเอียดของคลาส

หลังจากกด Next แล้ว Qt Creator จะได้โค้ดโปรแกรมดังรูปด้านล่าง



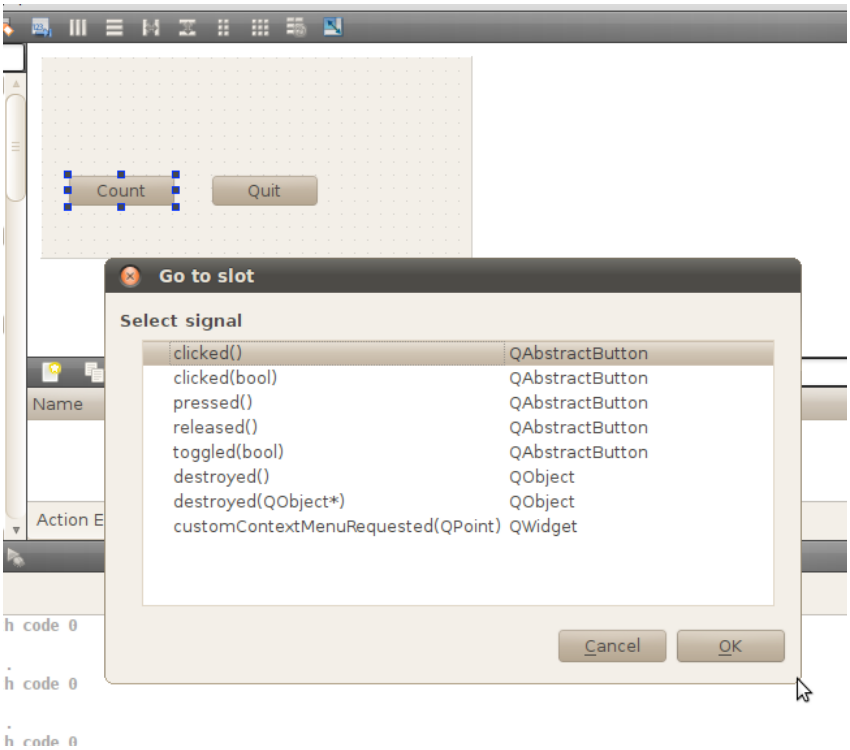
รูปที่ 4.20 แสดงโค้ดเบื้องต้นที่ถูกสร้างขึ้นมาอัตโนมัติ

เปิดไฟล์ dialog.ui เพื่อสร้าง UI ให้กับโปรแกรม โดยการเพิ่ม QPushButton และ QLabel ดังแสดงในรูปข้างล่าง



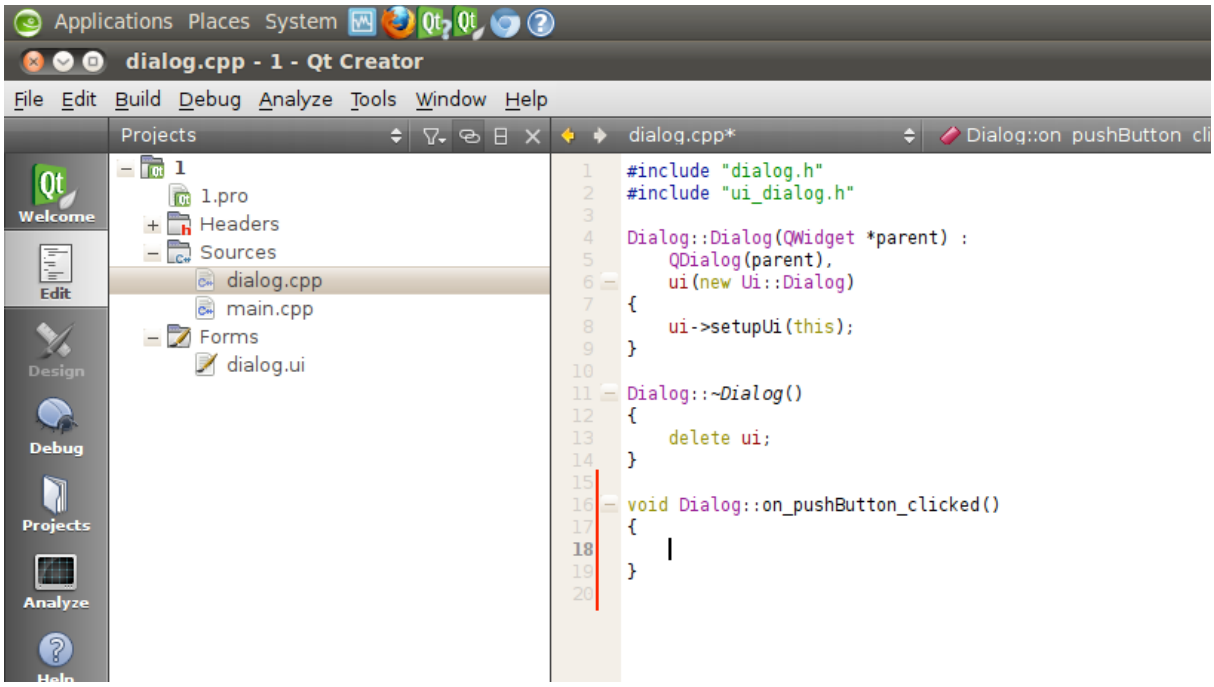
รูปที่ 4.21 การสร้าง GUI ผ่านส่วนของ Design

ขั้นตอนต่อไปจะเป็นการสร้างการเชื่อมโยง โดยการคลิ๊กขวาที่ปุ่ม Count แล้วเลือก “Go to Slot” จากนั้นเลือก signal ที่ต้องการเชื่อมไปยัง slot ในที่นี้ให้เลือก clicked() จากนั้นกด OK



รูปที่ 4.22 การตั้งค่า Signal เพื่อกำหนด Slot เป้าหมาย

โปรแกรมจะทำการสร้างฟังก์ชันสำหรับการเชื่อมต่อระหว่าง signals และ slots ให้อัตโนมัติ โดยจะตั้งชื่อ slots ให้ชื่อว่า “on_pushButton_clicked” ดังแสดงในรูปข้างล่าง



รูปที่ 4.23 Method ที่ถูกสร้างขึ้นเมื่อมีการเลือก slot

เพิ่มเติมโค้ดโปรแกรมเข้าไปในฟังก์ชัน on_pushButton_clicked() ดังนี้

```
void Dialog::on_pushButton_clicked()
{
    static int count;
```

```
    count++;
    ui->label->setText(QString::number(count));
}
```

ในขั้นตอนสุดท้ายทำการคอมไพล์และรันโปรแกรม โดยกดปุ่มลัด Ctrl+R จะได้ผลลัพธ์ดังรูปด้านล่าง ซึ่งเมื่อผู้ใช้กดปุ่ม Count โปรแกรมก็จะทำการบวกเลขเพิ่มขึ้นทีละ 1



รูปที่ 4.24 ผลลัพธ์การรันโปรแกรมเมื่อกดปุ่ม Count

การพัฒนาโปรแกรมติดต่อพอร์ตอนุกรม

การพัฒนาโปรแกรมด้วย Qt นั้นเริ่มได้รับความนิยมมากขึ้น โดยเฉพาะในวงการการพัฒนาบนระบบสมองกลฝังตัว เนื่องจากการทำงานของโปรแกรมที่รวดเร็ว และการแสดงผลทางด้านกราฟฟิกที่ดีขึ้น นอกจากนั้นการพัฒนาโปรแกรมเพื่อติดต่อสื่อสารกับพอร์ตอนุกรม กับอุปกรณ์ภายนอก เช่น บอร์ดไมโครคอลโทรเลอร์ เป็นต้น ก็ยังจำเป็นต้องมีอยู่ ดังนั้นใน Qt เวอร์ชันใหม่ ตั้งแต่เวอร์ชัน 5 เป็นต้นไป ได้มีการรวมเอาโมดูล QtSerialPort เข้าไปอยู่ในไลบรารี Qt5 เป็นที่เรียบร้อยแล้ว เพื่อความสะดวกในการเรียกใช้งาน ซึ่งสามารถติดต่อได้ทั้ง hardware serial ports และ virtual serial ports

ภายในโมดูล QtSerialPort ประกอบไปด้วย 2 คลาสหลักๆคือ SerialPort และ SerialPortInfo โดยคลาส SerialPort จะเป็นคลาสพื้นฐานที่มีเมทธอดต่างๆ ในการตั้งค่าพื้นฐานสำหรับพอร์ตอนุกรม

ตาราง 4.5 การรองรับการตั้งค่าพอร์มอนุกรมด้วย Qt ในแต่ละระบบปฏิบัติการ

ระบบปฏิบัติการ	สนับสนุน	รายละเอียด
Windows NT/2K/XP/Vista/7/8	YES	Full support
Windows CE	YES	เวอร์ชัน 5 และ 6 บนตัว WinCE Emulator
Gnu/Linux	YES	Full support
MacOS X	YES	Full support

ระบบปฏิบัติการ	สนับสนุน	รายละเอียด
Unix อื่นๆ	YES	POSIX-compatible ทั้งหมด

สำหรับคลาส SerialPortInfo จะเป็นคลาสตัวช่วย ที่จะแสดงค่าสถานะต่างๆของพอร์ตอนุกรมที่สามารถเรียกใช้ได้ในระบบคอมพิวเตอร์ หรือบอร์ดสมองกลฝังตัว

ตาราง 4.6 การรองรับของฟังก์ชัน SerialPortInfo ในแต่ละระบบปฏิบัติการ

ระบบปฏิบัติการ	สนับสนุน	รายละเอียด
Windows NT/2K/XP/Vista/7/8	YES	Full support โดยเรียกใช้ฟังก์ชัน SetupAPI
Windows CE	YES	เวอร์ชัน 5 และ 6 บนตัว WinCE Emulator
Gnu/Linux	YES	Full support โดยเรียกใช้ libudev หรือค้นหาได้ในไดเรกทอรี /dev/ โดยตรง
MacOS X	YES	Full support
Unix อื่นๆ	YES	POSIX-compatible ทั้งหมด ค้นหาได้ในไดเรกทอรี /dev/ โดยตรง

แต่โมดูล QtSerialPort ยังมีข้อจำกัดบางอย่างที่ยังไม่รองรับได้แก่

-การทำงานแบบโหมด Terminal, control CR/LF

-ตั้งค่า timeouts และ delays ของการอ่าน

-การแจ้งเตือนเมื่อการมีเปลี่ยนสถานะของ RS-232 ขาOUT

ดาวน์โหลดโมดูล QtSerialPort แล้วทำการคอมไพล์และติดตั้งเพิ่มเข้าไปใน Qt4/Qt5 ด้วยคำสั่งดังนี้

```
$ git clone git://gitorious.org/qt/qtserialport.git
Initialized empty Git repository in
/home/student/Downloads/qtserialport/.git/
remote: Counting objects: 4482, done.
remote: Compressing objects: 100% (1737/1737), done.
remote: Total 4482 (delta 3158), reused 3764 (delta 2719)
Receiving objects: 100% (4482/4482), 1.22 MiB | 233 KiB/s, done.
Resolving deltas: 100% (3158/3158), done.
```

```
$ cd qtserialport/
$ qmake qtserialport.pro
$ make
$ sudo make install
```

ในการพัฒนาโปรแกรม Qt เพื่อติดต่อพอร์ตอนุกรมนั้นในกรณีที่เป็น Qt4 และ Qt5 นั้น จะต้องมีการระบุการเรียกใช้โมดูล QtSerialPort ในไฟล์โปรเจค (.pro) เพิ่มเข้าไปด้วยหลังจากรันคำสั่ง `qmake -project` เพื่อสร้างไฟล์โปรเจค (.pro) ดังนี้

กรณีที่เป็น Qt4

```
CONFIG += serialport
```

กรณีที่เป็น Qt5

```
QT += serialport
```

แต่สำหรับการ include ในโค้ดโปรแกรมทั้ง Qt4 และ Qt5 จะเหมือนกัน

```
...
#include <QtSerialPort/QSerialPort>
#include <QtSerialPort/QSerialPortInfo>
...
```

การเปิดพอร์ตอนุกรมสามารถเลือกระดับของการเข้าถึงได้ดังนี้ read-only (r/o), write-only (w/o), read/write (r/w) และเมื่อเปิดพอร์ตได้สำเร็จ โมดูล QSerialPort จะใช้ค่าที่ตั้งไว้ดั้งเดิมของพอร์ตนั้น แต่อย่างไรก็ตามก็สามารถทำการตั้งค่าใหม่ได้ โดยเรียกใช้ฟังก์ชัน setBaudRate(), setDataBits(), setParity(), setStopBits() และ setFlowControl() ตามลำดับ

ตัวอย่างการเปิดพอร์ตอนุกรม /dev/ttyS0

```
QSerialPort serial;
serial.setPortName("/dev/ttyS0");
serial.setBaudRate(QSerialPort::Baud9600);
if (serial.open(QIODevice::ReadWrite)) {
    // to do this
}
```

ตัวอย่างโปรแกรมการใช้คลาส QSerialPortInfo เพื่อทำการตรวจสอบพอร์ตที่เรียกใช้งานได้ แล้วทำการส่งรายละเอียดของพอร์ตนั้นต่อไปให้คลาส QSerialPort เพื่อทำการเปิดต่อไป

```
#include <QtCore/QCoreApplication>
#include <QtCore/QDebug>
```

```
#include <QtSerialPort/QSerialPort>
#include <QtSerialPort/QSerialPortInfo>

QT_USE_NAMESPACE

int main(int argc, char *argv[]) {
        QCoreApplication a(argc, argv);

        // Example use QSerialPortInfo
   foreach (const QSerialPortInfo &info, QSerialPortInfo::availablePorts()){
      qDebug() << "Name    : " << info.portName();
      qDebug() << "Description : " << info.description();
      qDebug() << "Manufacturer: " << info.manufacturer();

         // Example use QSerialPort
    QSerialPort serial;
    serial.setPort(info);
      if (serial.open(QIODevice::ReadWrite))
        serial.close();
      }
      return a.exec();
}
```

สร้างไฟล์โปรเจค (.pro) แล้วเพิ่มให้รองรับการใช้โมดูล QtSerialPort และทำการคอมไพล์ ดังคำสั่งข้างล่างนี้

```
$ qmake -project
$ echo “CONFIG += serialport” >> serial.pro
$ qmake
$ make
g++ -Wl,-O1 -Wl,-rpath,/usr/local/qt-embedded-4.8.5/lib -o serial main.o
-L/usr/local/qt-embedded-4.8.5/lib -lQtSerialPort -lQtGui
-L/usr/local/qt-embedded-4.8.5/lib -lQtNetwork -lQtCore -lpthread
$ ls
main.cpp  main.o  Makefile  serial  serial.pro
$ ./serial
Name    :  "ttyS0"
Description :  ""
Manufacturer:  ""
Name    :  "ttyS1"
Description :  ""
Manufacturer:  ""
```

การพัฒนาโปรแกรมแบบ MULTI-THREADING

การเขียนโปรแกรมเพื่อให้เทรดหลายตัวสามารถทำงานพร้อมกันได้ (multi-threading) ดังที่เคยอธิบายการพัฒนาด้วยภาษา C ในบทที่ผ่านมา สำหรับการพัฒนาโปรแกรมในลักษณะเป็นกราฟฟิกที่ติดต่อกับผู้ใช้ด้วยภาษา C++ กับตัว Qt นั้น ถือว่าเป็นสิ่งจำเป็นอย่างมาก เนื่องจากถ้างานไหนที่มีโอกาสจะมีการประมวลผลนาน ก็ควรเปลี่ยนให้ไปประมวลผลการทำงานอยู่เบื้องหลังแทน (background processing) เพื่อไม่ให้กระทบต่อการใช้งานส่วนกราฟฟิกกับผู้ใช้ ซึ่งการพัฒนาโปรแกรมในลักษณะ multi-threading นั้นได้นำมาใช้อย่างแพร่หลาย สำหรับการพัฒนาใน Qt นั้นสามารถเรียกใช้งานได้ง่าย และยังรองรับการเชื่อมต่อกับแบบ signal และ slot เพื่อใช้ส่งข้อมูลระหว่างกันไปมาของเทรดได้

Qt สามารถใช้งาน Thread ได้ 2 แบบใหญ่ คือ

1. แบบ Low Level ด้วย QThread

คลาส QThread นั้นจะใช้สำหรับ 1 เทรด ต่อ 1 QThread ซึ่งจะเริ่มทำงานจากฟังก์ชัน run() แต่จะต้องทำการ implement ฟังก์ชัน run() นี้ก่อน (**::run()**) ดังตัวอย่างโปรแกรมข้างล่าง

สร้างไฟล์ writer.h และ main.cpp ดังนี้

```
$ vim writer.h
// writer.h
#ifndef WRITER_H
#define WRITER_H

#include <QThread>
#include <QString>
#include <iostream>
class Writer: public QThread {
public:
      explicit Writer(const QString& mark) :
                  mark_(mark) {
      }

      void run();
private:
      QString mark_;
};
#endif // WRITER_H

void Writer::run() {
      for (int i = 0; i < 20; ++i) {
            std::cout << qPrintable(mark_);
            msleep(200);
      }
}
```

ในไฟล์ main.cpp จะมีการเขียนลงไปในฟังก์ชัน run() และจะมีการหยุดไปชั่วขณะประมาณ 200 มิลลิวินาที หลังจากนั้นก็จะแสดงข้อความออกทางหน้าจอ ดังโค้ดข้างล่างนี้

```
$ vim main.cpp
// main.cpp
#include <QtCore/QCoreApplication>
#include <QTimer>
#include "writer.h"

int main(int argc, char *argv[])
{
   QCoreApplication a(argc, argv);

   Writer w1("One");
   Writer w2("Two");
   Writer w3("Three");

   w1.start();
   w2.start();
   w3.start();

   w1.wait();
   w2.wait();
   w3.wait();

   QTimer::singleShot(12000, &a, SLOT(quit()));

   return a.exec();
}
```

สร้างไฟล์โปรเจค (.pro) และไฟล์ Makefile เพื่อคอมไพล์โปรแกรม ดังคำสั่งข้างล่างนี้

```
$ qmake -project
$ qmake
$ make
$ ./thread
Two Three One Two Three One Two Three One Three Two One Two One Three Two One
Three Three Two One Three One Two Two Three One Two Three One Two Three One
Two One Three Two One Three Two One Three Two One Three One Three Two One
Three Two One Three Two One Three Two One Three Two
```

2. แบบ High Level ด้วย QConcurrent

QtConcurrent เป็นตัว API ในระดับสูง สำหรับการเขียนโปรแกรมแบบ multi-threading บน Qt นี้ จะสามารถใช้งานได้ง่ายยิ่งขึ้นโดยที่ไม่ต้องลงลึกในรายละเอียดของเทรดและการจัดการจำนวนเทรดต่างๆ โดย QtConcurrent ได้รวมเอาฟังก์ชันไว้หลายรูปแบบเพื่อให้เหมาะสำหรับการนำไปประยุกต์ใช้

กับงานแบบไม่ตรงจังหวะกัน (asynchronous task) โดยเฉพาะในโปรแกรมแบบกราฟฟิกติดต่อกับผู้ใช้ (GUI applications)

QtConcurrent::run() จะทำการเรียกฟังก์ชันขึ้นมาภายใน thread ที่สร้างขึ้นมาใหม่

```
extern void aFunction();
extern void bFunction();

QFuture<void> future1 = QtConcurrent::run(aFunction);
QFuture<void> future2 = QtConcurrent::run(bFunction);
```

สามารถใช้คลาส QFuture และ QFutureWatcher ในการตรวจสอบสถานะของฟังก์ชันที่ถูกเรียกเข้าไปในเทรดได้ตัวอย่างโปรแกรมข้างล่างนี้

```
#include <QCoreApplication>
#include <qtconcurrentrun.h>
#include <QThread>

#ifndef QT_NO_CONCURRENT

void myRunFunction(QString name) {
	for (int i = 0; i <= 5; i++) {
		qDebug() << name << " " << i << "from" << QThread::currentThread();
	}
}

int main(int argc, char *argv[]) {
	QCoreApplication a(argc, argv);
	QFuture<void> t1 = QtConcurrent::run(myRunFunction, QString("A"));
	QFuture<void> t2 = QtConcurrent::run(myRunFunction, QString("B"));
	QFuture<void> t3 = QtConcurrent::run(myRunFunction, QString("C"));

	t1.waitForFinished();
	t2.waitForFinished();
	t3.waitForFinished();

	return a.exec();
}

#else

#include <QLabel>

int main(int argc, char *argv[])
{
	QApplication app(argc, argv);
	QString text("Qt Concurrent is not yet supported on this platform");
	QLabel *label = new QLabel(text);
	label->setWordWrap(true);
	label->show();
```

```
        qDebug() << text;
        app.exec();
}
#endif
```

ทำการสร้างไฟล์โปรเจค (.pro) แล้วเพิ่มไลบรารี concurrent เข้าไปในไฟล์โปรเจ็ค ดังคำสั่งข้างล่างนี้

```
$ qmake -project
$ echo "CONFIG += concurrent" >> qtconcurrent.pro
```

ทำการสร้างไฟล์ Makefile และคอมไพล์โปรแกรม เพื่อทดสอบการทำงานของ multi-threading ดังคำสั่งข้างล่างนี้

```
$ qmake
$ make
$ ls
main.cpp  main.o  Makefile  qtconcurrent  qtconcurrent.pro
$ ./qtconcurrent
"B"   0 from QThread(0x128d1c0, name = "Thread (pooled)")
"B"   1 from QThread(0x128d1c0, name = "Thread (pooled)")
"B"   2 from QThread(0x128d1c0, name = "Thread (pooled)")
"A"   0 from QThread(0x128cb10, name = "Thread (pooled)")
"A"   1 from QThread(0x128cb10, name = "Thread (pooled)")
"A"   2 from QThread(0x128cb10, name = "Thread (pooled)")
"A"   3 from QThread(0x128cb10, name = "Thread (pooled)")
"A"   4 from QThread(0x128cb10, name = "Thread (pooled)")
"A"   5 from QThread(0x128cb10, name = "Thread (pooled)")
"B"   3 from QThread(0x128d1c0, name = "Thread (pooled)")
"B"   4 from QThread(0x128d1c0, name = "Thread (pooled)")
"C"   0 from QThread(0x128cb10, name = "Thread (pooled)")
"C"   1 from QThread(0x128cb10, name = "Thread (pooled)")
"B"   5 from QThread(0x128d1c0, name = "Thread (pooled)")
"C"   2 from QThread(0x128cb10, name = "Thread (pooled)")
"C"   3 from QThread(0x128cb10, name = "Thread (pooled)")
"C"   4 from QThread(0x128cb10, name = "Thread (pooled)")
"C"   5 from QThread(0x128cb10, name = "Thread (pooled)")
```

บทที่ 5 พื้นฐานระบบปฏิบัติการแอนดรอยด์สำหรับนักพัฒนา

ระบบปฏิบัติการแอนดรอยด์

ประวัติศาสตร์ของระบบปฏิบัติการแอนดรอยด์เริ่มต้นตั้งแต่ปี ค.ศ. 2002 โดยเมื่อผู้ก่อตั้งบริษัทกูเกิลทั้งสองคือ Larry Page และ Sergey Brin ได้เข้าร่วมฟังสัมนา ณ มหาวิทยาลัยสแตนฟอร์ด (Stanford University) ในหัวข้อการพัฒนาโทรศัพท์มือถือรุ่นใหม่ชื่อ “SideKick Phone” นำเสนอโดยนาย Andy Rubin ซึ่งขณะนั้นดำรงตำแหน่งซีอีโอ (CEO) ของบริษัท Danger Inc. เขาได้แสดงความสามารถของตัวโทรศัพท์ชนิดพกพาชื่อว่า SideKick Phone ซึ่งเป็นโทรศัพท์ชนิดพกพาตัวแรกที่สามารถเชื่อมต่อผ่านระบบอินเทอร์เน็ต (Internet) และทำงานได้หลากหลายตัวอย่างเช่น การส่งข้อความ (Instant Messenger), อีเมล์ (Mail), ตารางปฏิทิน (Calendar), และการท่องเว็ปไซต์ (Browser) โดยเฉพาะการตั้งหน้าการค้นหาหลัก (Search Engine default) ให้เป็น Google search ก็ยิ่งทำให้ Larry Page เกิดความประทับใจและได้กลายเป็นหนึ่งในผู้ใช้ Sidekick Phone

แม้ว่าตัว Sidekick Phone จะเป็นมือถือรุ่นใหม่ที่โดดเด่นและทำให้ผู้ใช้ต่างรู้สึกตื่นเต้นได้ไม่น้อยก็ตามที แต่ก็ยังไม่สามารถสร้างให้เกิดผลสำเร็จในทางธุรกิจได้เท่าที่ควรจะเป็น จึงทำให้ Andy Rubin ถูกบีบให้ออกจากบริษัท Danger Inc. ในช่วงต้นปี ค.ศ. 2003 หลังจากหนึ่งปีผ่านไป Andy Rubin พร้อมกับเงินก้อนหนึ่ง เขาก็ยังคงมีความมุ่งมั่นที่ต้องการสร้างธุรกิจทางด้านระบบปฏิบัติการแนวคิดใหม่สำหรับโทรศัพท์เคลื่อนที่ (Mobile Operating System) เขาจึงทำการจดโดเมนใหม่ในชื่อว่า Android.com พร้อมกับก่อตั้งบริษัทใหม่ในชื่อเดียวกันว่า Android Inc. และได้เริ่มทำการพัฒนา Open Mobile handset platform เพื่อให้กลายเป็นระบบปฏิบัติการแบบเปิด (Open Operating System) สำหรับผู้ผลิตโทรศัพท์เพื่อให้สามารถนำไปใช้เป็นระบบปฏิบัติการพื้นฐานภายในเครื่องของตนได้

จนกระทั่งประมาณเดือนสิงหาคม ปี ค.ศ. 2005 ทางบริษัทกูเกิลได้ขอเจรจากับ Andy Rubin เพื่อขอเข้าซื้อกิจการบริษัทของเขาจนในที่สุดก็ได้บรรลุข้อตกลงร่วมกันด้วยมูลค่าที่ไม่ได้ถูกเปิดเผยต่อสาธารณะชน และหลังจากนั้นเป็นต้นมาในช่วงปลายปี ค.ศ. 2007 บริษัทกูเกิลได้ประกาศเข้าร่วมเป็นสมาชิกใน Open Handset Alliance (OHA) เพื่อที่จะสร้างระบบปฏิบัติการแบบเปิดสำหรับโทรศัพท์เคลื่อนที่ในชื่อระบบปฏิบัติการแอนดรอยด์ (Android Operating System) อย่างเป็นทางการ และต่อมาในเดือนกันยายน ปี ค.ศ. 2008 บริษัทกูเกิลจึงได้ปล่อยระบบปฏิบัติการแอนดรอยด์ เวอร์ชั่น 1.0 โดยใช้ชื่อรหัส (Codename) ว่า Astro ต่อสาธารณะชนเป็นครั้งแรก และก็ได้ปล่อยรุ่นต่างๆอย่างต่อเนื่องจนกระทั่งปัจจุบันดังแสดงในตารางข้างล่าง



รูปที่ 5.1 รุ่นของระบบปฏิบัติการแอนดรอยด์

ตาราง 5.1 รายละเอียดการเปลี่ยนแปลงที่สำคัญของแต่ละรุ่นของแอนดรอยด์

ชื่อโค้ด	เวอร์ชัน	ออกเมื่อ	การเปลี่ยนแปลงสำคัญ	KERNEL	SDK API	NDK API
ไม่ระบุ	1.0	กันยายน 2551		2.6.25	1	N/A
Petit Four	1.1	กุมภาพันธ์ 2552		2.6.25	2	N/A
Cupcake	1.5	เมษายน 2552	- Onscreen soft keyboard - หน้าจอหมุนอัตโนมัติ	2.6.27	3	1
Donut	1.6	กันยายน 2552	- มีสถานะการใช้แบตเตอรี่ - สนับสนุนการเชื่อมต่อ VPN	2.6.27	4	2
Eclair	2.0	ตุลาคม 2552	- มี Live Wallpaper - รองรับระบบ Exchange	2.6.29	5	2
	2.0.1	ธันวาคม 2552		2.6.29	6	2
	2.1	มกราคม 2553		2.6.29	7	3
Froyo	2.2	พฤษภาคม 2553	- มีฟังก์ชัน WiFi Hotspot - Just In Time Compile - กำเนิด Nexus One	2.6.32	8	4
Ginger-bread	2.3 - 2.3.2	พฤศจิกายน 2553	- รองรับ NFC และ SIP - มีตัว Download Manager - กำเนิด Nexus S	2.6.35	9	5
	2.3.3 - 2.3.7	กุมภาพันธ์ 2554	- รองรับ ADK	2.6.35	10	5
Honey-comb	3.0	กุมภาพันธ์ 2554	- รองรับการแสดงผลบนแท็ปเล็ต - รองรับ Multi-core CPU	2.6.36	11	6
	3.1.x	พฤษภาคม 2554	- รองรับ USB Host และมี APIs	2.6.36	12	6
	3.2.x	มิถุนายน 2554		2.6.36	13	6

ชื่อโค้ด	เวอร์ชัน	ออกเมื่อ	การเปลี่ยนแปลงสำคัญ	KERNEL	SDK API	NDK API
Ice-Cream Sandwich	4.0 - 4.0.2	ตุลาคม 2554	- รองรับการแสดงผลบนมือถือและแท็ปเล็ต - กำเนิด Galaxy Nexus	3.0.1	14	7
	4.0.3 - 4.0.4	ธันวาคม 2554	- กำเนิด WiFi Direct และ Android Beam - มีความสามารถ Face Unlock	3.0.1	15	7
Jelly Bean	4.1.1 - 4.1.2	มิถุนายน 2555	- มีการปรับปรุงประสิทธิภาพโดยรวมมากขึ้น	3.0.31	16	8
	4.2	พฤศจิกายน 2555	- รองรับการใช้งานแบบ Multi-user	3.0.31	17	8
	4.3	กรกฎาคม 2556	- รองรับเทคโนโลยี Low Power Bluetooth - รองรับการแสดงผลระดับ 4K	3.4.5	18	9
KitKat	4.4	พฤศจิกายน 2556	- รองรับเครื่องรุ่นเก่า (RAM >512MB) - รองรับเซนเซอร์นับก้าวเดิน - รองรับระบบส่ง Infrared Remote	3.4.5	19	9

ด้วยความร่วมมือระหว่างทีมดูแลลีนุกซ์คอร์เนลและทีมนักพัฒนาของกูเกิล ก็ได้ทำให้ในช่วงปี ค.ศ. 2011 ถึง ค.ศ. 2012 เกิดโครงการ Android Mainlining Project ขึ้นโดยมีวัตถุประสงค์หลักคือการรวมไดรเวอร์ของแอนดรอยด์ (Android drivers) และความสามารถต่างๆเข้าไปอยู่ในลีนุกซ์คอร์เนลหลัก ตั้งแต่เคอร์เนล เวอร์ชัน 3.3 เป็นต้นมาจนกระทั่งปัจจุบันโครงการนี้ก็สามารถรวมกันได้อย่างสมบูรณ์ในเคอร์เนลเวอร์ชัน 3.5

ภายในระบบปฏิบัติการแอนดรอยด์เองก็ได้มีการปรับปรุงการทำงานภายในระบบปฏิบัติการลีนุกซ์ที่เป็นแกนกลางเพื่อให้เหมาะสมและสามารถรองรับการใช้งานในสภาพแวดล้อมของอุปกรณ์ชนิดพกพา (Mobile Environment) ที่ต้องใช้พลังงานไฟฟ้าจากแบตเตอรี่ รวมทั้งการปรับปรุงความเร็วของการสื่อสารระหว่างโปรเซส (InterProcess Communication - IPC) ปรับปรุงระบบการจัดการหน่วยความจำในระหว่างโหลดโปรแกรมใช้งาน และแก้ปัญหาการจัดการหน่วยความจำ เป็นต้น ซึ่งส่วนรายละเอียดหลักๆที่ระบบปฏิบัติการแอนดรอยด์ได้ปรับปรุงและเพิ่มเติมเข้าไปในลีนุกซ์คอร์เนล ดังแสดงในตารางข้างล่าง

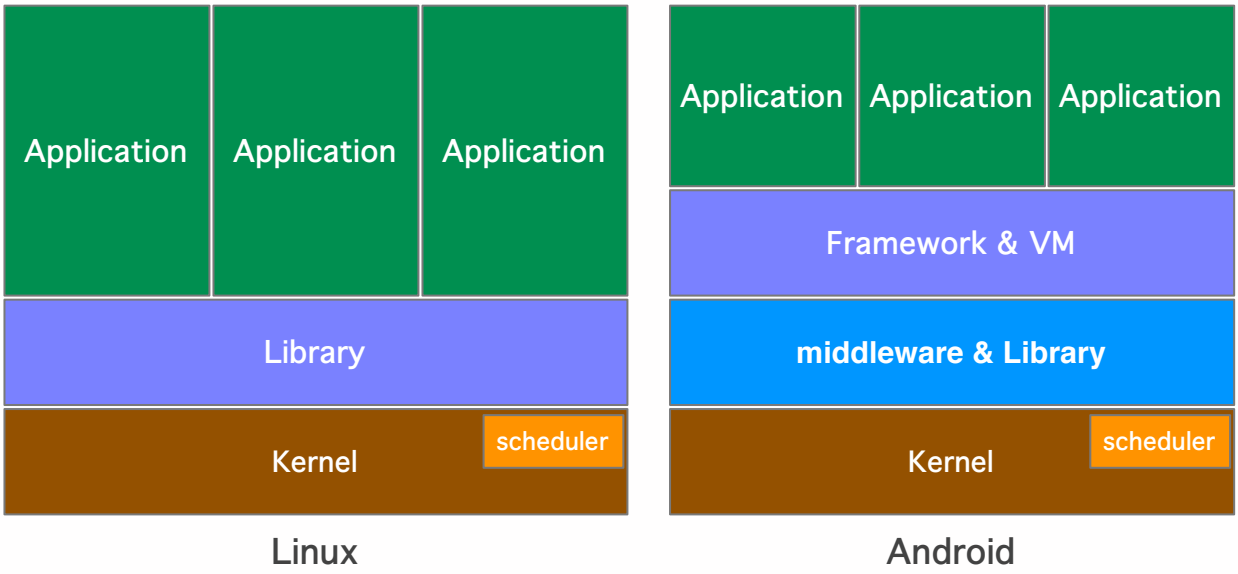
ตาราง 5.2 การปรับเปลี่ยนลีนุกซ์คอร์เนลของระบบปฏิบัติการแอนดรอยด์

เรื่อง	คำอธิบาย
Binder	กลไกการติดต่อสื่อสารระหว่างโปรเซสภายในระบบปฏิบัติการแอนดรอยด์ (Android's IPC mechanism) ซึ่งได้แรงบันดาลใจมาจากโปรเจค Open Binder
Ashmem	ย่อมาจาก Anonymous Shared Memory ทำหน้าที่เป็นตัวจัดสรรพื้นที่ ที่ต้องใช้ร่วมกันภายในหน่วยความจำกลาง (Shared memory allocator) โดยจะถอดส่วนพื้นที่ที่ถูกแชร์ไว้ ในกรณีที่หน่วยความจำเริ่มวิกฤติ
Logger	ส่วนโปรแกรมไดรเวอร์ที่บันทึกข้อมูลต่างๆจากการใช้งานของผู้ใช้ (user space)

เรื่อง	คำอธิบาย
Wakelocks	ส่วนควบคุมไม่ให้เครื่องเข้าสู่โหมดพัก (suspend) ในกรณีที่พลังงานไฟฟ้าน้อย หรือมีการพับปิดหน้าจอเหมือนเครื่องคอมพิวเตอร์ทั่วไปที่ระบบปฏิบัติการจะปิดเครื่องแล้วเข้าสู่โหมด suspend แต่ Wakelocks จะทำงานเพียงแค่ดับหน้าจอ (Screen Turn-Off) เมื่อผู้ใช้ต้องการใช้งานต่อก็เพียงเปิดหน้าจอกลับมา (Screen Turn-On) โดยที่เครื่องไม่ได้ดับ
OOM	ย่อมาจาก Out Of Memory handler คือการจัดการปัญหาในกรณีที่หน่วยความจำเต็มโดยจะมีการส่งสัญญาณ Triggers ก่อนจะส่งต่อให้ Kernel ดำเนินการจัดการ
Alarm	ตัวนับหน่วยเวลาเพื่อแจ้งเตือน (Android's alarm timer) ซึ่งพัฒนาต่อยอดมาจากตัวนาฬิกา RTC (Real-time Clock) ของลีนุกซ์คอร์เนลเดิม
RAM console	ตัวแสดงบันทึกข้อมูลการทำงานของ Kernel ภายใน โดยการเรียกผ่านไฟล์ /proc/last_kmsg

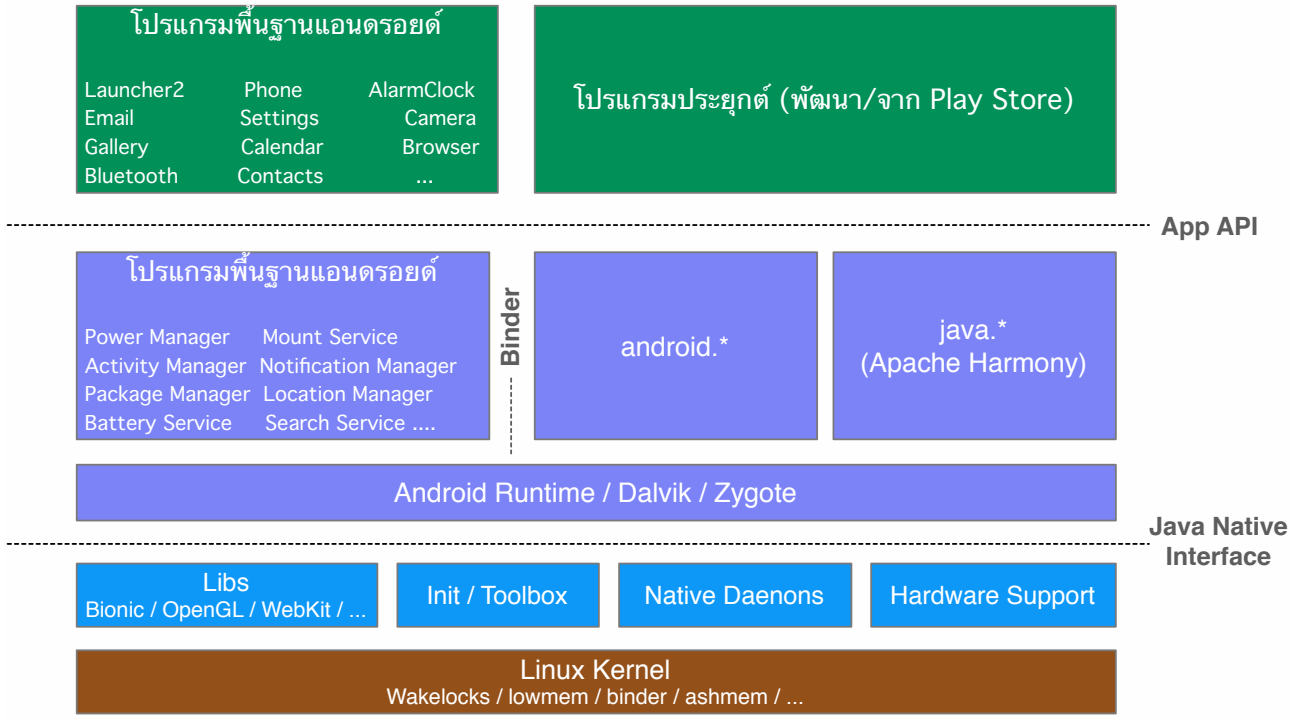
สถาปัตยกรรมของระบบปฏิบัติการแอนดรอยด์

โครงสร้างสถาปัตยกรรมของระบบปฏิบัติการแอนดรอยด์โดยเปรียบเทียบกับระบบปฏิบัติการลีนุกซ์ทั่วไปนั้นจะมีความคล้ายคลึงกันหลายส่วน แต่ก็ยังมีอีกหลายๆส่วนที่แตกต่างกันออกไปเนื่องจากการปรับแต่งให้เหมาะสมกับการนำไปใช้ในอุปกรณ์ชนิดพกพาที่มีทรัพยากรภายในที่กำจัดและพฤติกรรมการใช้ของผู้ใช้ที่แตกต่างจากคอมพิวเตอร์แบบตั้งโต๊ะ ดังรูปเปรียบเทียบสถาปัตยกรรมทั้งสอง



รูปที่ 5.2 เปรียบเทียบสถาปัตยกรรมระบบปฏิบัติการลีนุกซ์ และปฏิบัติการแอนดรอยด์

รายละเอียดภายในโครงสร้างสถาปัตยกรรมของระบบปฏิบัติการแอนดรอยด์ดังแสดงในรูปข้างล่าง



รูปที่ 5.3 รายละเอียดภายในสถาปัตยกรรมแอนดรอยด์

จากโครงสร้างสถาปัตยกรรมของระบบปฏิบัติการแอนดรอยด์จะสังเกตได้ว่ามีการแบ่งออกเป็นส่วนๆที่มีความเกี่ยวเนื่องกัน โดยส่วนบนสุดจะเป็นส่วนที่ผู้ใช้งานทำการติดต่อโดยตรงซึ่งเรียกว่าโปรแกรมประยุกต์ (Applications) และลำดับชั้นลงมาก็จะเป็นองค์ประกอบของโปรแกรมระบบ ไลบรารีต่างๆ จนถึงชั้นล่างสุดก็จะเป็นส่วนที่ติดต่อกับอุปกรณ์โดยผ่านลีนุกซ์คอร์เนล ซึ่งในรายละเอียดของโครงสร้างสถาปัตยกรรมจะอธิบายดังนี้

- **Applications** ส่วนของโปรแกรมที่มีมากับระบบปฏิบัติการหรือเป็นกลุ่มของโปรแกรมที่ผู้ใช้งานได้ทำการติดตั้งไว้ โดยผู้ใช้งานสามารถเรียกใช้โปรแกรมต่างๆได้โดยตรงซึ่งการทำงานของแต่ละโปรแกรมจะเป็นไปตามที่นักพัฒนาโปรแกรมได้ออกแบบและเขียนโค้ดโปรแกรมเอาไว้
- **Application Framework** เป็นส่วนที่มีการพัฒนาขึ้นเพื่อให้นักพัฒนาสามารถพัฒนาโปรแกรมได้สะดวกและมีประสิทธิภาพมากยิ่งขึ้น ดังนั้นนักพัฒนาเพียงแค่ทำการศึกษาวิธีการเรียกใช้งาน Application Framework ในส่วนที่ต้องการใช้งานแล้วนำมาใช้งานซึ่งมีหลายกลุ่มด้วยกัน ตัวอย่างเช่น
    - Activities Manager เป็นกลุ่มของชุดคำสั่งที่จัดการเกี่ยวกับวงจรการทำงานของหน้าต่างโปรแกรม (Activity)
    - Content Providers เป็นกลุ่มของชุดคำสั่งที่ใช้ในการเข้าถึงข้อมูลของโปรแกรมอื่นและสามารถแบ่งปันข้อมูลให้โปรแกรมอื่นเข้าถึงได้
    - View System เป็นกลุ่มของชุดคำสั่งที่เกี่ยวกับการจัดการโครงสร้างของหน้าจอที่แสดงผลในส่วนที่ติดต่อกับผู้ใช้งาน (User Interface)

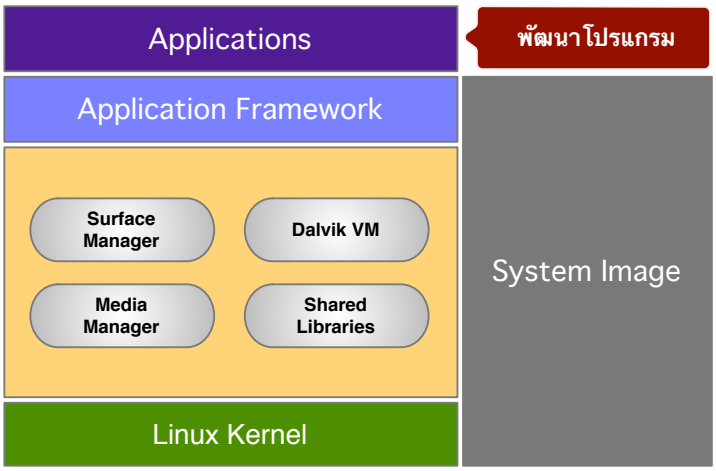
- Telephony Manager เป็นกลุ่มของชุดคำสั่งที่ใช้ในการเข้าถึงข้อมูลด้านโทรศัพท์ เช่น หมายเลขโทรศัพท์ เป็นต้น
- Resource Manager เป็นกลุ่มของชุดคำสั่งในการเข้าถึงข้อมูลที่เป็นข้อความ, รูปภาพ เป็นต้น
- Location Manager เป็นกลุ่มของชุดคำสั่งที่เกี่ยวกับตำแหน่งทางภูมิศาตร์ ที่ระบบปฏิบัติการได้รับค่าจากอุปกรณ์
- Notification Manager เป็นกลุ่มของชุดคำสั่งที่จะถูกเรียกใช้เมื่อโปรแกรม ต้องการแสดงผลให้กับผู้ใช้งาน ผ่านทางแถบสถานะ(Status Bar) ของหน้าจอ

- **Libraries** เป็นส่วนของชุดคำสั่งที่พัฒนาด้วยภาษา C/C++ โดยแบ่งชุดคำสั่งออกเป็นกลุ่มตามวัตถุประสงค์ของการใช้งาน เช่น Surface Manage จัดการเกี่ยวกับการแสดงผล, Media Framework จัดการเกี่ยวกับการการแสดงภาพและเสียง, Open GL | ES และ SGL จัดการเกี่ยวกับภาพ 3มิติ และ 2มิติ, SQLite จัดการเกี่ยวกับระบบฐานข้อมูล เป็นต้น
- **Android Runtime** จะมี Dalvik Virtual Machine ที่ถูกออกแบบมา เพื่อให้ทำงานบนอุปกรณ์ที่มีหน่วยความจำ (Memory), หน่วยประมวลผลกลาง (CPU) และพลังงาน (Battery)ที่จำกัด ซึ่งการทำงานของ Dalvik Virtual Machine จะทำการแปลงไฟล์ที่ต้องการทำงาน ไปเป็นไฟล์ .DEX ก่อนการทำงาน เหตุผลก็เพื่อให้มีประสิทธิภาพเพิ่มขึ้นเมื่อใช้งานกับหน่วยประมวลผลกลางที่มีความเร็วไม่มาก ส่วนต่อมาคือ Core Libraries ที่เป็นส่วนรวบรวมคำสั่งและชุดคำสั่งสำคัญๆซึ่งถูกเขียนด้วยภาษาจาวา (Java Language)
- **Linux Kernel** เป็นส่วนหัวใจสำคัญในการจัดการกับบริการหลักของระบบปฏิบัติการ เช่น เรื่องหน่วยความจำ พลังงาน ติดต่อกับอุปกรณ์ต่างๆ ความปลอดภัย ระบบเครือข่าย โดยแอนดรอยด์ได้นำเอาส่วนนี้มาจากระบบปฏิบัติการลีนุกซ์ 2.6 (Kernel 2.6.24) ที่ได้ถูกออกแบบมาเป็นอย่างดี ถ้าจะนำ vanilla kernel ให้สามารถทำงานบนระบบปฏิบัติการแอนดรอยด์ได้นั้น เป็นเรื่องไม่ง่ายนักสำหรับนักพัฒนาระบบสมองกลฝังตัว ดังนั้นทางกูเกิลจึงได้เตรียมซอร์สโค้ดและเคอร์เนลพื้นฐานภายใต้โปรเจ็คชื่อว่า AOSP (Android Open Source Project) เพื่อเป็นแนวทางให้นักพัฒนาสามารถพัฒนาได้สะดวกยิ่งขึ้น

แนวทางการพัฒนา Embedded Android

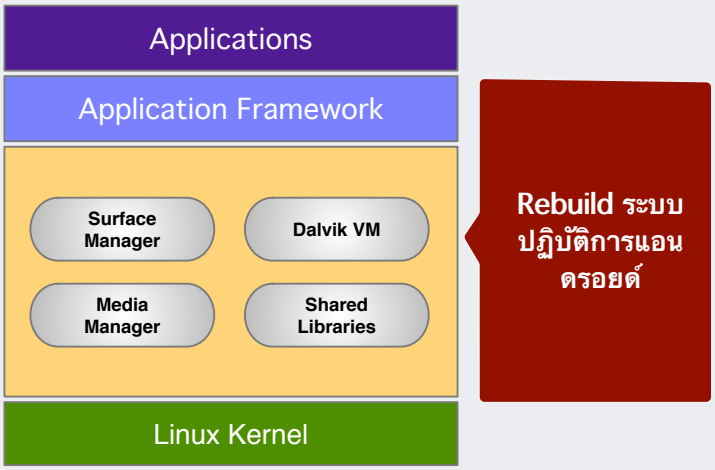
เมื่อวิเคราะห์แนวการพัฒนาในเชิงลึกของนักพัฒนาทางด้านระบบสมองกลฝังตัว โดยเฉพาะทางด้านระบบสมองกลฝังตัว ที่ใช้ระบบปฏิบัติการแอนดรอยด์ สามารถแยกได้ 4 กลุ่มได้แก่

นักพัฒนาโปรแกรม (Application Developers)



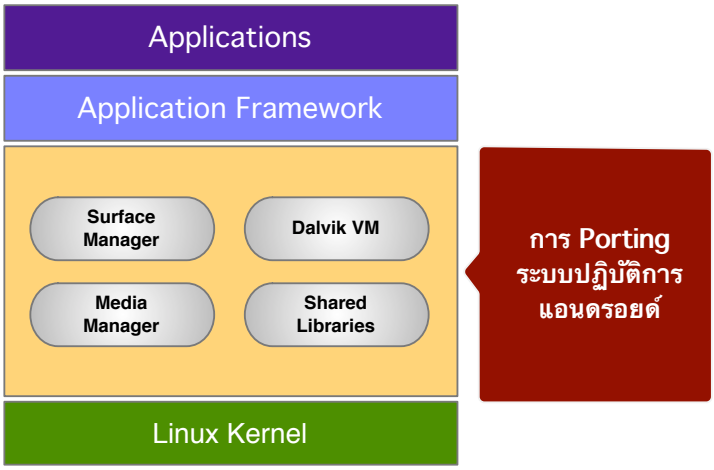
เน้นการพัฒนาโปรแกรมประยุกต์ด้วยภาษาจาวา (Android JAVA) และ ภาษาซี (Native C language) ในระดับ Application Layer เช่น โปรแกรมทางด้านการศึกษา เกมส์ อเนกประสงค์ การสื่อสาร ธุรกิจการเงิน เป็นต้น

นักพัฒนาโปรแกรม (Platform Developers)



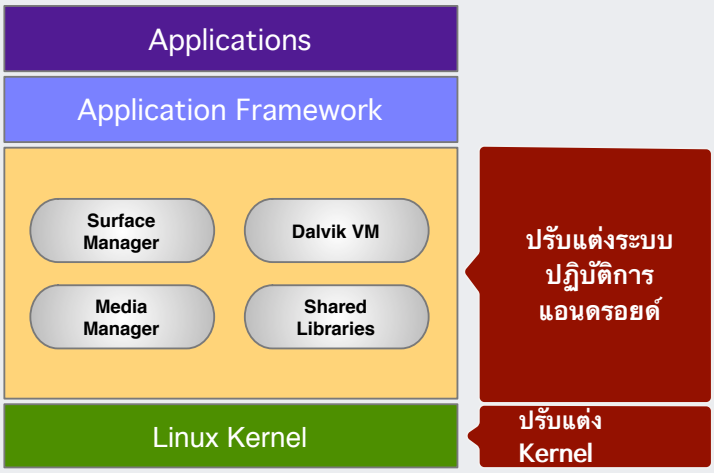
เป็นการปรับแต่ง Android Application Framework เช่น โปรแกรมระบบ โปรแกรมประยุกต์พื้นฐานที่มากับเครื่อง ปรับแต่งการแสดง UI พิเศษ การแสดงผลแบบสองหน้าพร้อมกัน เป็นต้น ซึ่งจะเป็นกลุ่มที่เน้นปรับแต่ง ROMs เช่น CyanogenMod เป็นต้น

นักปรับแต่งระบบปฏิบัติการ (Android Porting)



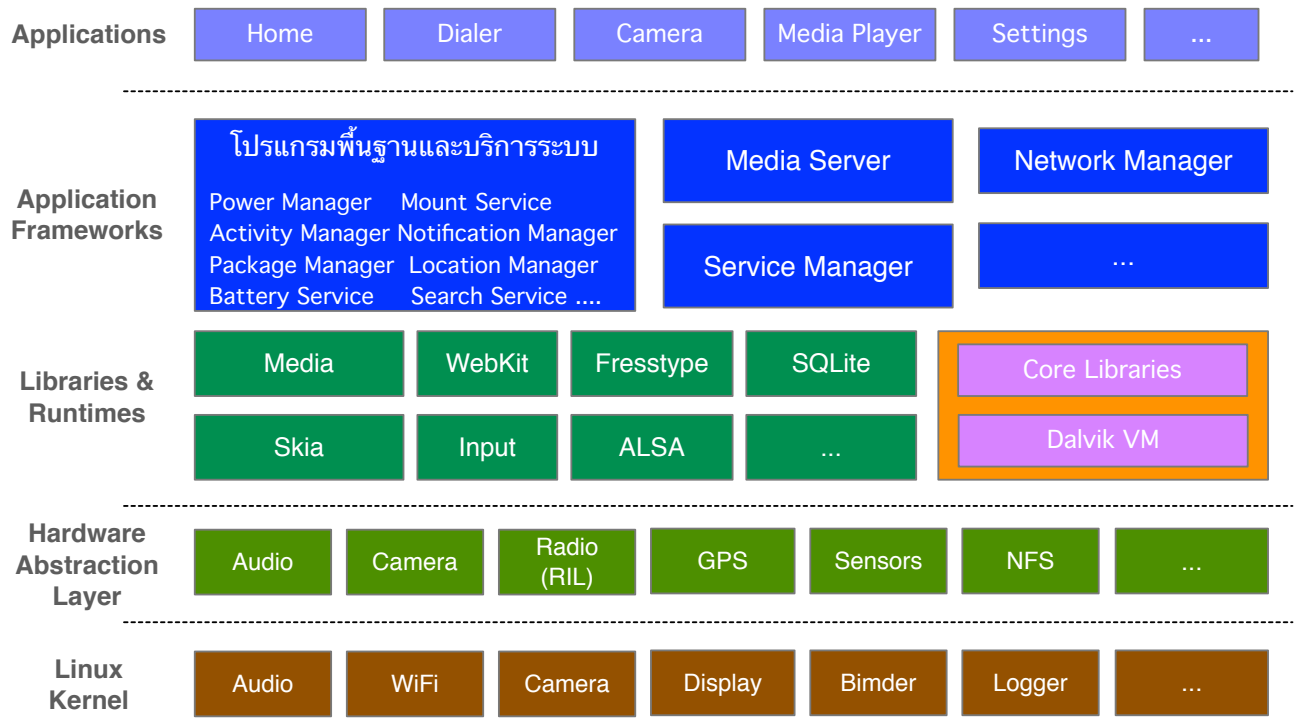
เน้นงานทางด้านการ Porting แอนดรอยด์รุ่นต่างๆ ให้สามารถทำงานอยู่บนสถาปัตยกรรมอื่น เช่น MIPS, Intel x86 เป็นต้น หรือไปอยู่บนเครื่อง Netbook (EeePC) หรือ Virtual Machine (VMWare, Virtual Box, GenyMotion เป็นต้น)

นักระบบฮาร์ดแวร์ (Hardware Developers)



เน้นการออกแบบบอร์ดสมองกลฝังตัว สำหรับงานประยุกต์ด้านต่างๆ เช่น อุปกรณ์ Set-top-box, Android USB Stick, ตู้ Kiosk เป็นต้น โดยจะต้องมีการปรับแต่งลีนุกซ์คอร์เนลเพื่อให้รองรับอุปกรณ์พ่วงรอบข้างตามที่ต้องการ และปรับแต่งโปรแกรมระบบ, Launcher UI หรือการเล่นมัลติมีเดียแบบพิเศษ เป็นต้น

เตรียมสภาพแวดล้อมสำหรับการพัฒนา Embedded Android



รูปที่ 5.4 รายละเอียดภายในแต่ละชั้นของระบบปฏิบัติการแอนดรอยด์

เตรียมสภาพแวดล้อมบนเครื่อง Host

ระบบปฏิบัติการลีนุกซ์ตัวที่เหมาะสำหรับการศึกษาและทดสอบระบบปฏิบัติการแอนดรอยด์คือ Ubuntu LTS (Long Term Support) รุ่น 10.04 หรือ Ubuntu LTS รุ่น 12.04 ซึ่งทางกูเกิลได้ใช้เป็นตัวหลักในการทดสอบระบบปฏิบัติการแอนดรอยด์รุ่นต่างๆโดยเฉพาะรุ่น Android Gingerbread (2.3.x) ขึ้นไปนั้นจะต้องใช้สภาพแวดล้อมที่เป็นระบบปฏิบัติการแบบ 64 บิตเท่านั้น นอกจากนั้นทรัพยากรพื้นฐานของเครื่อง Host ก็ควรจะมีหน่วยความจำไม่น้อยกว่า 8GB (แนะนำอยู่ที่ 16GB) และมีพื้นที่ความจุว่างไม่น้อยกว่า 30GB ตัวอย่างเช่น Android IceScream Sandwich (ICS) จะใช้พื้นที่ประมาณ 25GB สำหรับหนึ่งแพลตฟอร์ม ในกรณีต้องการสร้างทุกรุ่นของ AOSP จะต้องมีพื้นที่ไม่น้อยกว่า 80GB สำหรับโปรแกรมและไลบรารีที่ต้องติดตั้งลงไป สำหรับโปรแกรมและเครื่องมือพื้นฐานที่ต้องมีอยู่ในเครื่อง Host ตัวอย่างเช่น

- โปรแกรม Python รุ่น 2.6 - 2.7
- เครื่องมือในการคอมไพล์ GNU Make รุ่น 3.81 - 3.82
- โปรแกรม JDK รุ่น 6 สำหรับ Android Gingerbread ขึ้นไป แต่ถ้าต้องการ Android Froyo ต่ำลงมา จะต้องลง JDK รุ่น 5 (*fromjava.sun.com*)
- โปรแกรม Git รุ่น 1.7 ขึ้นไป (git-scm.com)

ตัวอย่างการติดตั้ง JDK รุ่น 6 สำหรับ Gingerbread ขึ้นไป และ JDK รุ่น 5 สำหรับ Froyo ต่ำลงมา

```
$ sudo add-apt-repository "deb http://archive.canonical.com/ lucid part-
ner"
$ sudo apt-get update
$ sudo apt-get install openjdk-6-jre
```

หรือ

```
$ sudo add-apt-repository "deb http://archive.ubuntu.com/ubuntu hardy
main multiverse"
$ sudo add-apt-repository "deb http://archive.ubuntu.com/ubuntu hardy-
updates main multiverse"
$ sudo apt-get update
$ sudo apt-get install openjdk-6-jre
```

ติดตั้งโปรแกรม และไลบรารีที่เกี่ยวข้องสำหรับระบบปฏิบัติการ 64-bit Ubuntu LTS 10.04

```
$ sudo apt-get install git-core gnupg flex bison gperf build-essential \
  zip curl zlib1g-dev libc6-dev lib32ncurses5-dev ia32-libs \
  x11proto-core-dev libx11-dev lib32readline5-dev lib32z-dev \
  libgl1-mesa-dev g++-multilib mingw32 tofrodos python-markdown \
  libxml2-utils xsltproc
```

ในกรณีที่ใช้ระบบปฏิบัติการ 64-bit Ubuntu LTS 12.04

```
$ sudo apt-get install git gnupg flex bison gperf build-essential \
zip curl libc6-dev libncurses5-dev:i386 x11proto-core-dev \
libx11-dev:i386 libreadline6-dev:i386 libgl1-mesa-glx:i386 \
libgl1-mesa-dev g++-multilib mingw32 tofrodos \
python-markdown libxml2-utils xsltproc zlib1g-dev:i386

$ sudo ln -s /usr/lib/i386-linux-gnu/mesa/libGL.so.1
/usr/lib/i386-linux-gnu/libGL.so
```

Android Open Source Project (AOSP)

ดาวน์โหลดซอร์ส AOSP

ในการดาวน์โหลดซอร์สโค้ดของระบบปฏิบัติการแอนดรอยด์นั้นจะใช้โปรแกรม repo ซึ่งเป็น Git Wrapper script เพื่อระบุรุ่นของแอนดรอยด์ที่ต้องการและดาวน์โหลดโค้ดที่เกี่ยวข้องทั้งหมด โดยปกติแล้วจะใช้พื้นที่ในการเก็บโค้ดทั้งหมดไม่ต่ำกว่า 20GB และอาจจะต้องใช้เวลาในการดาวน์โหลดประมาณ 3-6 ชั่วโมง ขึ้นอยู่กับความเร็วของอินเตอร์เน็ตของเครือข่ายนั้นๆ

ดังตัวอย่างขั้นตอนการดาวน์โหลด AOSP

```
$ mkdir ~/bin
$ export PATH=$PATH:~/bin
$ sudo apt-get install curl
$ curl https://dl-ssl.google.com/dl/googlesource/git-repo/repo > ~/bin/repo
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 22889  100 22889    0     0  50164      0 --:--:-- --:--:-- --:--:--  111k

$ chmod a+x ~/bin/repo
```

ไฟล์ manifest จะเก็บลิงค์และข้อมูลที่จำเป็นสำหรับดาวน์โหลดโปรแกรมที่เกี่ยวข้องทั้งหมด ดังตัวอย่างแสดงภายในไฟล์ XML ข้างล่าง

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest>

  <remote  name="aosp"
           fetch=".." />
  <default revision="ics-mr0"
           remote="aosp"
           sync-j="4" />

  <project path="build" name="platform/build">
    <copyfile src="core/root.mk" dest="Makefile" />
  </project>
  <project path="abi/cpp" name="platform/abi/cpp" />
  <project path="bionic" name="platform/bionic" />
  <project path="bootable/bootloader/legacy" name="platform/bootable/bootloader/legacy" />
  <project path="bootable/diskinstaller" name="platform/bootable/diskinstaller" />
  <project path="bootable/recovery" name="platform/bootable/recovery" />
  <project path="cts" name="platform/cts" />
```

ตาราง 5.3 รายละเอียดภายใน XML ไฟล์ของ AOSP

ชื่อ	รายละเอียด
name	ชื่อโปรเจค ที่บรรจุโค้ดทั้งหมด
fetch	Server URL
revision	git branch
remote	รายละเอียด server
path	ไดเรกทอรีที่จะเก็บจากการ unpack
project	รายละเอียดของโปรเจค

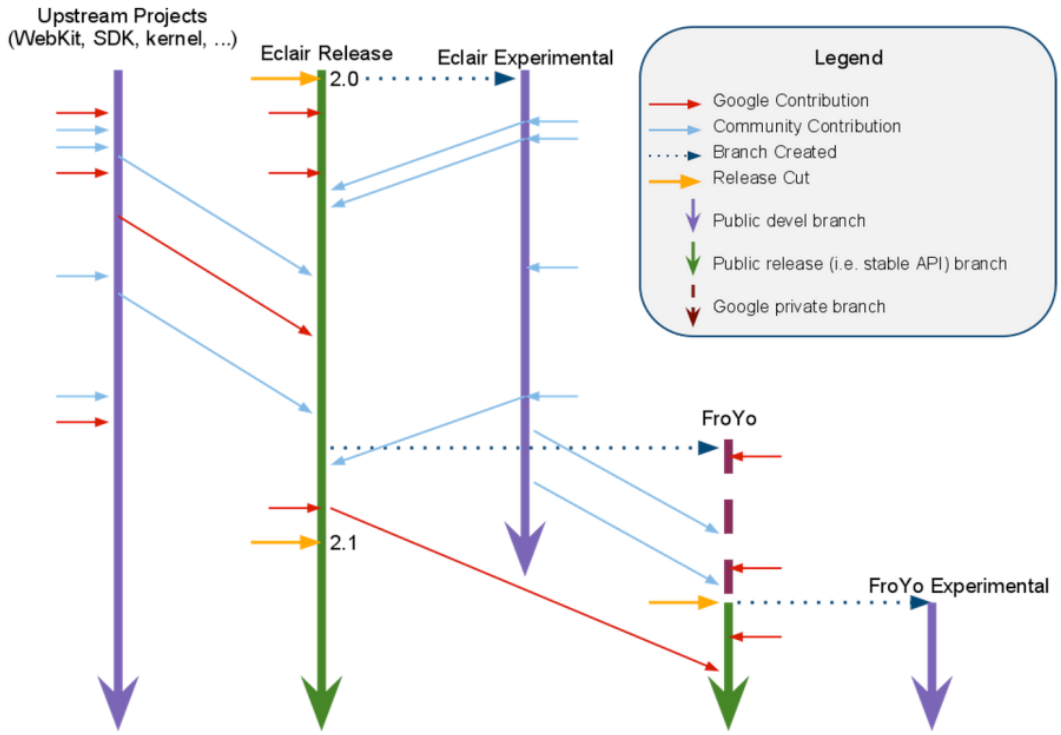
ดาวน์โหลด AOSP รุ่นล่าสุด สามารถใช้คำสั่ง

```
$ repo init -u https://android.googlesource.com/platform/manifest
```

ในกรณีที่ต้องการระบุรุ่น เช่น รุ่น 4.0.1_r1 ก็สามารถทำโดยการเพิ่ม option -b ดังคำสั่งข้างล่าง

```
$ repo init -u https://android.googlesource.com/platform/manifest -b
android-4.0.1_r1
```

จากรูปข้างล่าง แสดงให้เห็นแนวคิดของ AOSP ในการจัดการและปล่อยโค้ดแต่ละรุ่น



รูปที่ 5.5 แสดงแนวทางการออกเวอร์ชันของแต่ละรุ่น

เริ่มต้นการดาวน์โหลดโค้ด AOSP ลงในไดเรกทอรี ~/aosp ซึ่งใช้เวลาในการดาวน์โหลดไม่น้อยกว่า 3-5 ชั่วโมง

```
$ mkdir ~/aosp
$ cd ~/aosp
$ repo sync
```

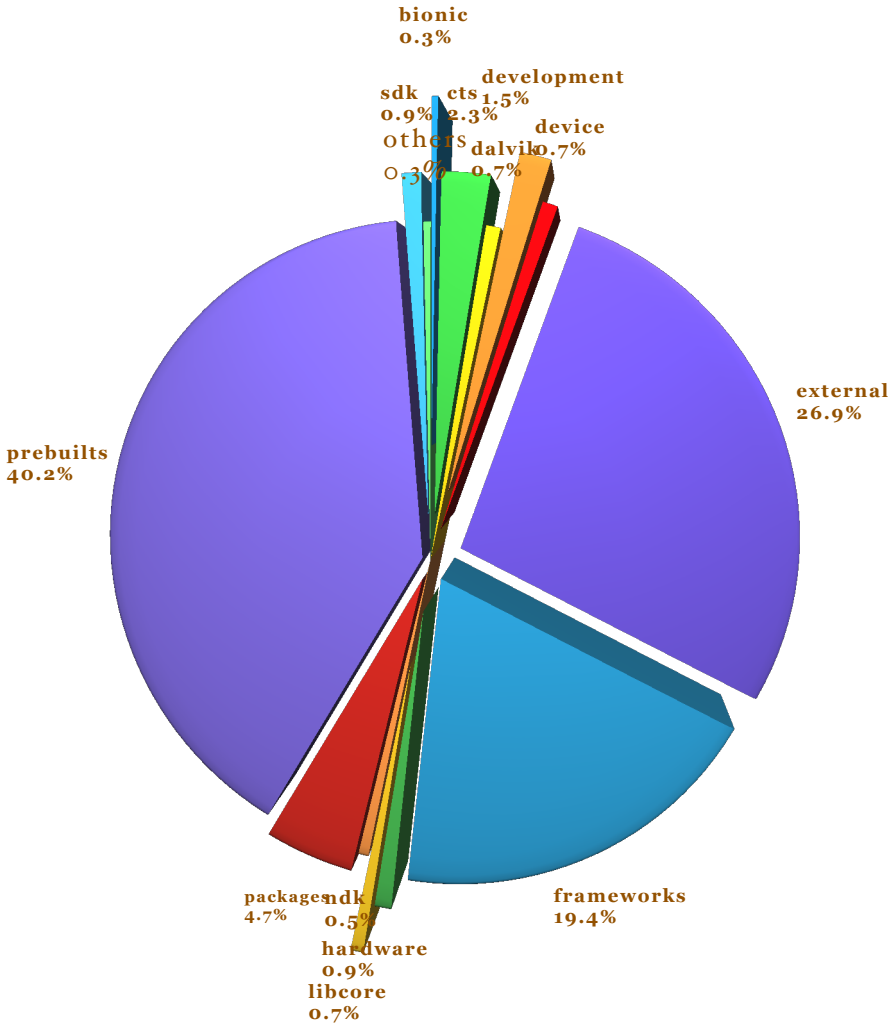
โครงสร้างภายใน AOSP

เมื่อดาวน์โหลดไฟล์ทั้งหมดเสร็จสิ้น ภายในไดเรกทอรี ~/aosp จะมีไดเรกทอรีย่อยๆไม่น้อยกว่า 35,000 ไดเรกทอรี สำหรับไดเรกทอรีสำคัญๆที่นักพัฒนาควรจะทำความเข้าใจ ได้แก่

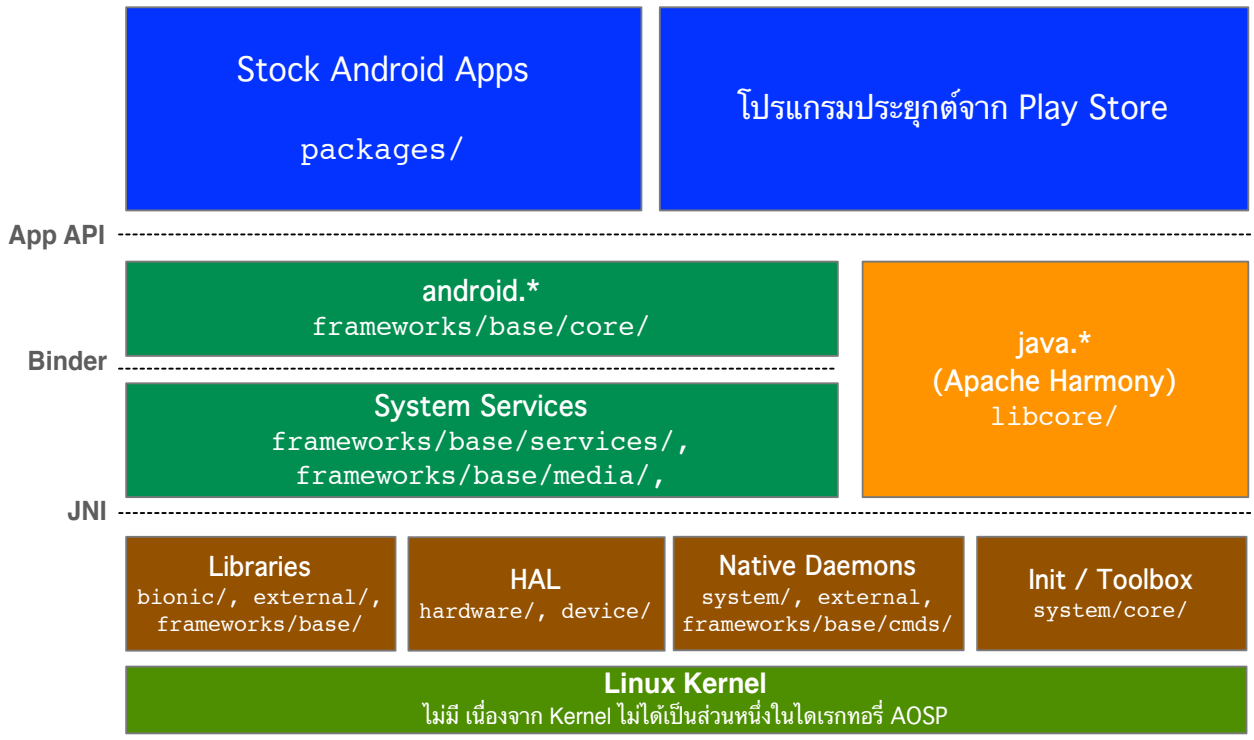
ตาราง 5.4 รายละเอียดไดเรกทอรีของ AOSP

ไดเรกทอรี	รายละเอียด
abi	บรรจุ Minimal C++ Run-time Type information
bionic	ไลบรารีภาษา C ของแอนดรอยด์
bootable	บรรจุโค้ดสำหรับ Bootloader และ Recovery
build	บรรจุโครงสร้าง framework สำหรับสร้าง AOSP
cts	ชุด Test Suite สำหรับแอนดรอยด์
dalvik	บรรจุตัว Dalvik Virtual Machine
development	ชุดเครื่องมือสำหรับพัฒนาต่างๆ
device	บรรจุไฟล์ configuration ต่างๆของแต่ละผู้ขายตัว System-on-Chip
external	โปรเจ็คภายนอกที่ถูกนำมาใช้ใน AOSP
frameworks	Android core application framework
hardware	ไลบรารีและโมดูลต่างๆสำหรับระบบฮาร์ดแวร์ (HAL)
kernel	Linux Kernel
libcore	บรรจุโปรเจ็ค Apache Harmony และไลบรารีพื้นฐานของจาวาที่ถูกใช้ใน Dalvik
ndk	Native Development Kit
packages	โปรแกรมประยุกต์ต่างๆภายใน AOSP
prebuilt	เครื่องมือ Toolchain สำเร็จรูป
sdk	Android's Software Development Kit
system	ไลบรารีและไฟล์ไบนารีหลักๆของระบบแอนดรอยด์ เช่น init, toolbox, adb
tools	เครื่องมือต่างๆสำหรับการตั้งค่าและใช้งาน

เมื่อคิดเป็นสัดส่วนเปอร์เซ็นต์เปรียบเทียบขนาดแต่ละไดเรกทอรี สามารถแสดงเป็นกราฟวงกลมได้ดังรูปข้างล่างนี้



รูปที่ 5.6 แสดงขนาดพื้นที่ของแต่ละไดเรกทอรีของ AOSP



รูปที่ 5.7 ไดเรกทอรีที่เกี่ยวข้องของแต่ละชั้นของระบบปฏิบัติการแอนดรอยด์

สิทธิ์อนุญาต (License) ของแต่ละส่วนภายใน AOSP สามารถแยกประเภทของลิขสิทธิ์ที่คุ้มครองอยู่ได้ 4 กลุ่ม ได้แก่

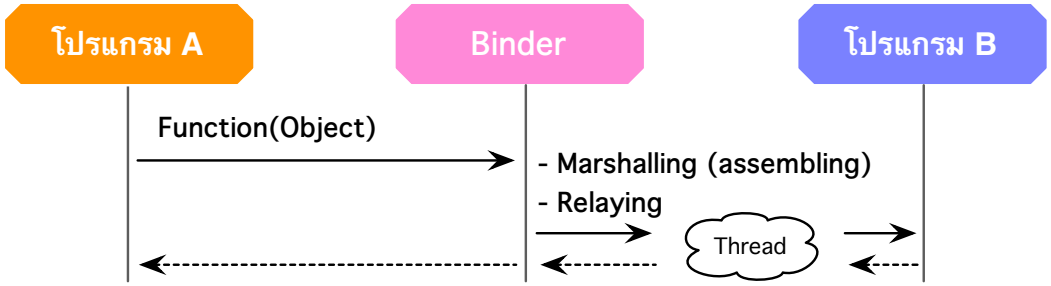
- ส่วน Kernel จะมี GNU GPLv2/GPLv3 license คุ้มครอง
- ส่วน external libraries จะมี LGPL และ GPL license คุ้มครอง
- ส่วน bionic และ toolbox จะมี BSD license คุ้มครอง
- ส่วน framework จะมี Apache 2.0 license คุ้มครอง

Android Kernel

โค้ดของ Android Kernel patches ไม่ได้ถูกนำเข้าไปไว้ในระบบกลางของลีนุกซ์คอร์เนลหลัก เนื่องจากทางบริษัทกูเกิลเป็นผู้ดูแลเองทั้งหมดเรียกในส่วนนี้ว่า “android-common” โดยส่วน Android Kernel patches จะวางอยู่ในตำแหน่งระดับบนของพื้นฐานระบบปฏิบัติการตามข้อกำหนดของ Linus Torvalds

ส่วนเคอร์เนลที่บริษัทกูเกิลได้พัฒนาขึ้นเองนั้น ก็เพื่อต้องการเพิ่มความสามารถที่ขาดอยู่และปรับปรุงการทำงานของลีนุกซ์คอร์เนลเดิมบางส่วนให้เหมาะสมกับทรัพยากรที่มีอยู่จำกัด สภาพแวดล้อมและพฤติกรรมการใช้งานอุปกรณ์ชนิดพกพา (Portable device) จนเรียกว่าเป็นแอนดรอยด์เคอร์เนล (Android Kernel) เฉพาะขึ้นมา โดยมีรายละเอียดดังนี้

- **Binder** - กลไกการติดต่อสื่อสารระหว่างโปรเซสภายในระบบปฏิบัติการแอนดรอยด์ (Android's IPC mechanism) ซึ่งได้แรงบันดาลใจมาจากโปรเจค Open Binder



รูปที่ 5.8 แสดงขั้นตอนการรับส่งข้อมูลระหว่างโปรแกรมผ่าน Binder

- **Ashmem** (Anonymous Shared Memory) - ทำหน้าที่เป็นตัวจัดสรรพื้นที่ที่ต้องใช้ร่วมกันภายในหน่วยความจำกลาง (Shared memory allocator) โดยจะถอดส่วนพื้นที่ที่ถูกแชร์ไว้ในกรณีที่หน่วยความจำเริ่มวิกฤติ
- **Logger** - ส่วนโปรแกรมไดรเวอร์ที่บันทึกข้อมูลต่างๆจากการใช้งานของผู้ใช้ (user space)
- **Wakelocks** - ส่วนควบคุมไม่ให้เครื่องเข้าสู่โหมดพัก (suspend) ในกรณีที่พลังงานไฟฟ้าน้อยหรือมีการพับปิดหน้าจอเหมือนเครื่องคอมพิวเตอร์ทั่วไปที่ระบบปฏิบัติการจะปิดเครื่องแล้วเข้าสู่โหลด suspend แต่ Wakelocks จะทำงานเพียงแค่ดับหน้าจอ (Screen Turn-Off) เมื่อผู้ใช้ต้องการใช้งานต่อก็เพียงเปิดหน้าจอกลับมา (Screen Turn-On) โดยที่เครื่องไม่ได้ดับ



รูปที่ 5.9 ระดับการควบคุมไม่ให้เครื่องเข้าสู่โหมดพัก

- **OOM** (Out Of Memory handler) - ตัวจัดการปัญหาในกรณีหน่วยความจำเต็ม โดยจะมีการส่งสัญญาณ Triggers ก่อนจะส่งต่อให้เคอร์เนลดำเนินการจัดการ
- **Alarm** - ตัวนับหน่วยเวลาเพื่อแจ้งเตือน (Android's alarm timer) ซึ่งพัฒนาต่อยอดมาจากตัวนาฬิกา RTC (Real-time Clock) ของลีนุกซ์คอร์เนลเดิม
- **RAM console** - ตัวแสดงบันทึกข้อมูลการทำงานของเคอร์เนลภายในโดยการเรียกผ่านไฟล์ /proc/last_kmsg
- **Paranoid network** - ตัวดูแลจัดการระบบเครือข่ายผ่าน uid (user id) และ pid (process id) เช่น การกำหนดให้ uid หรือ pid มีสิทธิ์ในการเข้าถึง (access permission) ระบบเครือข่ายอินเทอร์เน็ตด้วยการตั้งค่าภายในไฟล์ manifest.xml ดังตัวอย่างข้างล่างนี้

```
<permission name="android.permission.INTERNET">
        <group gid="inet" />
</permission>
```

ขั้นตอนการคอมไพล์ AOSP มาเป็นระบบปฏิบัติการแอนดรอยด์

ตั้งค่าพื้นฐานสำหรับเตรียมพร้อมสภาพแวดล้อมในการคอมไพล์ AOSP

```
$ cd ~/aosp
$ source build/envsetup.sh
```

จากคำสั่งด้านบนเป็นคำสั่งใช้ในการตั้งค่าสภาพแวดล้อมให้กับระบบเพื่อจำเป็นต่อการคอมไพล์ และเพิ่มคำสั่ง shell script เพื่อใช้ในการเลือกแพลมฟอร์มที่ต้องการ เช่น

- `hmm()` – สำหรับแสดงมนูช่วยเหลือ
- `lunch()` – สำหรับแสดง lunch เมนูเพื่อเลือกแพลมฟอร์มที่ต้องการ (target board)
- `add_lunch_combo()` – สำหรับเพิ่มชื่อผลิตภัณฑ์ใหม่ที่นักพัฒนากำหนดขึ้นเอง
- `mm()` – สำหรับสร้างโมดูลทั้งหมดที่อยู่ในไดเรกทอรีปัจจุบัน

นอกจากนั้นในระหว่างเรียก `envsetup.sh` ก็จะมีการรันคำสั่ง `source vendorsetup.sh, vendor/*/vendorsetup.sh, vendor/*/*/vendorsetup.sh` และ `device/*/*/vendorsetup.sh` เพื่ออนุญาตให้เพิ่มรายชื่อผลิตภัณฑ์ที่นักพัฒนากำหนดขึ้นเองอีกด้วย ตัวอย่างเช่น `Samsung Maguro product` ที่อยู่ในไดเรกทอรี `device/samsung/maguro`

```
#
# Copyright (C) 2011 The Android Open Source Project
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
add_lunch_combo full_maguro-userdebug
```

ทดสอบเรียกคำสั่ง shell script ที่ถูกตั้งค่าจากไฟล์ envsetup.sh

```
$ hmm

Invoke ". build/envsetup.sh" from your shell to add the following functions to your
environment:
- croot:   Changes directory to the top of the tree.
- m:       Makes from the top of the tree.
- mm:      Builds all of the modules in the current directory.
- mmm:     Builds all of the modules in the supplied directories.
- cgrep:   Greps on all local C/C++ files.
- jgrep:   Greps on all local Java files.
- resgrep: Greps on all local res/*.xml files.
- godir:   Go to the directory containing a file.

Look at the source to view more functions. The complete list is:
addcompletions add_lunch_combo cgrep check_product check_variant choosecombo choose-
product choosetype choosevariant cproj croot findmakefile gdbclient
get_abs_build_var getbugreports get_build_var getprebuilt gettop godir hmm is-
viewserverstarted jgrep key_back key_home key_menu lunch _lunch m mm mmm pid print-
config print_lunch_menu resgrep runhat runtest set_java_home setpaths set_se-
quence_number set_stuff_for_environment settitle smoketest startviewserver
stopviewserver systemstack tapas tracedmdump
```

ขั้นตอนถัดไปเป็นการเลือกบอร์ด target (ซึ่งในที่นี้จะเลือกสถาปัตยกรรม x86) ด้วยคำสั่ง

```
$ lunch
You're building on Linux
Lunch menu... pick a combo:
```

```
     1. full-eng
     2. full_x86-eng
     3. vbox_x86-eng
     4. full_grouper-userdebug
     5. mini_armv7a_neon-userdebug
     6. mini_armv7a-userdebug
     7. full_wingray-userdebug
     8. full_crespo-userdebug
     9. full_maguro-userdebug
     10. full_panda-userdebug

Which would you like? [full-eng] full_x86-eng

============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=4.1.1
TARGET_PRODUCT=full_x86
TARGET_BUILD_VARIANT=eng
TARGET_BUILD_TYPE=release
TARGET_BUILD_APPS=
TARGET_ARCH=x86
TARGET_ARCH_VARIANT=x86-atom
HOST_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-3.0.0-burapha-x86_64-with-Ubuntu-10.04-lucid
HOST_BUILD_TYPE=release
BUILD_ID=JRO03L
OUT_DIR=out
============================================
```

หลังจากเรียกคำสั่งข้างต้น การตั้งค่าผลิตภัณฑ์ที่นักพัฒนาเลือกไว้ จะถูกตั้งเป็นที่เรียบร้อยแล้ว รวมทั้งค่าตัวแปรสำคัญต่างๆ ได้แก่

- ANDROID_BUILD_TOP
- ANDROID_TOOLCHAIN
- ANDROID_PRODUCT_OUT
- ANDROID_HOST_OUT

สามารถตรวจสอบค่าตัวแปรสภาพแวดล้อมของระบบด้วยคำสั่ง

```
$ echo $ANDROID_BUILD_TOP
```

ก่อนเข้าสู่ขั้นตอนคอมไพล์ AOSP ที่ใช้เวลาอย่างน้อย 3-4 ชั่วโมงซึ่งขึ้นอยู่กับความเร็วอินเทอร์เน็ตนั้น จะมีอยู่อีก 2 คำสั่งที่นักพัฒนาควรรู้ได้แก่ คำสั่งการเก็บแคสซ (cache) อ๊อฟเจ็คไฟล์ที่ได้มาจากการคอมไพล์โดยเผื่อไว้สำหรับครั้งถัดไปที่ต้องมีการคอมไพล์ใหม่อีกรอบ โดยการตั้งค่าดังนี้

```
$ export USE_CCACHE=1
```

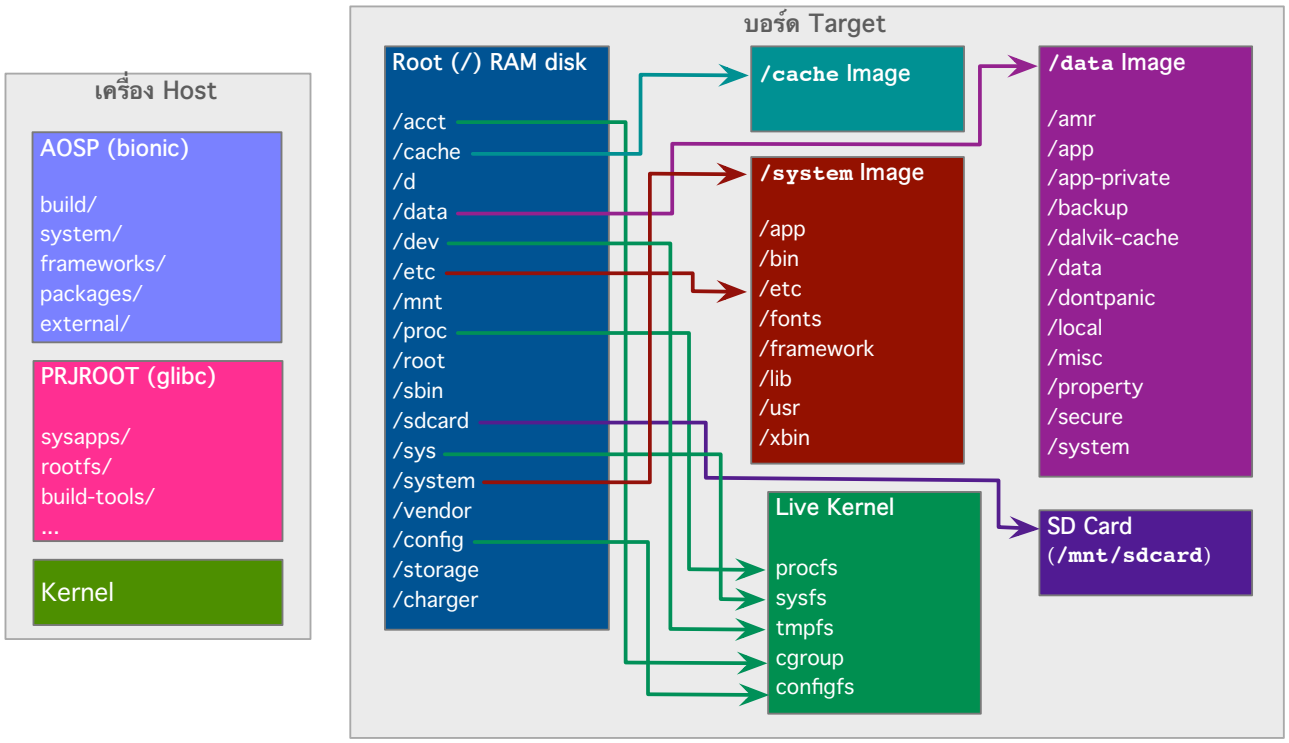
และการตั้งค่าจำนวนคอร์ของหน่วยประมวลผลที่จะนำมาใช้ระหว่างการคอมไพล์ เพื่อเพิ่มความเร็วยิ่งขึ้น โดยการระบุ option **-jX** โดยที่ **X** จะแทนจำนวนของคอร์ (Core) หน่วยประมวลผลกลางบวก 1 ดังตัวอย่างการใช้งานข้างล่าง

```
$ make -j3
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=4.1.1
TARGET_PRODUCT=full_x86
TARGET_BUILD_VARIANT=eng
TARGET_BUILD_TYPE=release
TARGET_BUILD_APPS=
TARGET_ARCH=x86
TARGET_ARCH_VARIANT=x86-atom
HOST_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-2.6.32-42-server-x86_64-with-Ubuntu-10.04-lucid
HOST_BUILD_TYPE=release
BUILD_ID=JRO03L
OUT_DIR=out
============================================
find: `src': No such file or directory
PRODUCT_COPY_FILES
frameworks/base/data/sounds/F1_MissedCall.ogg:system/media/audio/notifications/F1_Mi
ssedCall.ogg ignored.
PRODUCT_COPY_FILES
frameworks/base/data/sounds/F1_New_MMS.ogg:system/media/audio/notifications/F1_New_M
MS.ogg ignored.
.
.
.
PRODUCT_COPY_FILES
frameworks/base/data/sounds/effects/camera_click.ogg:system/media/audio/ui/camera_cl
ick.ogg ignored.
```

ผลลัพธ์ของไฟล์ที่ได้จากการคอมไพล์มีอย่างน้อย 6 ไฟล์ได้แก่

1. **boot.img** – Native system boot image
2. **ramdisk.img** – Ramdisk **rootfs**
3. **recovery.img** – Recovery image
4. **ramdisk-recovery.img** – Ramdisk **rootfs** สำหรับการกู้คืน (recovery)
5. **system.img** – System data (ไดเรกทอรี **/system**)
6. **userdata.img** – User data (ไดเรกทอรี **/data**)

ไฟล์ข้างต้นเหล่านี้เป็นเพียงส่วนหนึ่งของระบบปฏิบัติการแอนดรอยด์ ที่ได้มาจาก AOSP แต่ยังไม่ได้รวมถึง Android Kernel ซึ่งจะต้องดาวน์โหลดและคอมไพล์แยกออกไปจาก AOSP ซึ่งจะกล่าวถึงในบทถัดไป จากรูปข้างล่างแสดงตำแหน่งไฟล์ที่จะถูกนำไปใส่ไว้ในบอร์ด target ตามตำแหน่งไดเรกทอรีดังอธิบายไว้ในรูป



รูปที่ 5.10 แสดงรายละเอียดของแต่ละไดเรกทอรีภายในระบบไฟล์ของแอนดรอยด์

ตาราง 5.5 แสดงตำแหน่งการถูก mount ของแต่ละไฟล์ image ในแอนดรอยด์

IMAGE FILE	MTD PARTITION	MOUNT POINT
ramdisk.img	N/A	/
system.img	/dev/mtdblock0	/system
userdata.img	/dev/mtdblock1	/data
N/A	/dev/mtdblock2	/cache

รายละเอียดระบบไฟล์ภายในระบบปฏิบัติการแอนดรอยด์ ซึ่งประกอบไปด้วยไดเรกทอรีและไฟล์ที่สำคัญต่างๆ ดังอธิบายอยู่ในตารางข้างล่างนี้

ตาราง 5.6 รายละเอียดไดเรกทอรีภายในระบบปฏิบัติการแอนดรอยด์

ชื่อไฟล์	ประเภท	รายละเอียด
/acct	ไดเรกทอรี	จุด mount-point ไปยัง cgroup
/cache	ไดเรกทอรี	ส่วนเก็บข้อมูลชั่วคราวต่างๆ เช่นการดาวน์โหลด เป็นต้น
/d	symlink	ชี้ไปยัง /sys/kernel/debug ซึ่งเป็นจุด mount-point สำหรับ debugfs
/data	ไดเรกทอรี	จุด mount-point ไปยังพาร์ทิชัน data ซึ่งเป็นจุด mount ของไฟล์ userdata.img

ชื่อไฟล์	ประเภท	รายละเอียด
/dev	ไดเรกทอรี	จุด mount-point ไปยัง tmpfs ซึ่งบรรจุ device node ต่างๆ
/etc	symlink	ชี้ไปยัง /system/etc
/mnt	ไดเรกทอรี	จุดสำหรับ mount ระบบไฟล์ชั่วคราว
/proc	ไดเรกทอรี	จุด mount-point ไปยัง procfs
/root	ไดเรกทอรี	ไดเรกทอรีของผู้ดูแลระบบ root เดิมของ Linux แต่ใน Android จะไม่มีบรรจุไฟล์ใดๆ
/sbin	ไดเรกทอรี	โดยปกติใน Linux จะเก็บโปรแกรมสำหรับผู้ดูแลระบบ แต่ใน Android จะเก็บเพียง ueventd และ adbd
/sdcard	ไดเรกทอรี	จุดสำหรับ mount ระบบไฟล์บน SD Card
/sys	ไดเรกทอรี	จุด mount-point ไปยัง sysfs
/system	ไดเรกทอรี	จุด mount-point ไปยังพาร์ทิชัน system ซึ่งเป็นจุด mount ของไฟล์ system.img
/vendor	symlink	ชี้ไปยัง /system/vendor ซึ่งอาจจะพบเจอได้ในบางเครื่อง
/config	ไดเรกทอรี	(ตั้งแต่ Jelly Bean 4.2) จุด mount-point ไปยัง configfs
/storage	ไดเรกทอรี	(ตั้งแต่ Jelly Bean 4.1) สำหรับ mount ตัวเก็บข้อมูลภายนอก (External Storage)
/charger	ไฟล์	(ตั้งแต่ Jelly Bean 4.2) เป็นโปรแกรมแสดงสถานะของแบตเตอรี่
/res	ไดเรกทอรี	(ตั้งแต่ Jelly Bean 4.2) ไฟล์ resources สำหรับโปรแกรม charger
/init	ไดเรกทอรี	โปรแกรม init ที่จะถูกเรียกโดย Kernel หลังจากสิ้นสุดการเริ่มต้นระบบครั้งแรก
/init.rc	ไฟล์	ไฟล์ configuration สำหรับโปรแกรม init
/init<device_name>.rc	ไฟล์	ไฟล์ configuration ของบอร์ด target รุ่นนั้นๆ สำหรับโปรแกรม init
/ueventd.rc	ไฟล์	ไฟล์ configuration สำหรับโปรแกรม uventd
/uventd<device_name>.rc	ไฟล์	ไฟล์ configuration ของบอร์ด target รุ่นนั้นๆ สำหรับโปรแกรม ueventd
/default.prop	ไฟล์	ค่าคุณสมบัติทั่วไปของระบบ (global properties) ซึ่งจะอ่านโดยโปรแกรม init

ระบบปฏิบัติการแอนดรอยด์บน Android Emulator

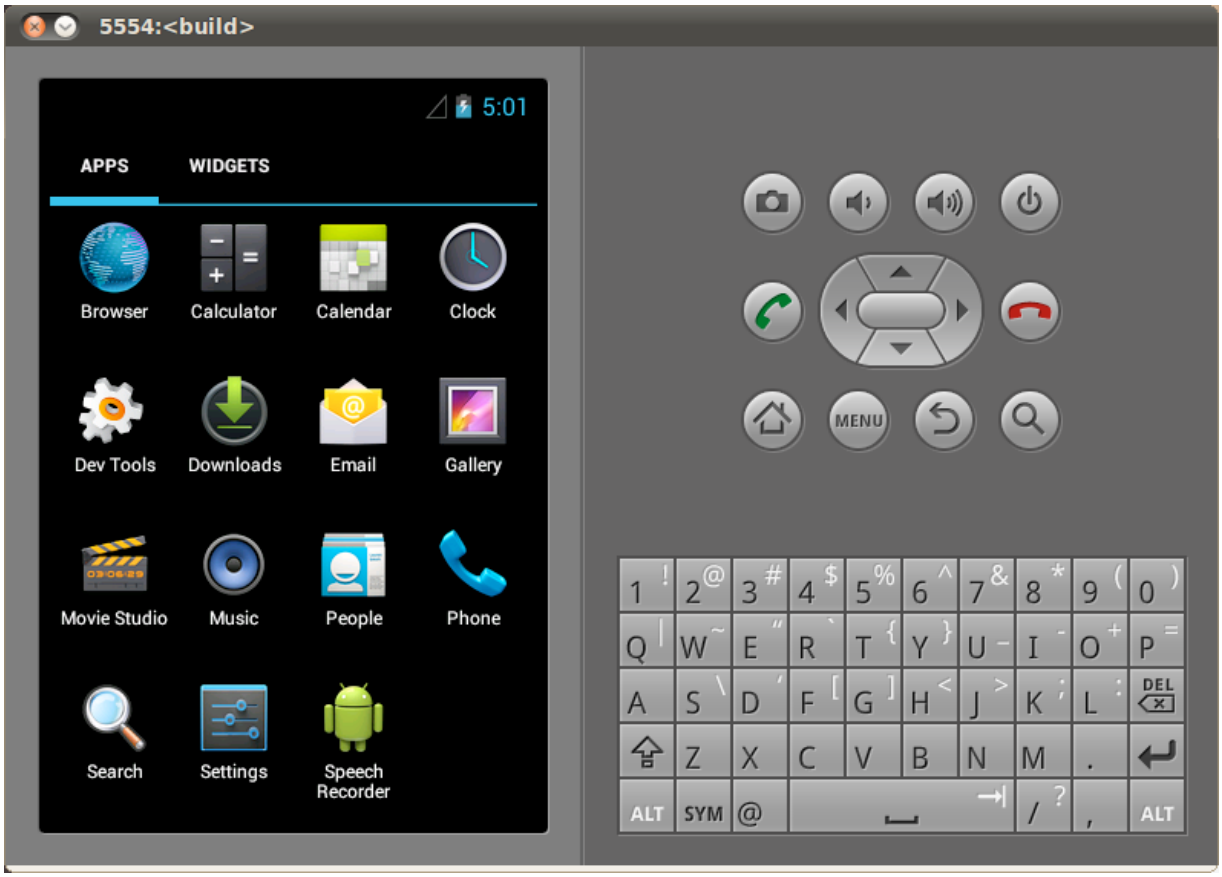
ไฟล์ระบบปฏิบัติการแอนดรอยด์ จะถูกเก็บไว้ในไดเรกทอรีดังแสดงข้างล่างนี้

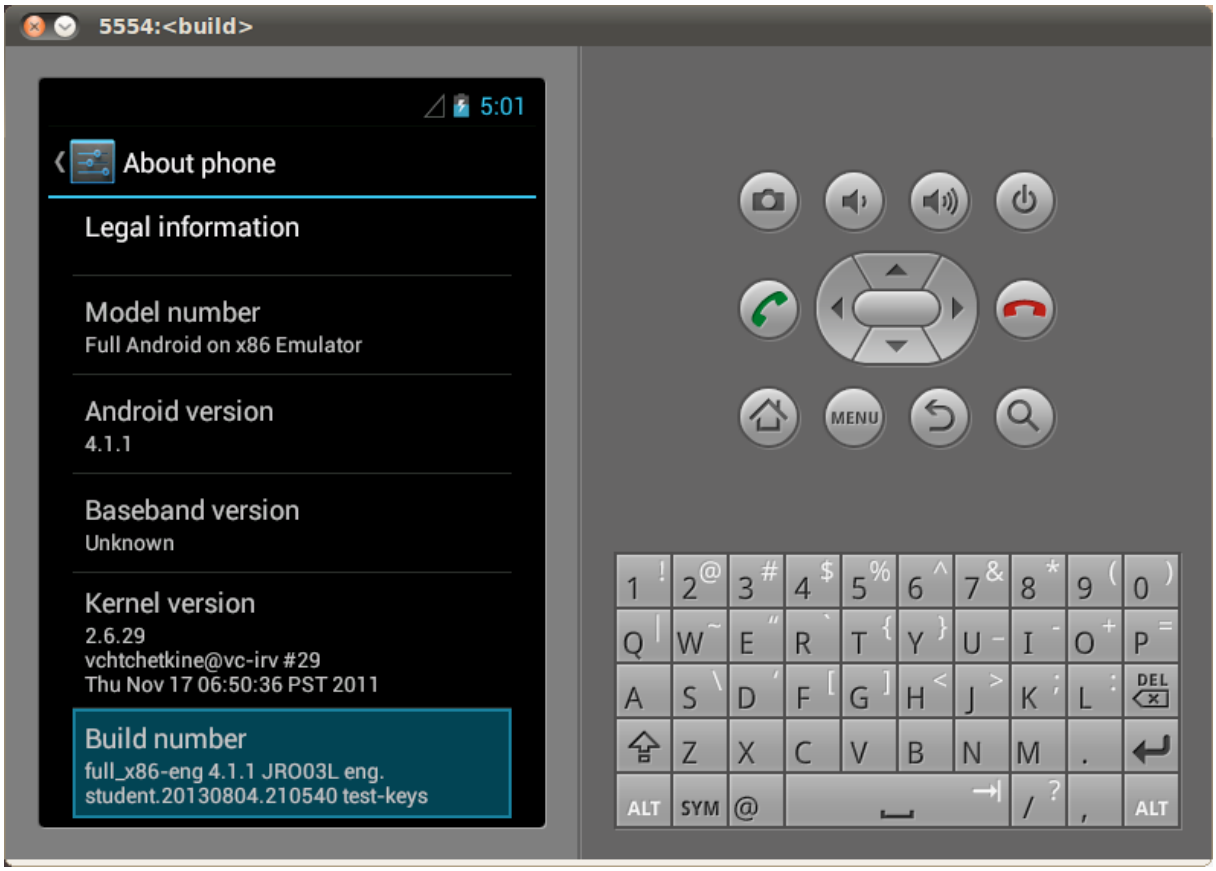
```
$ ls ~/aosp/out/target/product/generic_x86/
android-info.txt  grub              previous_build_config.mk   system
userdata-qemu.img clean_steps.mk      hardware-qemu.ini          ramdisk.img
system.img        data                installed-files.txt        root
test              dex_bootjars        obj                        symbols
userdata.img
```

ดังนั้นผู้ใช้ก็สามารถเปิดโปรแกรม Android Emulator ด้วยคำสั่งข้างล่างนี้ ก็จะขึ้นหน้าต่างโปรแกรมดังแสดงในรูปข้างล่าง

```
$ emulator &
```

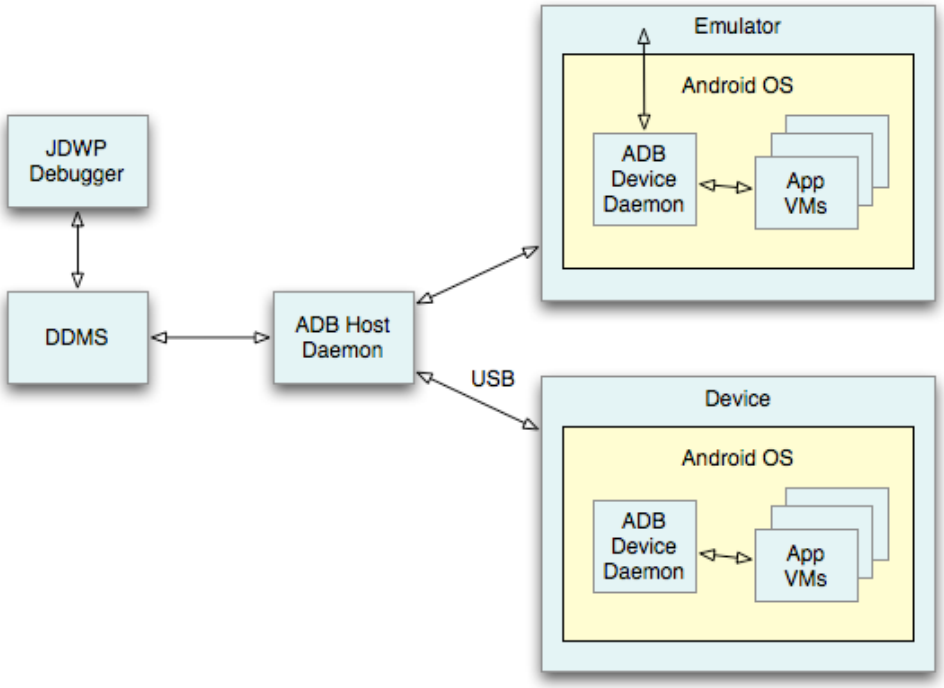



5554:<build>
5:01
APPS
WIDGETS
Browser
Calculator
Calendar
Clock
Dev Tools
Downloads
Email
Gallery
Movie Studio
Music
People
Phone
Search
Settings
Speech Recorder
MENU


5554:<build>
5:01
About phone
Legal information
Model number
Full Android on x86 Emulator
Android version
4.1.1
Baseband version
Unknown
Kernel version
2.6.29
vchtchetkine@vc-irv #29
Thu Nov 17 06:50:36 PST 2011
Build number
full_x86-eng 4.1.1 JRO03L eng.
student.20130804.210540 test-keys
MENU

พื้นฐานการใช้ Android Debug Bridge (ADB)

การเรียนรู้หลักการทำงานของ Android Emulator ถือว่าเป็นพื้นฐานการเริ่มต้นที่ดีสำหรับนักพัฒนาทางด้านสมองกลฝังตัวนี้ เนื่องจากมีองค์ประกอบทั้งด้านกราฟฟิก ด้านระบบปฏิบัติการพื้นฐาน และ Android Kernel รวมถึงการพัฒนาโปรแกรมประยุกต์ ดังนั้นเพื่อให้สามารถเข้าใจโครงสร้าง root filesystem ได้โดยละเอียด รวมทั้งสามารถติดตั้งหรือถอนโปรแกรมในระบบปฏิบัติการแอนดรอยด์ได้ จะต้องใช้โปรแกรมเครื่องมือชื่อว่า “adb” (Android Debug Bridge) ในการติดต่อและเข้าสู่ระบบภายใน Android Emulator ดังแสดงรายละเอียดการทำงานของ adb ดังรูปข้างล่าง



รูปที่ 5.11 แสดงกลไกการทำงานของ adb

ตัวอย่างคำสั่งแสดงรายการอุปกรณ์แอนดรอยด์ที่ต่ออยู่กับเครื่อง Host

```
$ adb devices
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
List of devices attached
emulator-5554     device
```

การติดต่อเข้าสู่ระบบภายในระบบปฏิบัติการแอนดรอยด์ของ Android Emulator

```
$ adb shell
# ls -al
drwxr-xr-x root     root              2013-08-17 15:11 acct
drwxrwx--- system   cache             2013-08-17 15:11 cache
dr-x------ root     root              2013-08-17 15:11 config
lrwxrwxrwx root     root              2013-08-17 15:11 d -> /sys/kernel/debug
```

```
drwxrwx--x system   system            2013-08-05 02:01 data
-rw-r--r-- root     root           116 1970-01-01 00:00 default.prop
drwxr-xr-x root     root               2013-08-17 15:11 dev
lrwxrwxrwx root     root               2013-08-17 15:11 etc -> /system/etc
-rwxr-x--- root     root        234929 1970-01-01 00:00 init
-rwxr-x--- root     root          2344 1970-01-01 00:00 init.goldfish.rc
-rwxr-x--- root     root         17057 1970-01-01 00:00 init.rc
-rwxr-x--- root     root          1637 1970-01-01 00:00 init.trace.rc
-rwxr-x--- root     root          3915 1970-01-01 00:00 init.usb.rc
drwxrwxr-x root     system             2013-08-17 15:11 mnt
dr-xr-xr-x root     root               2013-08-17 15:11 proc
drwx------ root     root               2011-11-14 19:00 root
drwxr-x--- root     root               1970-01-01 00:00 sbin
lrwxrwxrwx root     root               2013-08-17 15:11 sdcard -> /mnt/sdcard
drwxr-xr-x root     root               2013-08-17 15:11 sys
drwxr-xr-x root     root               2013-08-04 18:33 system
-rw-r--r-- root     root           272 1970-01-01 00:00 ueventd.goldfish.rc
-rw-r--r-- root     root          3879 1970-01-01 00:00 ueventd.rc
lrwxrwxrwx root     root               2013-08-17 15:11 vendor -> /system/vendor
```

```
# cat /proc/cpuinfo
processor     : 0
vendor_id     : GenuineIntel
cpu family     : 6
model          : 3
model name     : Pentium II (Klamath)
stepping     : 3
cpu MHz     : 2293.936
cache size     : 2048 KB
fdiv_bug     : no
hlt_bug     : no
f00f_bug     : no
coma_bug     : no
fpu          : yes
fpu_exception     : yes
cpuid level     : 2
wp          : yes
flags          : fpu de pse tsc msr pae mce cx8 apic sep pge cmov pat mmx fxsr sse
sse2 pni ssse3
bogomips     : 4587.87
clflush size: 32
```

แสดงพื้นที่ภายใน MTD Device ของระบบไฟล์ภายในระบบปฏิบัติการแอนดรอยด์แต่ละพาร์ทิชั่นด้วยคำสั่งดังนี้

```
# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 0f180000 00020000 "system"
mtd1: 0c200000 00020000 "userdata"
mtd2: 04000000 00020000 "cache"
```

```
# cat /proc/mounts
rootfs            / rootfs                  ro 0 0
tmpfs             /dev          tmpfs       rw,nosuid,mode=755 0 0
devpts            /dev/pts      devpts      rw,mode=600 0 0
proc              /proc          proc       rw 0 0
sysfs             /sys           sysfs      rw 0 0
tmpfs             /mnt/asec      tmpfs      rw,mode=755,gid=1000 0 0
```

```
tmpfs              /mnt/obb       tmpfs      rw,mode=755,gid=1000 0 0
/dev/block/mtdblock0      /system      yaffs2      ro 0 0
/dev/block/mtdblock1      /data        yaffs2      rw,nosuid,nodev 0 0
/dev/block/mtdblock2      /cache      yaffs2      rw,nosuid,nodev 0 0
```

Android Emulator

Android Emulator หรือเรียกอีกชื่อหนึ่งว่า Goldfish ถูกพัฒนาบนพื้นฐานจากโปรแกรม QEMU รวมทั้งซอร์สของ Android Kernel ซึ่งเหมาะในการทดสอบการปรับแต่งหรือเพิ่มเติมโปรแกรมระบบและโปรแกรมประยุกต์อื่นๆเข้าไปภายในระบบปฏิบัติการแอนดรอยด์ ซึ่งการปรับแต่งตัวจำลองเสมือนจริง Goldfish ตัวนี้มี แนวทางการเรียนรู้ออกมาได้เป็น 4 ขั้นตอนดังนี้

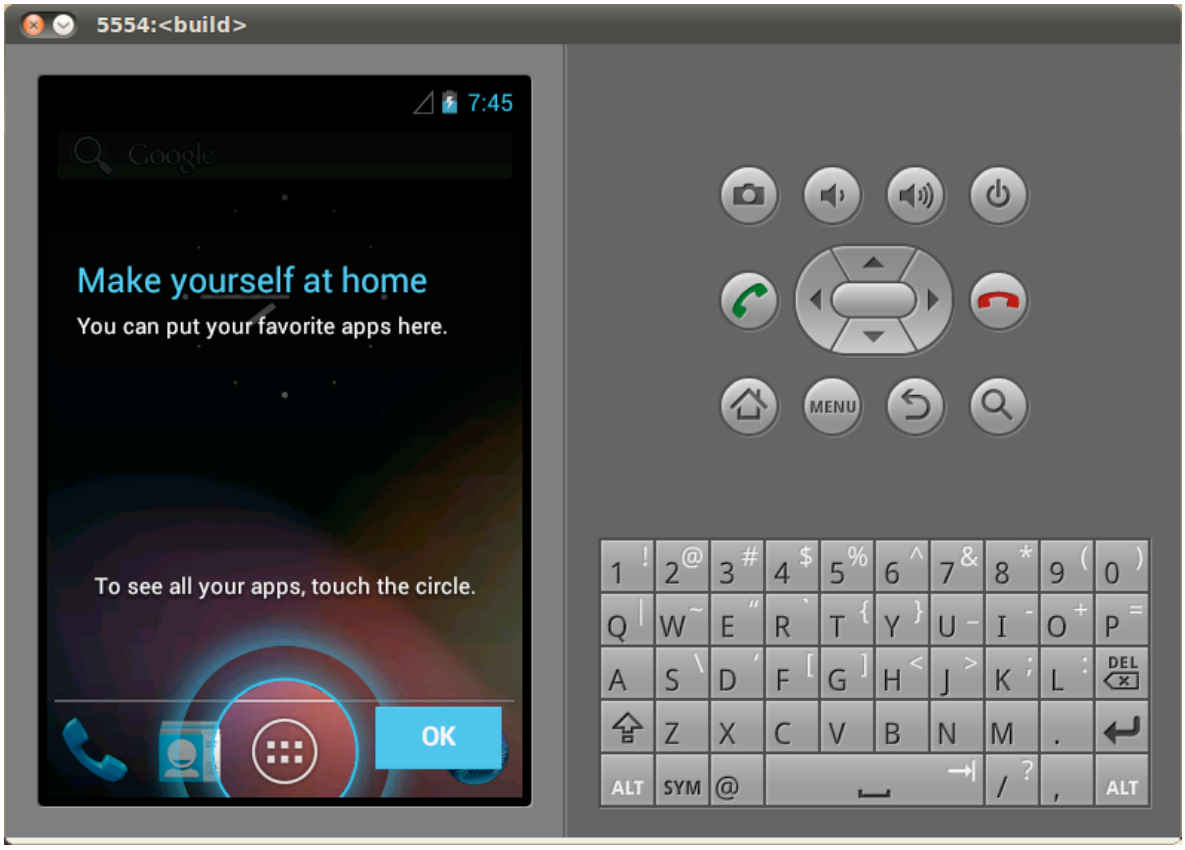
1. การตั้งค่า cross-compiling toolchain สำหรับสร้าง Android kernel
2. ดาวน์โหลดซอร์สโค้ดของ Android Kernel โดยระบุรุ่นของ Kernel ที่ต้องการ
3. การคอมไพล์ Android Kernel
4. การตั้งค่าเมนูให้รองรับ loadable kernel module (LKM)

```
$ cd ~/aosp
$ source build/envsetup.sh
$ lunch full_x86-eng
$ emulator -help
Android Emulator usage: emulator [options] [-qemu args]
options:
-sysdir <dir>                  search for system disk images in <dir>
-system <file>                 read initial system image from <file>
-datadir <dir>                 write user data into <dir>
....
-help-build-images            about disk images when building Android
-help-all                     prints all help content
```

ตัวอย่างการระบุตัวเลือก -show-kernel เพื่อแสดงผลการทำงานของ Android Kernel สามารถทำได้ดังนี้

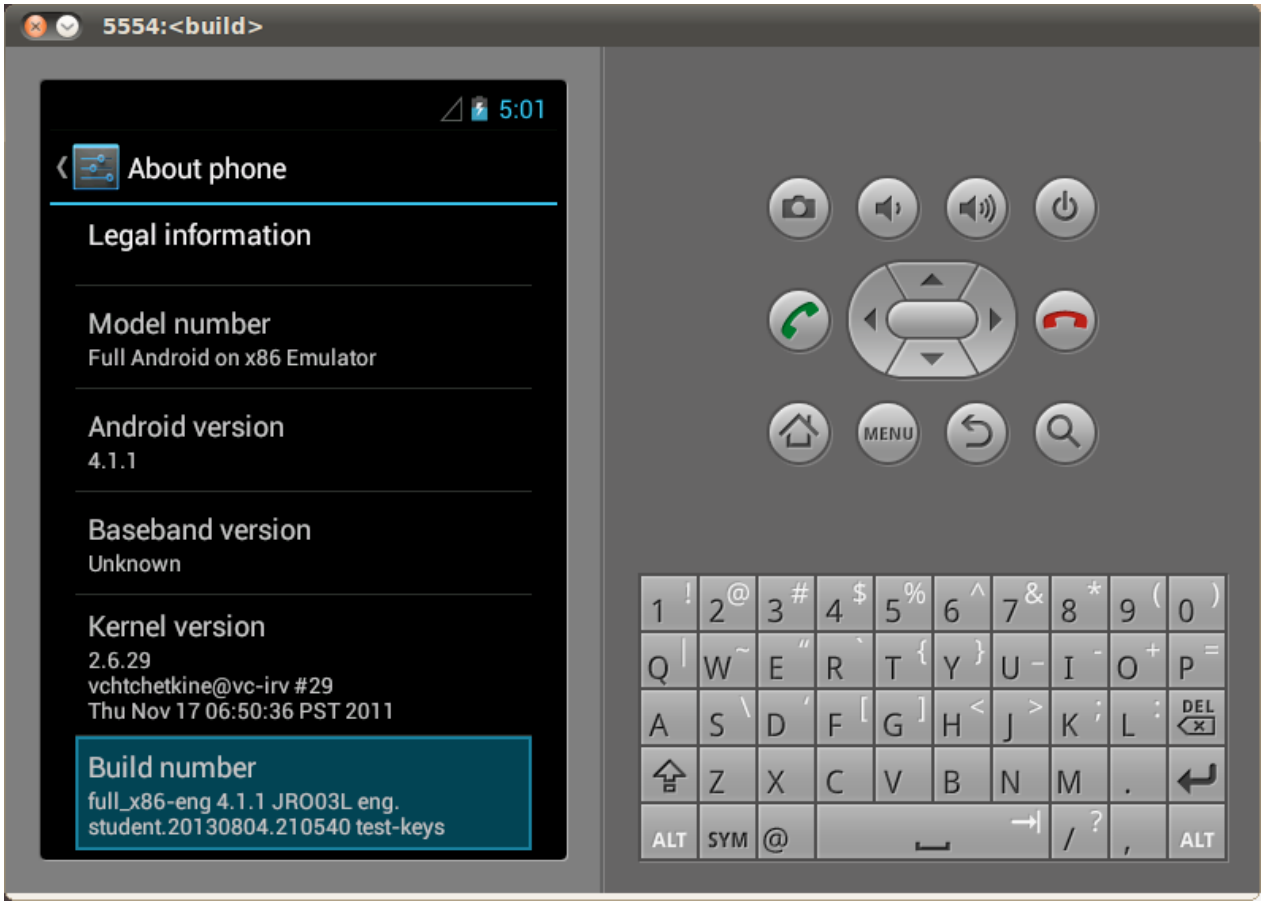
```
$ emulator -kernel ~/aosp/goldfish/arch/x86/boot/bzImage -wipe-data
-show-kernel
emulator: WARNING: system partition size adjusted to match image file (249 MB > 200
MB)
serial0 console
student@marabuntu:~/aosp$ [    0.000000] Linux version 3.4.0+ (student@marabuntu)
(gcc version 4.4.3 (GCC) ) #1 PREEMPT Tue Aug 6 12:14:53 PDT 2013
[    0.000000] BIOS-provided physical RAM map:
[    0.000000]  BIOS-e820: 0000000000000000 - 000000000009f000 (usable)
[    0.000000]  BIOS-e820: 000000000009f000 - 00000000000a0000 (reserved)
[    0.000000]  BIOS-e820: 00000000000e8000 - 0000000000100000 (reserved)
[    0.000000]  BIOS-e820: 0000000000100000 - 000000001fff0000 (usable)
[    0.000000]  BIOS-e820: 000000001fff0000 - 0000000020000000 (ACPI data)
[    0.000000]  BIOS-e820: 00000000fffc0000 - 0000000100000000 (reserved)
```

```
[    0.000000] Notice: NX (Execute Disable) protection missing in CPU!
[    0.000000] DMI 2.4 present.
[    0.000000] last_pfn = 0x1fff0 max_arch_pfn = 0x100000
[    0.000000] init_memory_mapping: 0000000000000000-000000001fff0000
[    0.000000] RAMDISK: 1ffb4000 - 1fff0000
.....
for 1f700000-1f796000, got write-back
[    4.816692] Clocksource tsc unstable (delta = 70001460 ns)
[   35.829113] warning: `zygote' uses 32-bit capabilities (legacy support in use)
[   86.091553] binder: 1208: binder_alloc_buf, no vma
[   86.091710] binder: 1115:1126 transaction failed 29201, size 4-0
[   86.091868] binder: send failed reply for transaction 3147 to 1208:1208
[   99.262724] init: sys_prop: permission denied uid:1003
name:service.bootanim.exit
```



ขั้นตอนการอัพเกรดและปรับแต่ง Android Kernel สำหรับ Emulator

จากรูปด้านล่างจะเห็นได้ว่า Android Emulator ของ Android รุ่น 4.1.1 (ICS) พื้นฐานที่มากับ AOSP นั้นยังใช้ Android Kernel รุ่น 2.6.29 เดิมอยู่ดังแสดงในรูปข้างล่าง



ขั้นตอนการอัพเกรด Android Kernel สำหรับ Android Emulator (Goldfish)

เริ่มต้นด้วยการดาวน์โหลด Android Kernel รุ่นใหม่ดังคำสั่งข้างล่าง

```
$ cd ~/aosp
$ git clone http://android.googlesource.com/kernel/goldfish.git
```

ใช้คำสั่ง git branch -r เพื่อตรวจสอบว่ามี Kernel รุ่นไหนอยู่บ้าง

```
$ cd goldfish
$ git branch -r
origin/HEAD -> origin/master
origin/android-goldfish-2.6.29
```

```
origin/android-goldfish-3.4
origin/linux-goldfish-3.0-wip
origin/master
```

เลือกใช้ Kernel รุ่นที่ต้องการ ซึ่งในที่นี่จะเลือกรุ่น 3.4

```
$ git checkout -b 3.4 origin/android-goldfish-3.4
Checking out files: 100% (38868/38868), done.
Branch 3.4 set up to track remote branch android-goldfish-3.4 from origin.
Switched to a new branch '3.4'
```

ตั้งค่า Kernel พื้นฐานสำหรับ Goldfish

โดยเริ่มต้นจะอ้างอิงจากค่าที่เลือกพื้นฐานไว้สำหรับ Goldfish Emulator ซึ่งถูกเก็บอยู่ภายใต้ไดเรกทอรี `arch/x86/configs/`

```
$ ls -al arch/x86/configs/
total 80
drwxr-xr-x  2 student student  4096 2013-08-17 18:46 .
drwxr-xr-x 25 student student  4096 2013-08-17 18:46 ..
-rw-r--r--  1 student student 55883 2013-08-17 18:46 goldfish_defconfig
-rw-r--r--  1 student student  7511 2013-08-17 18:46 i386_defconfig
-rw-r--r--  1 student student  7528 2013-08-17 18:46 x86_64_defconfig

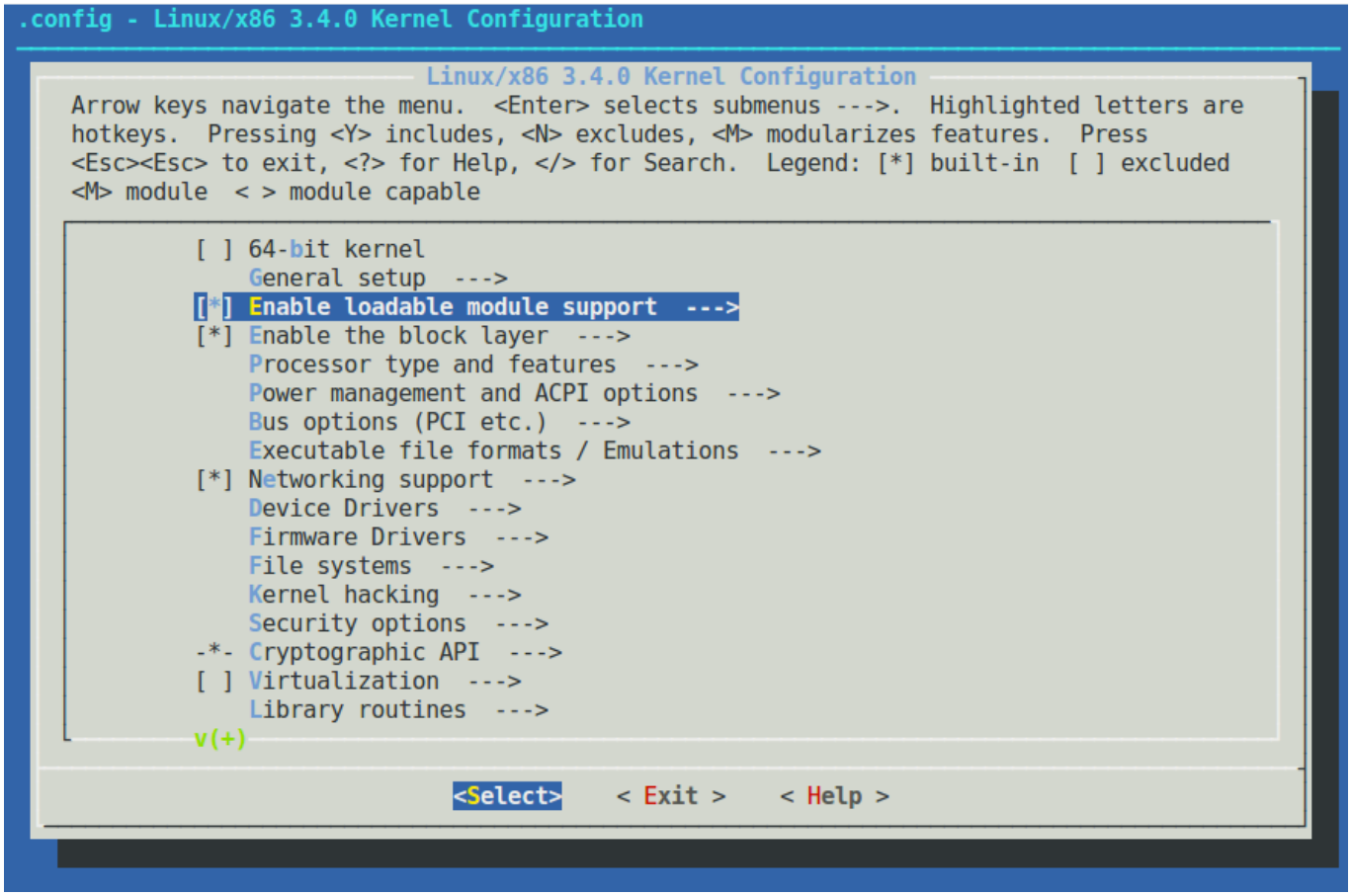
$ make goldfish_defconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  SHIPPED scripts/kconfig/zconf.tab.c
  SHIPPED scripts/kconfig/zconf.lex.c
  SHIPPED scripts/kconfig/zconf.hash.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
```

ในกรณีที่ต้องการใช้การตั้งค่าที่มากับบอร์ด target และอุปกรณ์มือถือ/แท็ปเล็ตนั้น โดยปกติจะมีไฟล์ .config เดิมอยู่แล้วซึ่งจะถูกบีบอัดและเก็บในชื่อไฟล์ว่า config.gz และจะถูกเก็บอยู่ในไดเรกทอรี /proc/ ของเครื่องนั้นๆ ดังนั้นนักพัฒนาสามารถใช้คำสั่ง adb pull เพื่อดึงไฟล์ config.gz ออกมา แล้วเปลี่ยนชื่อไฟล์เป็น .config ดังคำสั่งข้างล่าง แต่อย่างไรก็ตามนักพัฒนาก็ยังสามารถดาวน์โหลดเพิ่มเติมและตั้งค่าเคอร์เนลพิเศษเพิ่มเติมได้

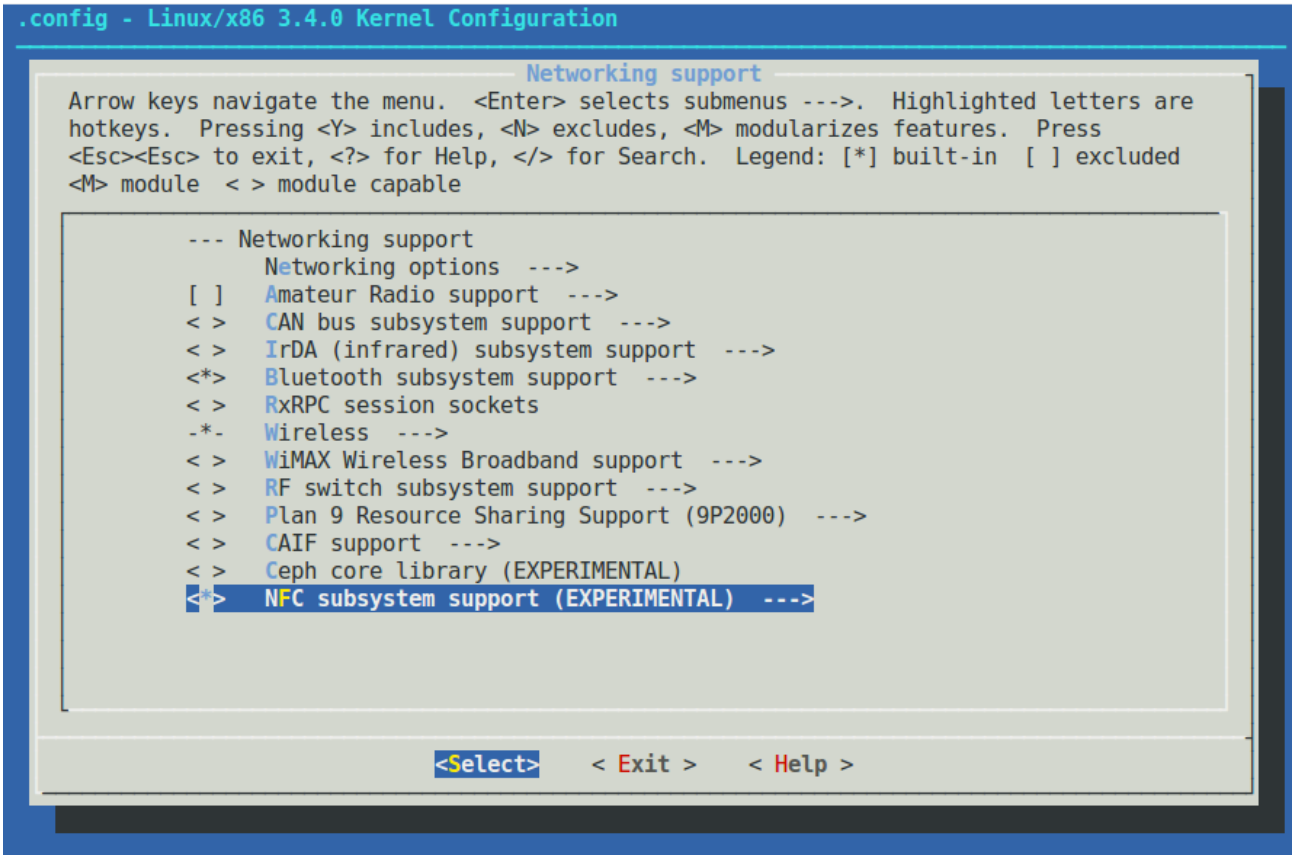
```
$ adb pull /proc/config.gz .
$ gunzip config.gz
$ cp config .config
```

ขั้นตอนสุดท้ายก่อนที่จะคอมไพล์เพื่อสร้าง Kernel image ใหม่ นักพัฒนาสามารถปรับแต่งค่าต่างๆ ที่มีอยู่ใน เคอร์เนลได้ตามต้องการ ตัวอย่างเช่น Networking support, Device Drivers, Firmware Drivers, File systems, Security options, Cryptographic API, Virtualization, Library routines เป็นต้น

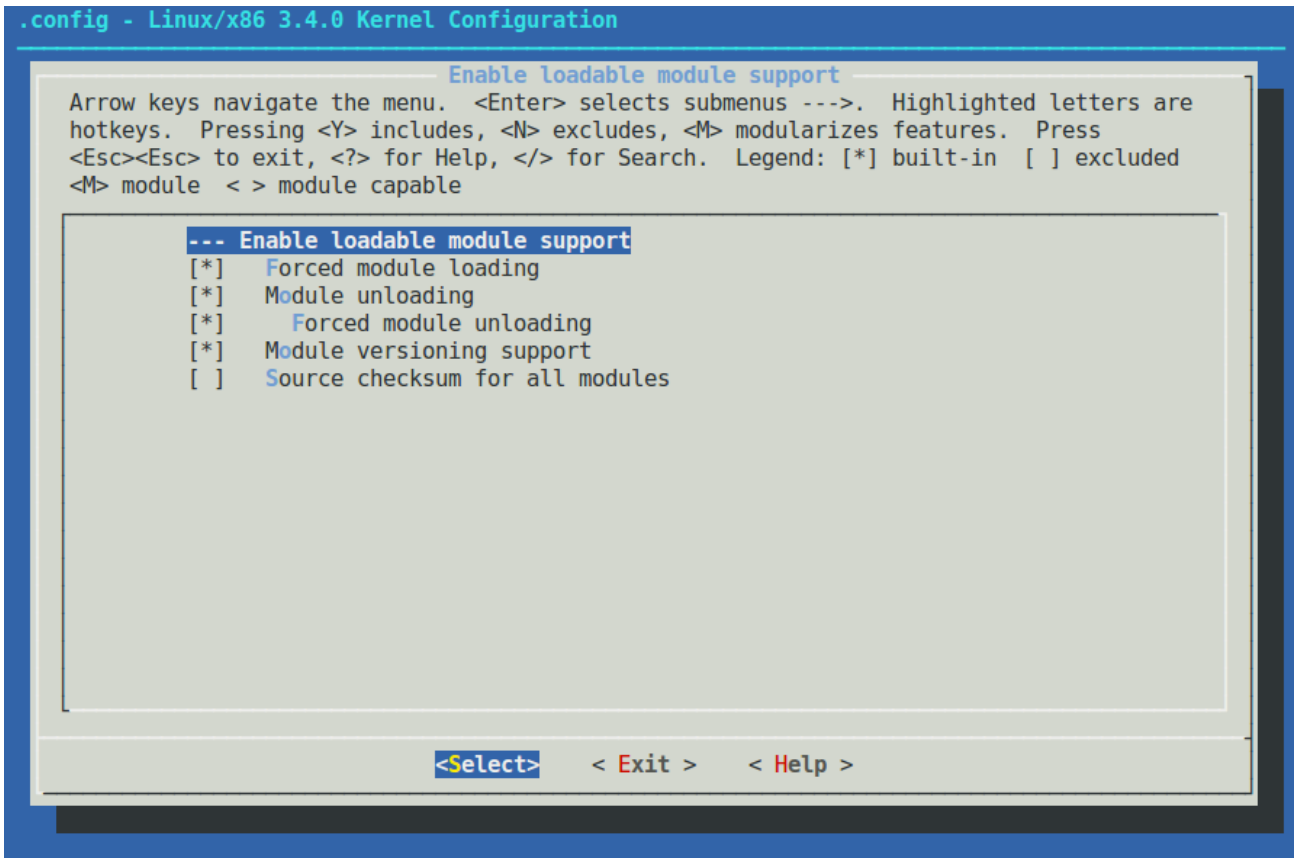
```
$ make menuconfig
```



ทดลองเพิ่มการรองรับการใช้งาน Bluetooth และ NFC เพิ่มเติม โดยการเข้าไปเลือกจากเมนู Networking support



เลือกให้รองรับ Kernel Loading Module (KLM) โดยการเลือกในเมนู enable loadable module support ดังแสดงในรูปข้างล่าง



คอมไพล์เคอร์เนลรุ่นใหม่สำหรับ Goldfish

เริ่มต้นโดยการตั้งค่าสภาพแวดล้อมสำหรับ cross-compiling และสถาปัตยกรรมของ Goldfish Emulator ซึ่งในที่นี้คือสถาปัตยกรรม x86

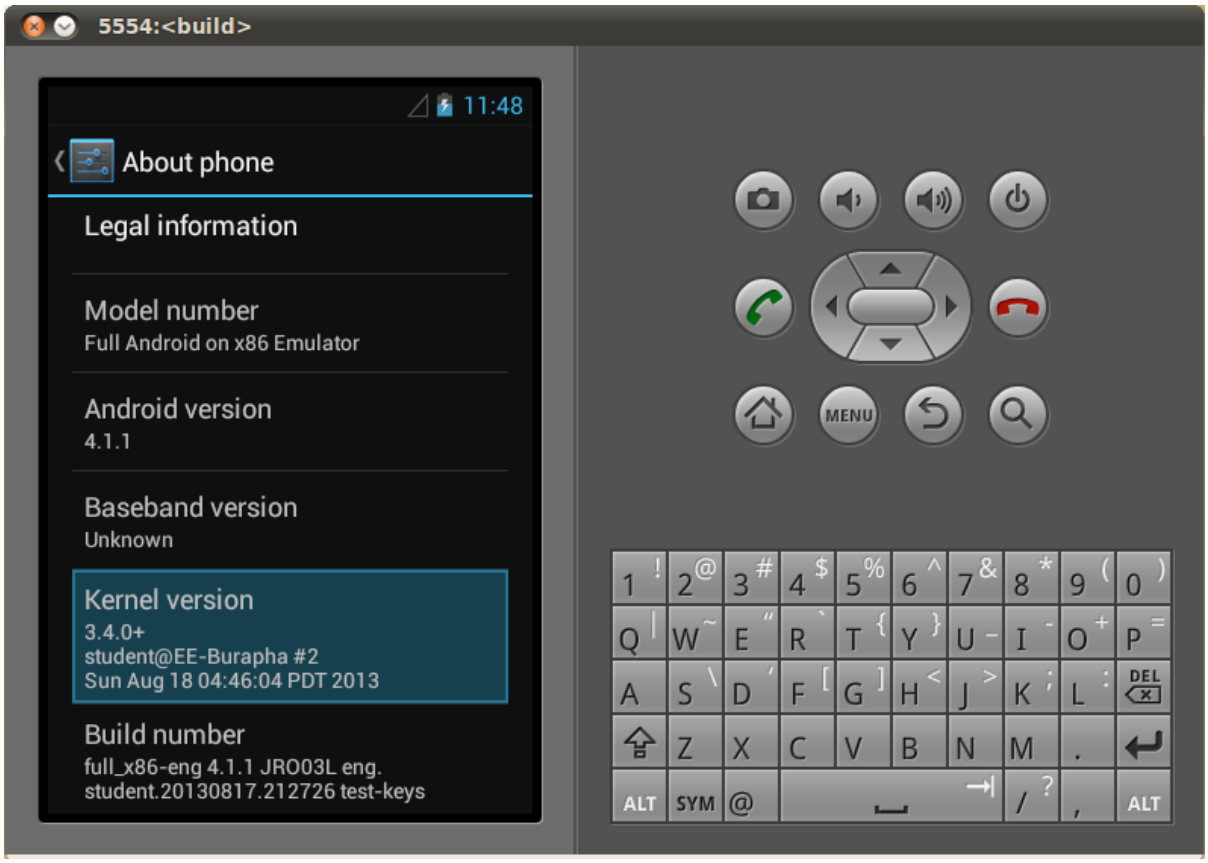
```
$ export
CROSS_COMPILE=~/aosp/external/qemu/distrib/kernel-toolchain/android-kernel-toolchain-
$ export
REAL_CROSS_COMPILE=~/aosp/prebuilts/gcc/linux-x86/x86/i686-android-linux-4.4.3/bin/i686-android-linux-
$ export ARCH=x86
$ export SUBARCH=x86
$ make -j3
```

หลังจากคอมไพล์สิ้นสุดลงก็จะได้ไฟล์ kernel image ชื่อว่า “bzImage” เกิดขึ้นอยู่ในไดเรกทอรี arch/x86/boot/ ดังรายการคำสั่งข้างล่าง

```
$ ls arch/x86/boot/
a20.c          cpustr.h               mtools.conf.in  version.o
a20.o          ctype.h                pm.c            vesa.h
apm.c          early_serial_console.c pmjump.o        video-bios.c
bioscall.o     early_serial_console.o pmjump.S        video-bios.o
bioscall.S     edd.c                  pm.o            video.c
bitops.h       edd.o                  printf.c        video.h
boot.h         header.o               printf.o        video-mode.c
bzImage        header.S               regs.c          video-mode.o
cmdline.c      install.sh             regs.o          video.o
cmdline.o      main.c                 setup.bin       video-vesa.c
code16gcc.h    main.o                 setup.elf       video-vesa.o
compressed     Makefile               setup.ld        video-vga.c
copy.o         mca.c                  string.c        video-vga.o
copy.S         mca.o                  string.o        vmlinux.bin
cpu.c          memory.c               tools           voffset.h
cpucheck.c     memory.o               tty.c           zoffset.h
cpucheck.o     mkcpustr               tty.o
cpu.o          mkcpustr.c             version.c
```

ทำการเรียกโปรแกรม Android Emulator ขึ้นมาใหม่อีกครั้งโดยระบุตำแหน่งที่เก็บไฟล์ kernel image ตัวใหม่ โดยใช้ option -kernel ดังคำสั่งข้างล่างนี้

```
$ cd ~/aosp
$ source build/envsetup.sh
$ lunch full_x86-eng
$ emulator -kernel ~/aosp/goldfish/arch/x86/boot/bzImage -wipe-data &
```



การพัฒนา Kernel Module สำหรับระบบปฏิบัติการแอนดรอยด์

ในตอนนี้ตัว Android Emulator สามารถใช้งานกับเคอร์เนลตัวใหม่ (รุ่น 3.4) ซึ่งขั้นตอนหลังจากนี้นักพัฒนาจะสามารถศึกษาและทดสอบการปรับแต่งค่าต่างๆภายใน Android Kernel ได้แล้ว นอกจากนั้นก็ยังสามารถพัฒนาโปรแกรม kernel module หรือ kernel device driver ในระดับ kernel space เพื่อปรับปรุงระบบปฏิบัติการแอนดรอยด์หรือรองรับการติดต่อกับอุปกรณ์ภายในได้ยืดหยุ่นมากขึ้น

ตัวอย่างการพัฒนา kernel module อย่างง่าย ดังแสดงในขั้นตอนข้างล่างนี้

```
$ mkdir ~/aosp/goldfish/my_modules
$ cd ~/aosp/goldfish/my_modules/
```

การเขียนโมดูลสามารถเขียนได้หลายแบบ ดังตัวอย่างโปรแกรม hello.c และ hello1.c ข้างล่าง

```
// hello.c
#include <linux/module.h>  /* Needed by all modules */
#include <linux/kernel.h>  /* Needed for KERN_ALERT */
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Wiroon Sriborrirux EE-Burapha, 2013");
MODULE_DESCRIPTION("Demo module");

int init_module(void)
```

```
{
    printk(KERN_INFO "Hello world! 1\n");
    return 0;
}

void cleanup_module(void)
{
    printk(KERN_ALERT "Cleaning up module 1\n");
}
```

ตัวอย่างโปรแกรม hello1.c

```
// hello1.c
// Defining __KERNEL__ and MODULE allows us to access kernel-level code not usually
available to userspace programs.
#undef __KERNEL__
#define __KERNEL__

#undef MODULE
#define MODULE

// Linux Kernel/LKM headers: module.h is needed by all modules and kernel.h is
needed for KERN_INFO.
#include <linux/module.h>    // included for all kernel modules
#include <linux/kernel.h>    // included for KERN_INFO
#include <linux/init.h>      // included for __init and __exit macros

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Wiroon Sriborrirux EE-Burapha, 2013");
MODULE_DESCRIPTION("Demo module");

static int __init hello_init(void)
{
    printk(KERN_INFO "Hello world! 2\n");
    return 0;    // Non-zero return means that the module couldn't be loaded.
}

static void __exit hello_cleanup(void)
{
    printk(KERN_INFO "Cleaning up module 2\n");
}
module_init(hello_init);
module_exit(hello_cleanup);
```

ทำการสร้างไฟล์ Makefile เพื่อใช้ในการคอมไพล์โปรแกรมที่สร้างขึ้น ดังนี้

```
$ vim Makefile

VERSION = 3
PATCHLEVEL = 4
```

```
SUBLEVEL = 0
EXTRAVERSION=

obj-m += hello.o hello1.o

all:
     make -C /home/student/aosp/goldfish/ M=$(PWD) modules
clean:
     make -C /home/student/aosp/goldfish/ M=$(PWD) clean
```

ก่อนจะทำการคอมไพล์โมดูล hello จำเป็นจะต้องคอมไพล์ Kernel เพื่อให้พร้อมสำหรับการใช้งานโมดูล ด้วยคำสั่ง

```
$ cd ~/aosp/goldfish/
$ make ARCH=x86
CROSS_COMPILE=~/aosp/prebuilts/gcc/linux-x86/x86/i686-android-linux-4.
4.3/bin/i686-android-linux- modules_prepare

make[1]: Nothing to be done for `all'.
make[1]: Nothing to be done for `relocs'.
  CHK     include/linux/version.h
  CHK     include/generated/utsrelease.h
  CC      kernel/bounds.s
  GEN     include/generated/bounds.h
  CC      arch/x86/kernel/asm-offsets.s
  GEN     include/generated/asm-offsets.h
  CALL    scripts/checksyscalls.sh
  CC      scripts/mod/empty.o
  MKELF   scripts/mod/elfconfig.h
  HOSTCC  scripts/mod/file2alias.o
  HOSTCC  scripts/mod/modpost.o
  HOSTCC  scripts/mod/sumversion.o
  HOSTLD  scripts/mod/modpost
```

เริ่มการคอมไพล์ kernel module ที่ได้สร้างไว้ (hello.c)

```
$ cd ~/aosp/goldfish/my_modules/
$ make
make -C /home/student/aosp/goldfish/ M=/home/student/aosp/goldfish-
/custom_modules modules
make[1]: Entering directory `/home/student/aosp/goldfish'
  CC [M]  /home/student/aosp/goldfish/custom_modules/hello.o
  CC [M]  /home/student/aosp/goldfish/custom_modules/hello1.o
  Building modules, stage 2.
  MODPOST 2 modules
  CC      /home/student/aosp/goldfish/custom_modules/hello.mod.o
  LD [M]  /home/student/aosp/goldfish/custom_modules/hello.ko
  CC      /home/student/aosp/goldfish/custom_modules/hello1.mod.o
  LD [M]  /home/student/aosp/goldfish/custom_modules/hello1.ko
make[1]: Leaving directory `/home/student/aosp/goldfish'

$ ls
```

```
hello1.c      hello1.mod.o  hello.ko     hello.o    Makefile_old
hello1.ko     hello1.o      hello.mod.c  Makefile   modules.order
hello1.mod.c  hello.c       hello.mod.o  Makefile1  Module.symvers
```

หลังจากการคอมไพล์เสร็จสมบูรณ์ก็จะได้ไฟล์โมดูลชื่อว่า hello.ko ให้ทำการคัดลอกโดยส่งผ่าน adb push ไปยัง Android Emulator ด้วยคำสั่งดังนี้

```
$ adb push hello.ko /data/local
52 KB/s (2203 bytes in 0.040s)
$ adb push hello1.ko /data/local
54 KB/s (2238 bytes in 0.040s)
$ adb shell
# cd /data/local
# ls
hello.ko
hello1.ko
tmp
```

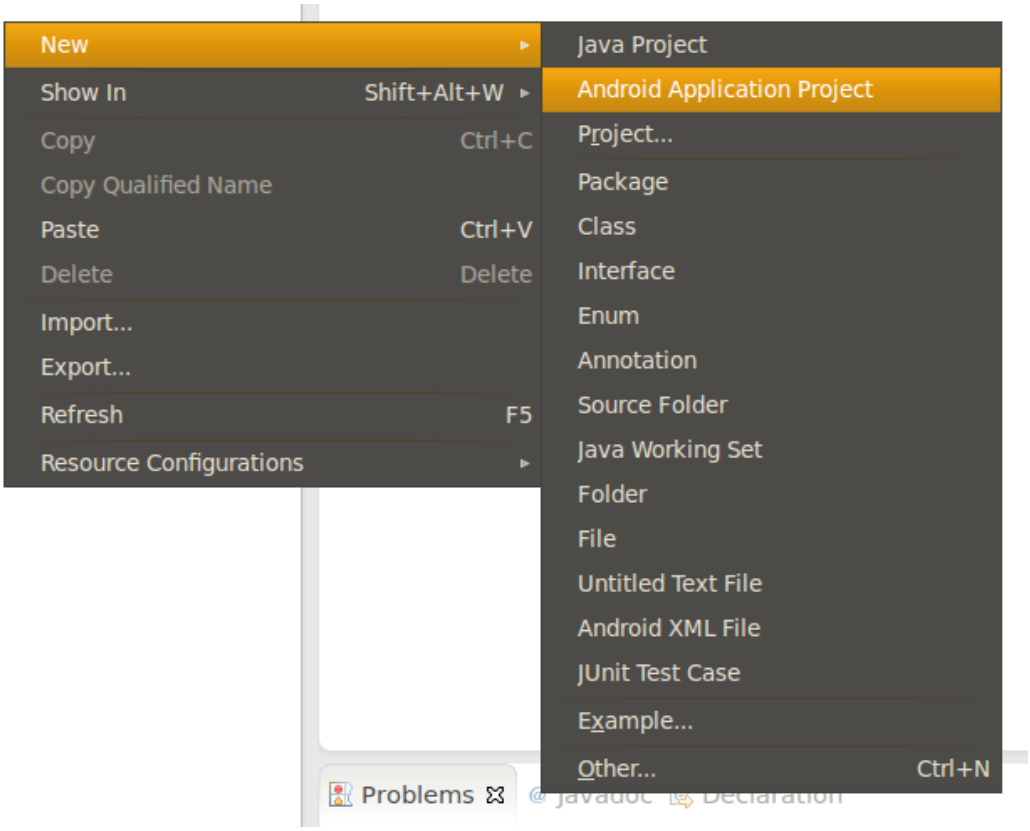
ทดสอบโหลดโมดูล hello และ hello1 ภายใต้ระบบปฏิบัติการแอนดรอยด์ ซึ่งใช้ Kernel รุ่นใหม่ (3.4.0) และตรวจสอบผลการทำงานของโมดูล ด้วยคำสั่ง dmesg

```
# insmod hello.ko
# insmod hello1.ko
# lsmod
hello1 506 0 - Live 0x00000000 (PO)
hello 540 0 - Live 0x00000000 (O)
# rmmod hello
# rmmod hello1
# dmesg
...
<3>[ 79.016988] init: sys_prop: permission denied uid:1003
name:service.bootanim.exit
<4>[ 4000.418990] hello: module license 'unspecified' taints kernel.
<4>[ 4000.419343] Disabling lock debugging due to kernel taint
<6>[ 4064.182198] Hello world! 1
<6>[ 4068.910892] Hello world! 2
<6>[ 4086.416483] Cleaning up module 1
<6>[ 4089.112969] Cleaning up module 2
```

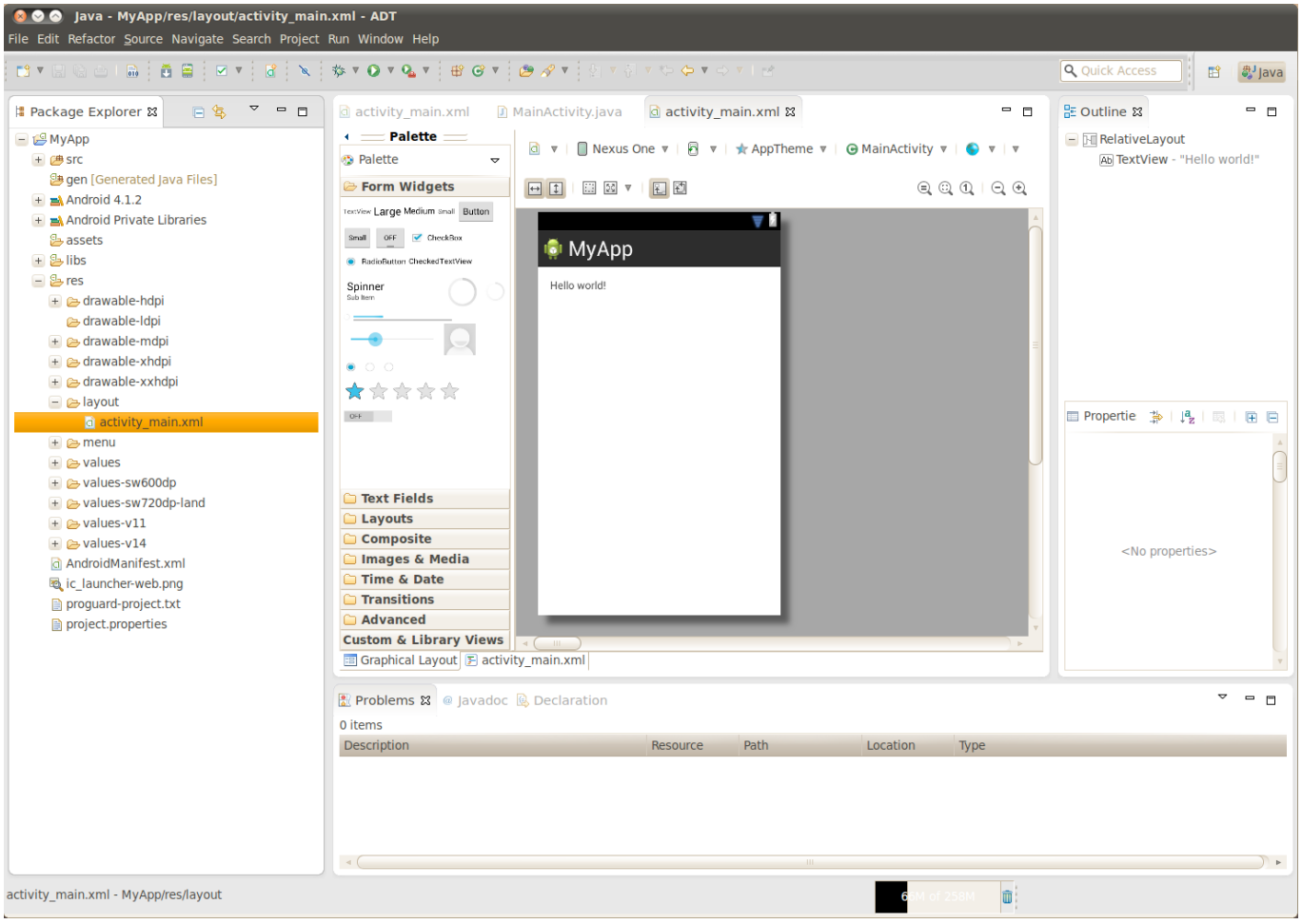
การสร้างโปรแกรมประยุกต์เพื่อฝังลงในแอนดรอยด์

เมื่อต้องการติดตั้งโปรแกรมประยุกต์สำหรับระบบปฏิบัติการแอนดรอยด์ (.APK) โดยทั่วไปแล้วจะมีวิธีการด้วยกัน 2 วิธีคือการนำขึ้น Google PlayStore แล้วดาวน์โหลดเพื่อติดตั้ง และวิธีที่สองคือการนำไฟล์ติดตั้ง (นามสกุล .APK) มาติดตั้งลงในเครื่องโดยตรง แต่อย่างไรก็ตามถ้าต้องการติดตั้งโปรแกรมประยุกต์ที่ได้พัฒนาขึ้นมาฝังลงไปอยู่ในระบบปฏิบัติการแอนดรอยด์เพื่อให้เป็นหนึ่งในชุดโปรแกรมพื้นฐานของอุปกรณ์ยี่ห้อนั้นตั้งแต่แรกนักพัฒนาจำเป็นจะต้องจัดเตรียมตั้งแต่การสร้างระบบปฏิบัติการแอนดรอยด์ตั้งแต่เริ่มต้นโดยจะมีขั้นตอนการทำดังต่อไปนี้

1. เปิดโปรแกรม Eclipse เพื่อสร้างแอพพลิเคชั่นอย่างง่าย โดยตั้งชื่อว่า MyApp




New Android Application
New Android Application
The prefix 'com.example.' is meant as a placeholder and should not be used
Application Name: MyApp
Project Name: MyApp
Package Name: com.example.myapp
Minimum Required SDK: API 8: Android 2.2 (Froyo)
Target SDK: API 16: Android 4.1 (Jelly Bean)
Compile With: API 16: Android 4.1 (Jelly Bean)
Theme: Holo Light with Dark Action Bar
Choose a target API to compile your code against, from your installed SDKs. This is typically the most recent version, or the first version that supports all the APIs you want to directly access without reflection.
< Back
Next >
Cancel
Finish


Java - MyApp/res/layout/activity_main.xml - ADT
File Edit Refactor Source Navigate Search Project Run Window Help
Package Explorer
MyApp
src
gen [Generated Java Files]
Android 4.1.2
Android Private Libraries
assets
libs
res
drawable-hdpi
drawable-ldpi
drawable-mdpi
drawable-xhdpi
drawable-xxhdpi
layout
activity_main.xml
menu
values
values-sw600dp
values-sw720dp-land
values-v11
values-v14
AndroidManifest.xml
ic_launcher-web.png
proguard-project.txt
project.properties
Palette
Form Widgets
Text Fields
Layouts
Composite
Images & Media
Time & Date
Transitions
Advanced
Custom & Library Views
MyApp
Hello world!
Outline
RelativeLayout
TextView - "Hello world!"
Problems
Javadoc
Declaration
0 items
Description
Resource
Path
Location
Type
activity_main.xml - MyApp/res/layout

```java
// MainActivity.java
package com.example.myapp;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

public class MainActivity extends Activity {

     @Override
protected void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.activity_main);
     }

     @Override
public boolean onCreateOptionsMenu(Menu menu) {
// Inflate the menu; this adds items to the action bar if it is present.
          getMenuInflater().inflate(R.menu.main, menu);
          return true;
     }
}
```

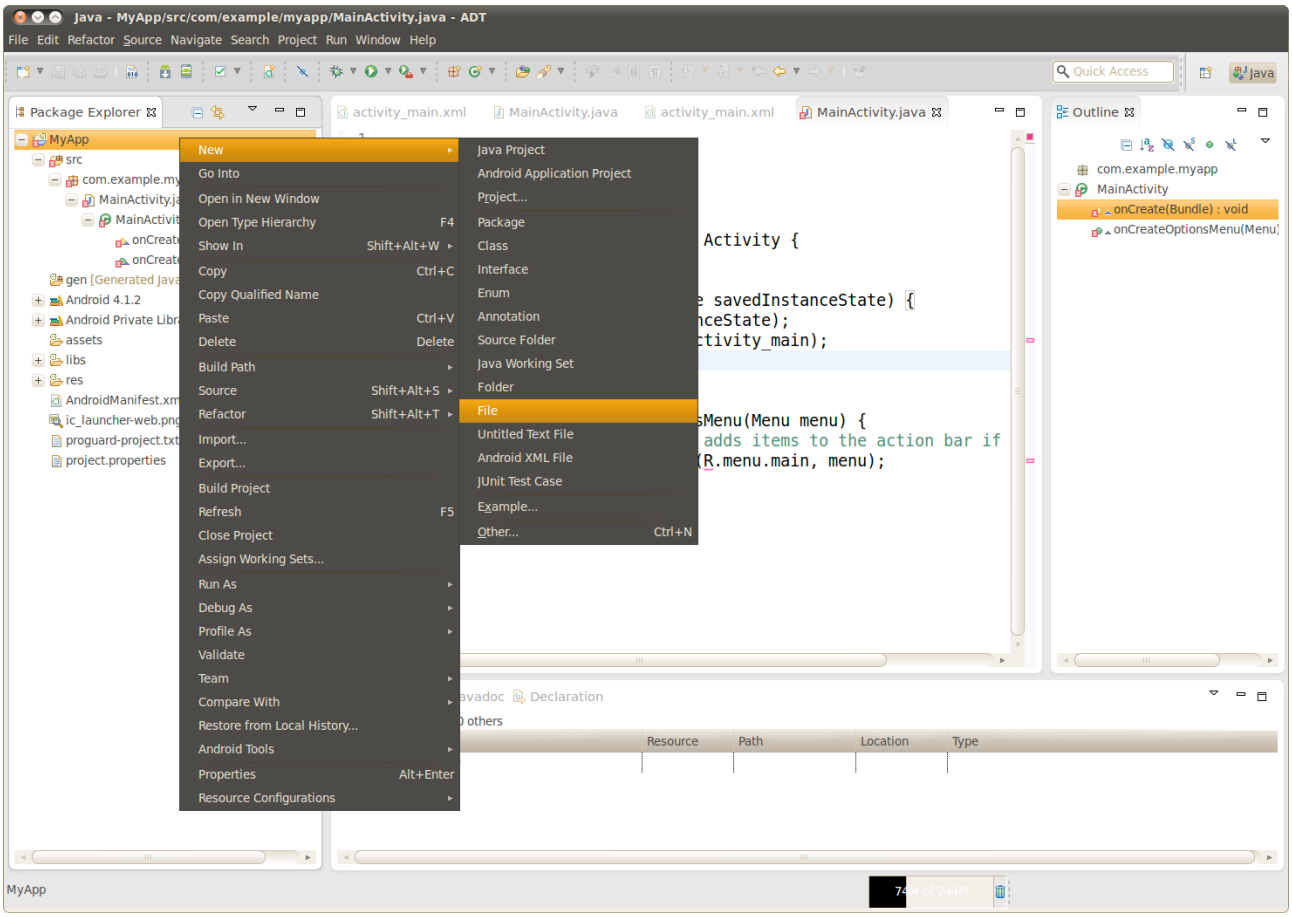
```xml
// activity_main.xml

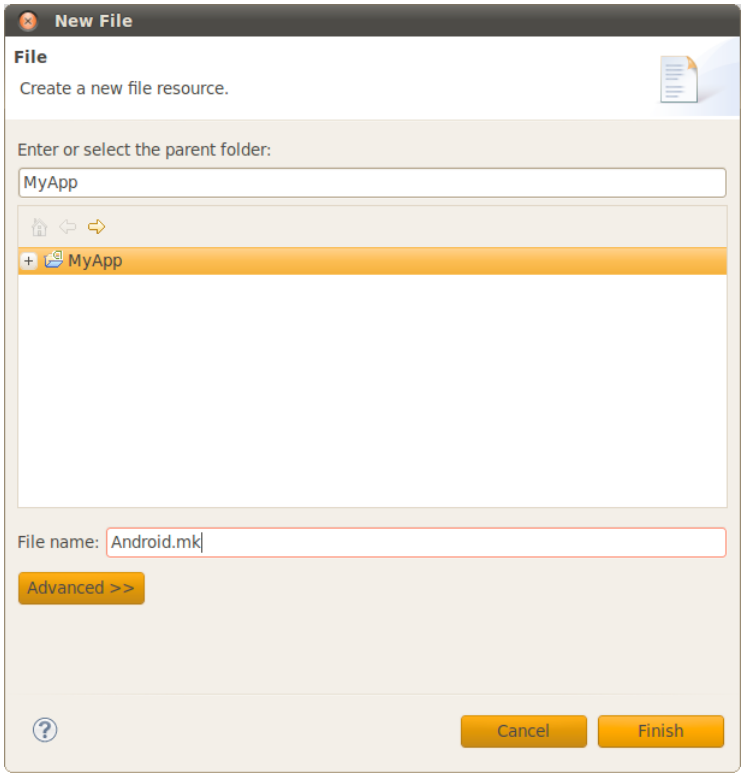
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</RelativeLayout>
```

2. สร้างไฟล์ Android.mk





3. สร้าง symbolic link เพื่อชี้ไปยังไดเรกทอรีโปรแกรม MyApp

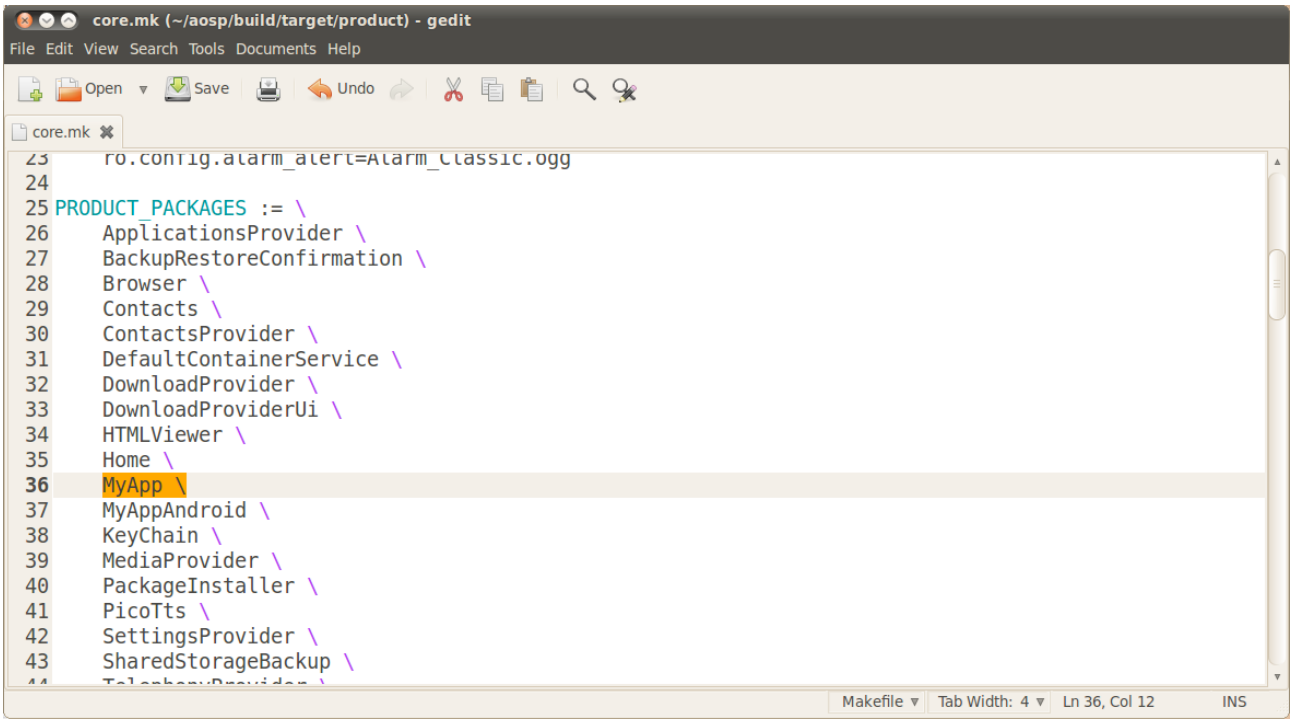
```
$ cd ~/aosp/packages/apps
$ ln -s /home/student/android/workspace/MyApp MyApp
$ ls -al
total 148
drwxr-xr-x 37 student student 4096 2013-08-19 02:27 .
drwxr-xr-x  7 student student 4096 2012-09-04 04:30 ..
drwxr-xr-x  5 student student 4096 2012-09-04 22:50 BasicSmsReceiver
drwxr-xr-x  4 student student 4096 2012-09-04 22:50 Bluetooth
drwxr-xr-x  7 student student 4096 2012-09-04 22:50 Browser
drwxr-xr-x  5 student student 4096 2012-09-04 22:50 Calculator
drwxr-xr-x  5 student student 4096 2012-09-04 22:50 Calendar
drwxr-xr-x  7 student student 4096 2012-09-04 22:50 Camera
drwxr-xr-x  5 student student 4096 2012-09-04 22:50 CellBroadcastReceiver
drwxr-xr-x  4 student student 4096 2012-09-04 22:50 CertInstaller
drwxr-xr-x  5 student student 4096 2012-09-04 22:50 Contacts
drwxr-xr-x  6 student student 4096 2012-09-04 22:50 DeskClock
drwxr-xr-x  7 student student 4096 2012-09-04 22:50 Email
drwxr-xr-x  6 student student 4096 2012-09-04 22:50 Exchange
drwxr-xr-x  5 student student 4096 2012-09-04 22:50 Gallery
drwxr-xr-x  8 student student 4096 2012-09-04 22:50 Gallery2
drwxr-xr-x  4 student student 4096 2012-09-04 22:50 HTMLViewer
drwxr-xr-x  6 student student 4096 2012-09-04 22:50 KeyChain
drwxr-xr-x  5 student student 4096 2012-09-04 22:50 Launcher2
drwxr-xr-x  6 student student 4096 2012-09-04 22:50 LegacyCamera
drwxr-xr-x  6 student student 4096 2012-09-04 22:50 Mms
drwxr-xr-x  5 student student 4096 2012-09-04 22:50 Music
drwxr-xr-x  4 student student 4096 2012-09-04 22:50 MusicFX
lrwxrwxrwx  1 student student   37 2013-08-19 02:27 MyApp -> /home/student/android/
workspace/MyApp
drwxr-xr-x  8 student student 4096 2012-09-04 22:50 Nfc
drwxr-xr-x  4 student student 4096 2012-09-04 22:50 PackageInstaller
drwxr-xr-x  5 student student 4096 2012-09-04 22:50 Phone
drwxr-xr-x  4 student student 4096 2012-09-04 22:50 Protips
drwxr-xr-x  3 student student 4096 2012-09-04 22:50 Provision
drwxr-xr-x  6 student student 4096 2012-09-04 22:50 QuickSearchBox
drwxr-xr-x  5 student student 4096 2012-09-04 22:50 Settings
drwxr-xr-x  4 student student 4096 2012-09-04 22:50 SoundRecorder
drwxr-xr-x  4 student student 4096 2012-09-04 22:50 SpareParts
drwxr-xr-x  4 student student 4096 2012-09-04 22:50 SpeechRecorder
drwxr-xr-x  4 student student 4096 2012-09-04 22:50 Stk
drwxr-xr-x  5 student student 4096 2012-09-04 22:50 Tag
drwxr-xr-x  4 student student 4096 2012-09-04 22:50 VideoEditor
drwxr-xr-x  5 student student 4096 2012-09-04 22:50 VoiceDialer
```

4. ลบไดเรกทอรี gen/ และ bin/ ที่อยู่ภายใต้ไดเรกทอรีของโปรแกรม MyApp

```
$ cd MyApp
$ rm -rf gen bin
```

5. เพิ่มชื่อแอพพลิเคชั่นเข้าไปในไฟล์ ~/aosp/build/target/product/core.mk ในส่วนของ PRODUCT_PACKAGES

```
$ cd /home/student/aosp/build/target/product/
$ gedit core.mk
```



6. ตั้งค่าสภาพแวดล้อมสำหรับ AOSP ใหม่อีกครั้ง

```
$ cd ~/aosp
$ source build/envsetup.sh
$ lunch 2
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=4.1.1
TARGET_PRODUCT=full_x86
TARGET_BUILD_VARIANT=eng
TARGET_BUILD_TYPE=release
TARGET_BUILD_APPS=
TARGET_ARCH=x86
TARGET_ARCH_VARIANT=x86-atom
HOST_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-2.6.32-42-server-x86_64-with-Ubuntu-10.04-lucid
HOST_BUILD_TYPE=release
BUILD_ID=JRO03L
OUT_DIR=out
============================================
```

7. คอมไพล์และสร้างโปรแกรม MyApp

```
$ cd package/apps/MyApp
$ make
$ mm
============================================
PLATFORM_VERSION_CODENAME=REL
```

```
PLATFORM_VERSION=4.1.1
TARGET_PRODUCT=full_x86
TARGET_BUILD_VARIANT=eng
TARGET_BUILD_TYPE=release
TARGET_BUILD_APPS=
TARGET_ARCH=x86
TARGET_ARCH_VARIANT=x86-atom
....
Processing
target/product/generic_x86/obj/APPS/MyAppAndroid_intermediates/package.apk
Done!
Install: out/target/product/generic_x86/system/app/MyAppAndroid.odex
Install: out/target/product/generic_x86/system/app/MyAppAndroid.apk
make: Leaving directory `/home/student/aosp'
```

8.ลบไฟล์ system.img ที่อยู่ในไดเรกทอรี ~/aosp/out/target/product/generic_x86 และแฟ้มทั้งหมดภายในไดเรกทอรี ~/aosp/out/target/product/generic_x86/obj/PACKAGING/systemimage_intermediates/

```
$ cd ~/aosp/out/target/product/generic_x86
$ rm -rf system.img obj/PACKAGING/systemimage_intermediates/
```

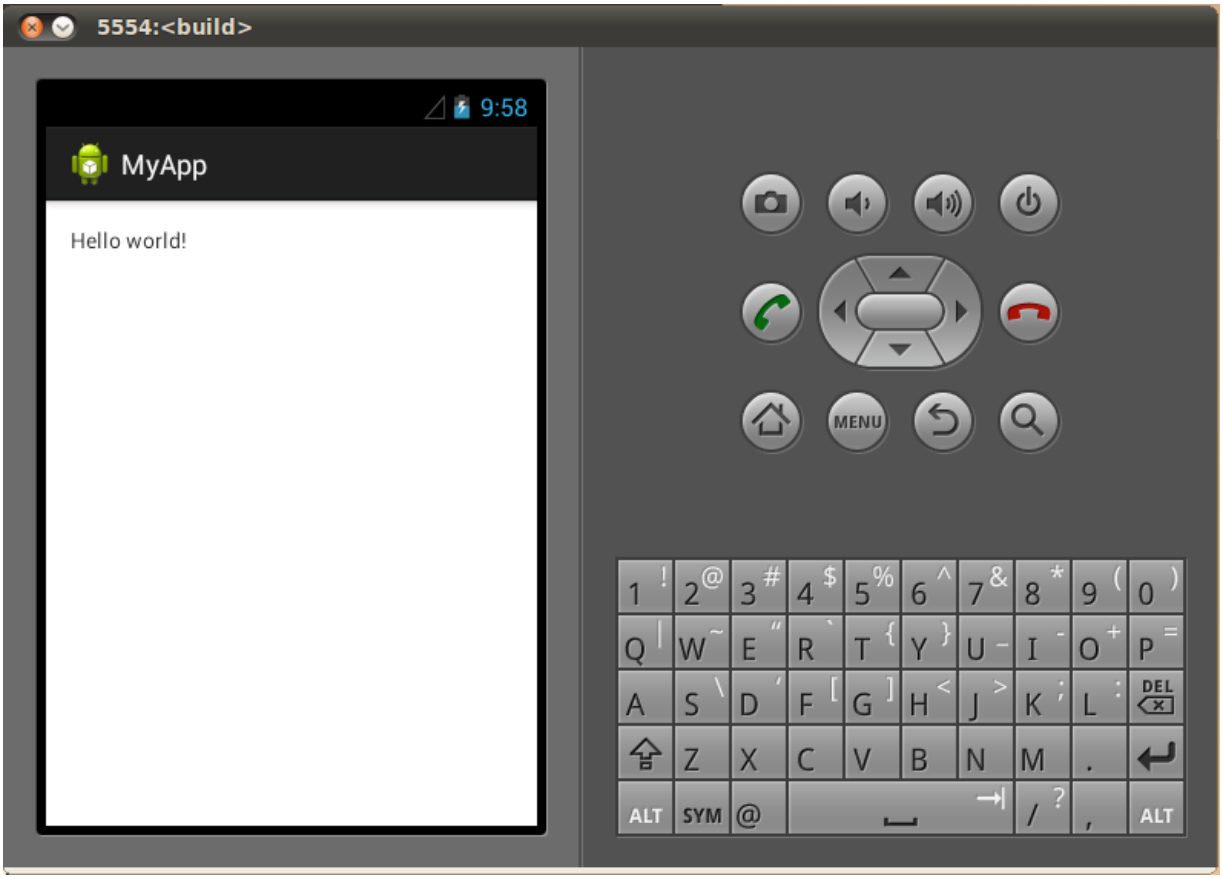
9. คอมไพล์ AOSP ใหม่อีกครั้ง

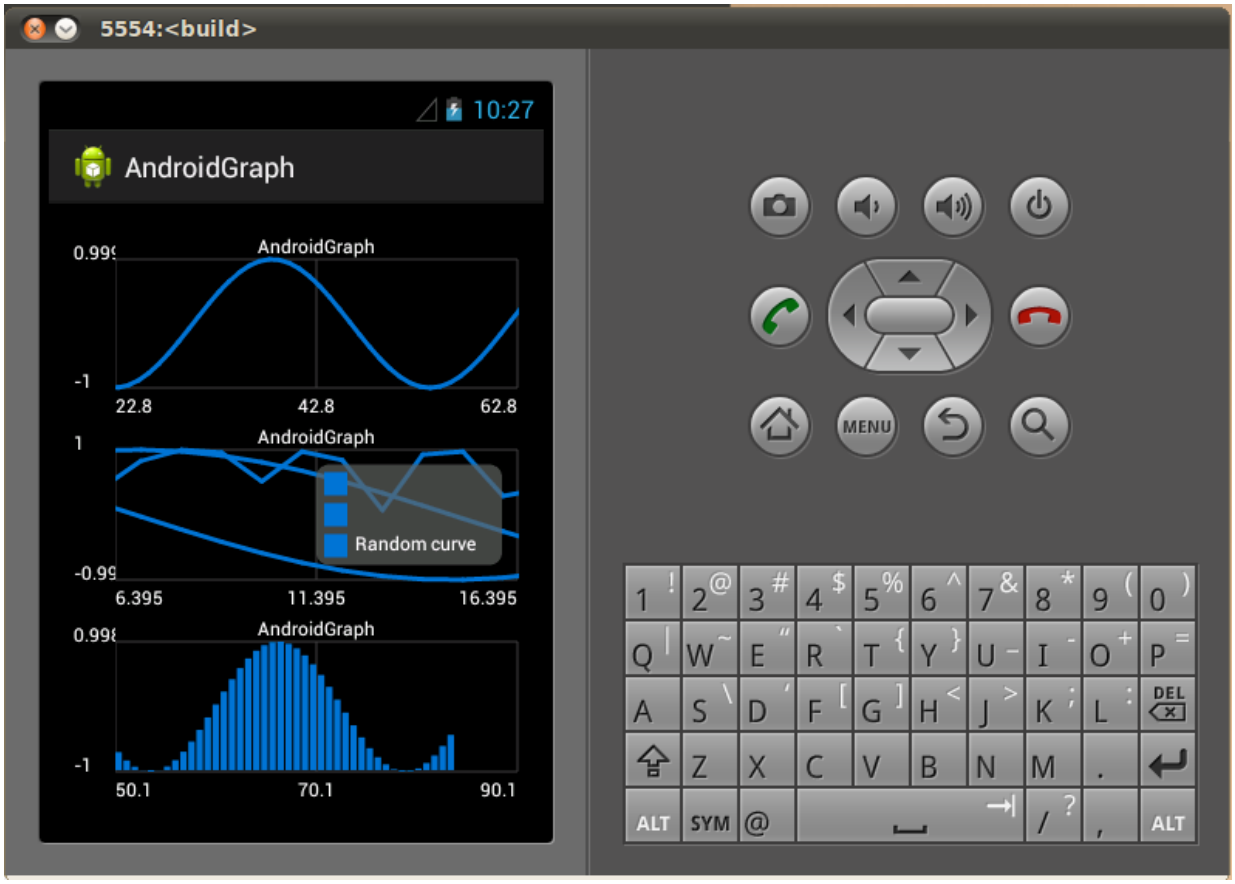
```
$ cd ~/aosp
$ make -j3
ringtones/Rigel.ogg ignored.
PRODUCT_COPY_FILES
frameworks/base/data/sounds/ringtones/ogg/Scarabaeus.ogg:system/media/audio/ringtone
s/Scarabaeus.ogg ignored.
PRODUCT_COPY_FILES
frameworks/base/data/sounds/ringtones/ogg/Solarium.ogg:system/media/audio/ringtones/
Solarium.ogg ignored.
PRODUCT_COPY_FILES
frameworks/base/data/sounds/ringtones/ogg/Themos.ogg:system/media/audio/ringtones/Th
emos.ogg ignored.
PRODUCT_COPY_FILES
frameworks/base/data/sounds/ringtones/ogg/UrsaMinor.ogg:system/media/audio/ringtones
/UrsaMinor.ogg ignored.
PRODUCT_COPY_FILES
frameworks/base/data/sounds/ringtones/ogg/Zeta.ogg:system/media/audio/ringtones/Zeta
.ogg ignored.
Target system fs image:
out/target/product/generic_x86/obj/PACKAGING/systemimage_intermediates/system.img
Running:  mkyaffs2image -f out/target/product/generic_x86/system
out/target/product/generic_x86/obj/PACKAGING/systemimage_intermediates/system.img
Install system fs image: out/target/product/generic_x86/system.img
```

10. ขั้นตอนสุดท้ายทำการเรียกโปรแกรม Android Emulator ขึ้นมาใหม่อีกครั้ง

```
$ emulator -kernel ~/aosp/goldfish/arch/x86/boot/bzImage -system
out/target/product/generic_x86/system.img -ramdisk
out/target/product/generic_x86/ramdisk.img &
```







ชุดเครื่องมือและคำสั่งภายใน Android Emulator

เพื่อศึกษากลไกการทำงานภายใน และสามารถจัดการตัว Android Emulator ได้เป็นอย่างดี นักพัฒนาจะต้องเรียนรู้ชัดคำสั่งอย่างน้อย ดังแสดงในตารางข้างล่างนี้

ตาราง 5.7 ตัวอย่างคำสั่งที่น่าสนใจภายใน Android Emulator

คำสั่ง	รายละเอียด
services	เป็นคำสั่งที่ทำหน้าที่เรียกดู Service ที่อยู่ Android ทั้งหมด
dumpsys	เป็นคำสั่งที่ทำหน้าที่เรียกดูข้อมูลระบบที่เราต้องการเช่นระดับแบตเตอรี่การทำงานของCPU
dumpstate	เป็นคำสั่งที่ทำหน้าที่เรียกสถานะการทำงานของแต่ละโปรแกรมใน Android
rawbu	เป็นคำสั่งที่ใช้ในการ backup และ restore ข้อมูลที่อยู่ /data
am	ตัวจัดการ Activity
pm	ตัวจัดการ Package
dalvikvm	เป็นคำสั่งที่ใช้ในการรันไฟล์ Java Class ในรูปแบบ DEX format

คำสั่ง	รายละเอียด
getprop	เป็นคำสั่งแสดงข้อมูลการระบบ เช่น DNS servers, gateways, GSM, services ที่กำลังทำงาน, build parameters, เวอร์ชั่น เป็นต้น

เราจะเริ่มต้นจากการเรียกโปรแกรม Android Emulator ด้วยคำสั่ง

```
$ cd ~/aosp/
$ source build/envsetup.sh
including device/asus/grouper/vendorsetup.sh
including device/generic/armv7-a-neon/vendorsetup.sh
including device/generic/armv7-a/vendorsetup.sh
including device/moto/wingray/vendorsetup.sh
including device/samsung/crespo/vendorsetup.sh
including device/samsung/maguro/vendorsetup.sh
including device/ti/panda/vendorsetup.sh
including sdk/bash_completion/adb.bash

$ lunch

You're building on Linux
Lunch menu... pick a combo:
     1. full-eng
     2. full_x86-eng
     3. vbox_x86-eng
     4. full_grouper-userdebug
     5. mini_armv7a_neon-userdebug
     6. mini_armv7a-userdebug
     7. full_wingray-userdebug
     8. full_crespo-userdebug
     9. full_maguro-userdebug
     10. full_panda-userdebug

Which would you like? [full-eng] 2

============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=4.1.1
TARGET_PRODUCT=full_x86
TARGET_BUILD_VARIANT=eng
TARGET_BUILD_TYPE=release
TARGET_BUILD_APPS=
TARGET_ARCH=x86
TARGET_ARCH_VARIANT=x86-atom
HOST_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-3.2.17-x86_64-with-Ubuntu-10.04-lucid
HOST_BUILD_TYPE=release
BUILD_ID=JRO03L
OUT_DIR=out
============================================

$ emulator &
```

การกำหนดสิทธิ์ให้กับพาร์ทิชัน (Mounting)

รูปแบบการใช้งานคำสั่ง:

```
mount -o rw,remount -t "filesystem" "Device" "Mount Point"
```

```
$ adb shell
```

การตั้งให้ root file system สามารถเขียนได้

```
# mount -o rw,remount -t rootfs /
```

การตั้งให้พาร์ทิชัน /system สามารถเขียนได้

```
# mount -o rw,remount -t yaffs2 /dev/block/mtdblock4 /system
$ adb shell
```

การเรียกใช้โปรแกรมบริการภายใน Android Emulator

```
# service
Usage: service [-h|-?]
       service list
       service check SERVICE
       service call SERVICE CODE [i32 INT | s16 STR] ...
Options:
   i32: Write the integer INT into the send parcel.
   s16: Write the UTF-16 string STR into the send parcel.

# service list
Found 65 services:
0     phone: [com.android.internal.telephony.ITelephony]
1     iphonesubinfo: [com.android.internal.telephony.IPhoneSubInfo]
2     simphonebook: [com.android.internal.telephony.IIccPhoneBook]
3     isms: [com.android.internal.telephony.ISms]
4     commontime_management: []
5     samplingprofiler: []
6     diskstats: []
7     appwidget: [com.android.internal.appwidget.IAppWidgetService]
8     backup: [android.app.backup.IBackupManager]
9     uimode: [android.app.IUiModeManager]
10    serial: [android.hardware.ISerialManager]
11    usb: [android.hardware.usb.IUsbManager]
12    audio: [android.media.IAudioService]
13    wallpaper: [android.app.IWallpaperManager]
14    dropbox: [com.android.internal.os.IDropBoxManagerService]
15    search: [android.app.ISearchManager]
16    country_detector: [android.location.ICountryDetector]
17    location: [android.location.ILocationManager]
18    devicestoragemonitor: []
19    notification: [android.app.INotificationManager]
20    updatelock: [android.os.IUpdateLock]
21    throttle: [android.net.IThrottleManager]
```

```
    servicediscovery: [android.net.nsd.INsdManager]
    connectivity: [android.net.IConnectivityManager]
    wifi: [android.net.wifi.IWifiManager]
    wifip2p: [android.net.wifi.p2p.IWifiP2pManager]
    netpolicy: [android.net.INetworkPolicyManager]
    netstats: [android.net.INetworkStatsService]
    textservices: [com.android.internal.textservice.ITextServicesManager]
    network_management: [android.os.INetworkManagementService]
    clipboard: [android.content.IClipboard]
    statusbar: [com.android.internal.statusbar.IStatusBarService]
    device_policy: [android.app.admin.IDevicePolicyManager]
    lock_settings: [com.android.internal.widget.ILockSettings]
    mount: [IMountService]
    accessibility: [android.view.accessibility.IAccessibilityManager]
    input_method: [com.android.internal.view.IInputMethodManager]
    input: [android.hardware.input.IInputManager]
    window: [android.view.IWindowManager]
    alarm: [android.app.IAlarmManager]
    vibrator: [android.os.IVibratorService]
    battery: []
    hardware: [android.os.IHardwareService]
    content: [android.content.IContentService]
    account: [android.accounts.IAccountManager]
    permission: [android.os.IPermissionController]
    cpuinfo: []
    dbinfo: []
    gfxinfo: []
    meminfo: []
    activity: [android.app.IActivityManager]
    package: [android.content.pm.IPackageManager]
    scheduling_policy: [android.os.ISchedulingPolicyService]
    telephony.registry: [com.android.internal.telephony.ITelephonyRegistry]
    usagestats: [com.android.internal.app.IUsageStats]
    batteryinfo: [com.android.internal.app.IBatteryStats]
    power: [android.os.IPowerManager]
    entropy: []
    sensorservice: [android.gui.SensorServer]
    media.audio_policy: [android.media.IAudioPolicyService]
    media.camera: [android.hardware.ICameraService]
    SurfaceFlinger: [android.ui.ISurfaceComposer]
    media.player: [android.media.IMediaPlayerService]
    media.audio_flinger: [android.media.IAudioFlinger]
    drm.drmManager: [drm.IDrmManagerService]
```

ตัวอย่างการเรียกใช้งาน

```
# service call statusbar 1
Result: Parcel(00000000    '....')
```




```
# service call statusbar 2
Result: Parcel(00000000    '....')
```



คำสั่ง dumpsys คือเครื่องมือที่ใช้สำหรับเรียกดูข้อมูลภายในสถานะต่างๆ ของตัวให้บริการของระบบ (system service) ตัวอย่างเช่น หน่วยประมวลผลกลาง หน่วยความจำ แบตเตอรี่ ตัวเก็บข้อมูล เป็นต้น

```
$ adb shell dumpsys | grep "DUMP OF SERVICE"
Output Command
DUMP OF SERVICE SurfaceFlinger:
DUMP OF SERVICE accessibility:
DUMP OF SERVICE account:
DUMP OF SERVICE activity:
DUMP OF SERVICE alarm:
DUMP OF SERVICE appwidget:
DUMP OF SERVICE audio:
DUMP OF SERVICE backup:
DUMP OF SERVICE battery:
DUMP OF SERVICE batteryinfo:
DUMP OF SERVICE clipboard:
DUMP OF SERVICE commontime_management:
DUMP OF SERVICE connectivity:
DUMP OF SERVICE content:
DUMP OF SERVICE country_detector:
DUMP OF SERVICE cpuinfo:
DUMP OF SERVICE dbinfo:
DUMP OF SERVICE device_policy:
DUMP OF SERVICE devicestoragemonitor:
DUMP OF SERVICE diskstats:
DUMP OF SERVICE drm.drmManager:
DUMP OF SERVICE dropbox:
DUMP OF SERVICE entropy:
DUMP OF SERVICE gfxinfo:
DUMP OF SERVICE hardware:
DUMP OF SERVICE input:
DUMP OF SERVICE input_method:
DUMP OF SERVICE iphonesubinfo:
DUMP OF SERVICE isms:
DUMP OF SERVICE location:
DUMP OF SERVICE lock_settings:
DUMP OF SERVICE media.audio_flinger:
DUMP OF SERVICE media.audio_policy:
DUMP OF SERVICE media.camera:
DUMP OF SERVICE media.player:
DUMP OF SERVICE meminfo:
DUMP OF SERVICE mount:
DUMP OF SERVICE netpolicy:
DUMP OF SERVICE netstats:
DUMP OF SERVICE network_management:
DUMP OF SERVICE notification:
DUMP OF SERVICE package:
DUMP OF SERVICE permission:
DUMP OF SERVICE phone:
DUMP OF SERVICE power:
DUMP OF SERVICE samplingprofiler:
DUMP OF SERVICE scheduling_policy:
DUMP OF SERVICE search:
DUMP OF SERVICE sensorservice:
DUMP OF SERVICE serial:
DUMP OF SERVICE servicediscovery:
DUMP OF SERVICE simphonebook:
DUMP OF SERVICE statusbar:
DUMP OF SERVICE telephony.registry:
DUMP OF SERVICE textservices:
```

```
DUMP OF SERVICE throttle:
DUMP OF SERVICE uimode:
DUMP OF SERVICE updatelock:
DUMP OF SERVICE usagestats:
DUMP OF SERVICE usb:
DUMP OF SERVICE vibrator:
DUMP OF SERVICE wallpaper:
DUMP OF SERVICE wifi:
DUMP OF SERVICE wifip2p:
DUMP OF SERVICE window:
```

$ adb shell dumpsys battery

```
Current Battery Service state:
  AC powered: true
  USB powered: false
  status: 2
  health: 2
  present: true
  level: 50
  scale: 100
  voltage:0
  temperature: 0
  technology: Li-ion
```

$ adb shell dumpsys wifi

```
Wi-Fi is disabled
Stay-awake conditions: 1
Internal state:
current HSM state: DriverUnloadedState
mLinkProperties LinkAddresses: [] Routes: [] DnsAddresses: []
mWifiInfo SSID: <none>, BSSID: <none>, MAC: <none>, Supplicant state: UNINITIALIZED,
RSSI: -9999, Link speed: -1, Net ID: -1, Metered hint: false
mDhcpInfoInternal addr: null/0 mRoutes:  dns: null,null dhcpServer: null leaseDura-
tion: 0
mNetworkInfo NetworkInfo: type: WIFI[], state: UNKNOWN/IDLE, reason: (unspecified),
extra: (none), roaming: false, failover: false, isAvailable: false
mLastSignalLevel -1
mLastBssid null
mLastNetworkId -1
mReconnectCount 0
mIsScanMode false
Supplicant status
null
mLastPriority -1
Configured networks
Latest scan results:
Locks acquired: 0 full, 0 full high perf, 0 scan
Locks released: 0 full, 0 full high perf, 0 scan
Locks held:
Scan count since last plugged in
WifiWatchdogStateMachine dump
WatchdogStatus: State:
android.net.wifi.WifiWatchdogStateMachine$NotConnectedState@b444f100mWifiInfo:
[null]
mLinkProperties: [null]
mCurrentSignalLevel: [0]
mArpCheckIntervalMs: [120000]
mRssiFetchIntervalMs: [1000]
mWalledGardenIntervalMs: [1800000]
```

```
mNumArppings: [5]
mMinArpResponses: [1]
mArppingTimeoutMs: [100]
mPoorNetworkDetectionEnabled: [true]
mWalledGardenTestEnabled: [true]
mWalledGardenUrl: [http://clients3.google.com/generate_204]
WifiStateMachine dump
WifiStateMachine:
 total messages=3
 msg[0]: time=09-21 16:03:00.308 state=DefaultState orgState=DriverUnloadedState
what=69632(0x11000)
 msg[1]: time=09-21 16:03:00.389 state=DefaultState orgState=DriverUnloadedState
what=131085(0x2000d)
 msg[2]: time=09-21 16:03:00.408 state=DefaultState orgState=DriverUnloadedState
what=131147(0x2004b)
curState=DriverUnloadedState
```

$ adb shell dumpsys cpuinfo

```
Load: 3.24 / 1.4 / 0.51
CPU usage from 533936ms to 513149ms ago:
  23% 2477/system_server: 22% user + 0.7% kernel / faults: 625 minor
  4.8% 2445/surfaceflinger: 4.6% user + 0.1% kernel / faults: 592 minor
  1.5% 2634/android.process.acore: 1.4% user + 0% kernel / faults: 252 minor
  1.9% 2782/android.process.media: 1.7% user + 0.1% kernel / faults: 125 minor
  1.7% 2619/com.android.launcher: 1.4% user + 0.2% kernel / faults: 145 minor
  1.5% 2570/com.android.inputmethod.latin: 1.3% user + 0.1% kernel / faults: 141 mi-
nor
  1.4% 2708/com.android.deskclock: 1.3% user + 0% kernel / faults: 150 minor
  1.3% 2765/com.android.email: 1.3% user + 0% kernel / faults: 106 minor
  1.3% 2851/com.android.providers.calendar: 1.3% user + 0% kernel / faults: 148 mi-
nor
  1.3% 2552/com.android.systemui: 1.2% user + 0% kernel / faults: 60 minor
  0.9% 2731/com.android.contacts: 0.8% user + 0% kernel / faults: 160 minor
  1% 2589/com.android.phone: 0.8% user + 0.1% kernel / faults: 55 minor
  0.7% 2681/zygote: 0.6% user + 0% kernel / faults: 160 minor
  0.9% 2805/com.android.exchange: 0.9% user + 0% kernel / faults: 120 minor
  0.5% 2659/com.android.smspush: 0.4% user + 0% kernel / faults: 165 minor
  0.8% 2878/com.android.calendar: 0.8% user + 0% kernel / faults: 99 minor
  0.5% 806/adbd: 0% user + 0.5% kernel / faults: 11 minor
  0.1% 2428/mediaserver: 0% user + 0% kernel / faults: 27 minor
  0% 4/events/0: 0% user + 0% kernel
  0% 783/servicemanager: 0% user + 0% kernel
 +0% 2900/sh: 0% user + 0% kernel
 +0% 2902/dumpsys: 0% user + 0% kernel
48% TOTAL: 45% user + 3.3% kernel
```

$ adb shell dumpsys meminfo

```
Applications Memory Usage (kB):
Uptime: 9118113 Realtime: 9118113
Total PSS by process:
    22980 kB: system (pid 2477)
    19899 kB: com.android.settings (pid 2681)
    19648 kB: com.android.launcher (pid 2619)
    12665 kB: com.android.systemui (pid 2552)
     8792 kB: com.android.phone (pid 2589)
     7504 kB: com.android.contacts (pid 2731)
     7085 kB: com.android.inputmethod.latin (pid 2570)
     6619 kB: android.process.acore (pid 2634)
     5815 kB: com.android.email (pid 2765)
     5402 kB: android.process.media (pid 2782)
```

```
     5328 kB: com.android.calendar (pid 2878)
     4603 kB: com.android.providers.calendar (pid 2851)
     4287 kB: com.android.deskclock (pid 2708)
     4282 kB: com.android.exchange (pid 2805)
     2824 kB: com.android.smspush (pid 2659)
     2686 kB: com.android.defcontainer (pid 2965)

Total PSS by OOM adjustment:
    22980 kB: System
               22980 kB: system (pid 2477)
    21457 kB: Persistent
               12665 kB: com.android.systemui (pid 2552)
                8792 kB: com.android.phone (pid 2589)
    19648 kB: Foreground
               19648 kB: com.android.launcher (pid 2619)
     2824 kB: Visible
                2824 kB: com.android.smspush (pid 2659)
     7085 kB: Perceptible
                7085 kB: com.android.inputmethod.latin (pid 2570)
    22585 kB: Previous
               19899 kB: com.android.settings (pid 2681)
                2686 kB: com.android.defcontainer (pid 2965)
    43840 kB: Background
                7504 kB: com.android.contacts (pid 2731)
                6619 kB: android.process.acore (pid 2634)
                5815 kB: com.android.email (pid 2765)
                5402 kB: android.process.media (pid 2782)
                5328 kB: com.android.calendar (pid 2878)
                4603 kB: com.android.providers.calendar (pid 2851)
                4287 kB: com.android.deskclock (pid 2708)
                4282 kB: com.android.exchange (pid 2805)

Total PSS by category:
    53374 kB: Dalvik
    23817 kB: Unknown
    19418 kB: Other mmap
    18091 kB: Native
    11556 kB: .apk mmap
    11425 kB: .so mmap
     1936 kB: Ashmem
      718 kB: .ttf mmap
       68 kB: Other dev
       12 kB: Cursor
        4 kB: .jar mmap
        0 kB: .dex mmap

Total PSS: 140419 kB
      KSM: 0 kB saved from shared 0 kB
           0 kB unshared; 0 kB volatile
```

$ adb shell dumpsys meminfo com.android.providers.calendar

```
Applications Memory Usage (kB):
Uptime: 8961566 Realtime: 8961566
** MEMINFO in pid 2851 [com.android.providers.calendar] **
                     Shared  Private     Heap     Heap     Heap
                Pss    Dirty    Dirty     Size    Alloc     Free
               ------   ------   ------   ------   ------   ------
       Native      806      684      760     1908     1692      143
       Dalvik     2145     8904     1608     6407     6093      314
       Cursor        0        0        0
       Ashmem        0        0        0
```

```
    Other dev        4         0         0
     .so mmap      522      1828       216
    .jar mmap        4         0         0
    .apk mmap       95         0         0
    .ttf mmap        0         0         0
    .dex mmap        0         0         0
   Other mmap      912        24        24
      Unknown      177        36       176
        TOTAL     4665     11476      2784     8315     7785      457
 Objects
               Views:        0         ViewRootImpl:        0
         AppContexts:        1           Activities:        0
              Assets:        2        AssetManagers:        2
       Local Binders:       10        Proxy Binders:        8
    Death Recipients:        0
     OpenSSL Sockets:        0
 SQL
         MEMORY_USED:      171
  PAGECACHE_OVERFLOW:       53          MALLOC_SIZE:       62
 DATABASES
      pgsz     dbsz   Lookaside(b)          cache  Dbname
         4      120            117       24/36/11
/data/data/com.android.providers.calendar/databases/calendar.db
 Asset Allocations
    zip:/system/app/CalendarProvider.apk:/resources.arsc: 57K
```

เพื่อให้สามารถเรียกใช้คำสั่ง dumpsys ได้จะต้องเพิ่มบรรทัด <uses-permission android:name="android.permission.DUMP" /> เข้าไปในไฟล์ AndroidManifest.xml

คำสั่ง dumpstate ใช้สำหรับดูสถานะการทำงานของแอนดรอยด์ทั้งหมด

```
$ adb shell
# dumpstate
......
PROVIDER ContentProviderRecord{b4669e98
com.android.email/.provider.AttachmentProvider} pid=1306
    Client:
      nothing to dump
PROVIDER ContentProviderRecord{b470bc88
com.android.providers.contacts/.CallLogProvider} pid=1151
    Client:
      nothing to dump
PROVIDER ContentProviderRecord{b462a2a0
com.android.providers.downloads/.DownloadProvider} pid=1324
    Client:
      nothing to dump
PROVIDER ContentProviderRecord{b46e1b68
com.android.providers.userdictionary/.UserDictionaryProvider} pid=1151
    Client:
      nothing to dump
[dumpsys: 0.4s elapsed]
============================================================
== dumpstate: done
============================================================
```

คำสั่ง dalvikvm (Dalvik VM) เป็นคำสั่งที่อนุญาตให้ผู้ใช้สามารถเรียกโปรแกรม Dalvik VM เพื่อรันไฟล์ Java class ที่เป็นรูปแบบ DEX ดังตัวอย่างข้างล่าง

```
$ mkdir dalvik
$ cd dalvik/
```

สร้างโปรแกรมอย่างง่าย

```
$ cat hello.java
package org.apache;

public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

ทำการคอมไพล์โปรแกรมด้วย JAVA Compiler

```
$ javac -d . -g Hello.java
```

ทำการรวมไฟล์ที่ได้จากการคอมไพล์ทั้งหมดมาเป็นไฟล์ .jar

```
$ jar -cvf Temp.jar *
added manifest
adding: Hello.java(in = 139) (out= 115)(deflated 17%)
adding: org/(in = 0) (out= 0)(stored 0%)
adding: org/apache/(in = 0) (out= 0)(stored 0%)
adding: org/apache/Hello.class(in = 541) (out= 336)(deflated 37%)
```

ใช้โปรแกรม “dx” เพื่อสร้างไฟล์ classes.dex จากไฟล์ Temp.jar

```
$ dx --dex --output=./classes.dex Temp.jar
$ ls
classes.dex  Hello.java  org  Temp.jar
```

ถัดไปใช้โปรแกรม “aapt” เพื่อสร้างไฟล์ .jar ตัวใหม่ เพื่อให้สามารถทำงานภายใต้ Dalvik VM ได้

```
$ aapt add CmdLine.jar classes.dex
 'classes.dex'...
```

นำไฟล์ CmdLine.jar ที่บรรจุไฟล์ classes.dex เข้าสู่ตัว Android Emulator แล้วทำการทดสอบเรียกโปรแกรมด้วย Dalvikvm ดังคำสั่งข้างล่างนี้

```
$ adb push CmdLine.jar /data
```

```
10 KB/s (547 bytes in 0.050s)

$ adb shell

# /system/bin/dalvikvm -Xbootclasspath:/system/framework/core.jar
-version
DalvikVM version 1.6.0
Copyright (C) 2007 The Android Open Source Project
This software is built from source code licensed under the Apache Licen-
se,Version 2.0 (the "License"). You may obtain a copy of the License at
     http://www.apache.org/licenses/LICENSE-2.0
See the associated NOTICE file for this software for further details.
Dalvik VM init failed (check log file)

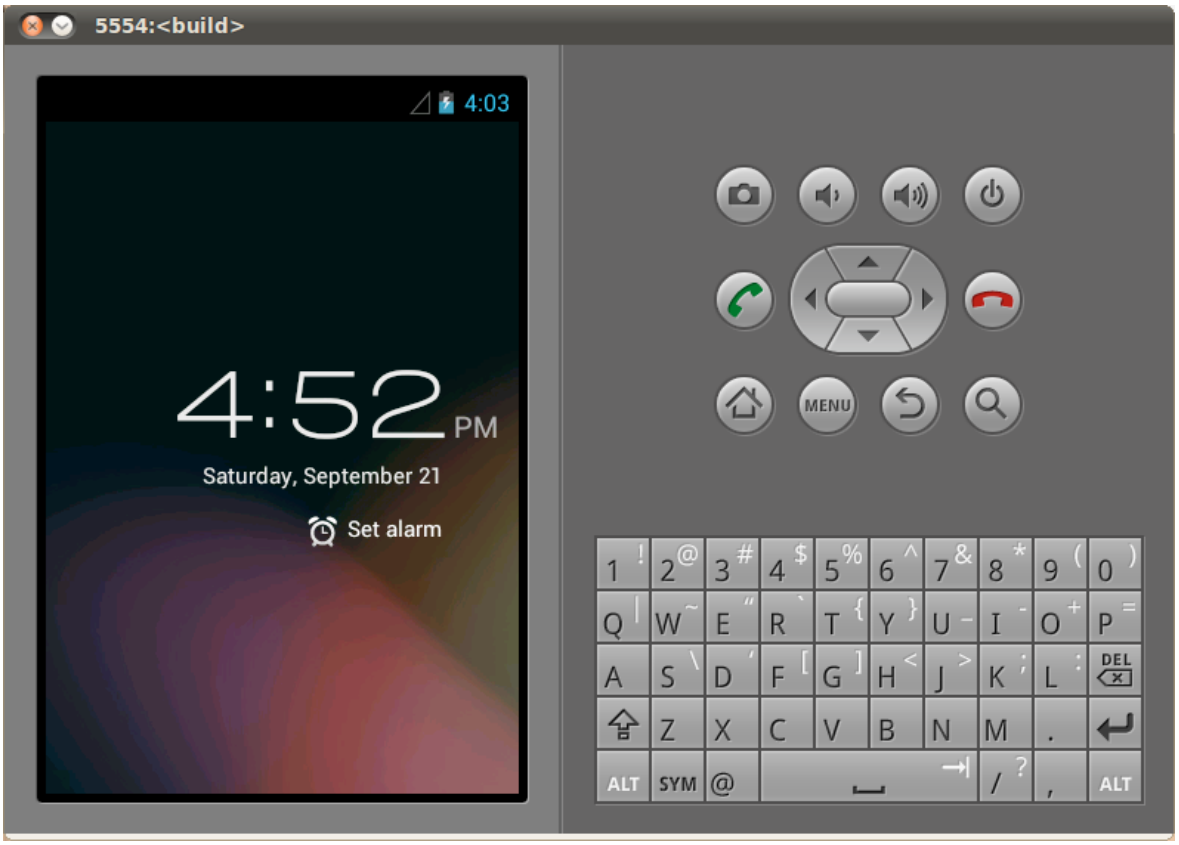
# /system/bin/dalvikvm -Xbootclasspath:/system/framework/core.jar
-classpath /data/CmdLine.jar org.apache.Hello
Hello World!
```

am (Activity Manager)

คำสั่ง am จะสามารถใช้ในการจัดการแอคทิวิตี้ (activities) เช่น เรียกให้ทำงาน สั่งหยุดการทำงาน และเฝ้าดูสถานะการทำงาน เป็นต้น

```
# am
usage: am [subcommand] [options]
usage: am start [-D] [-W] [-P <FILE>] [--start-profiler <FILE>]
               [--R COUNT] [-S] [--opengl-trace] <INTENT>
       am startservice <INTENT>
       am force-stop <PACKAGE>
       am kill <PACKAGE>
       am kill-all
       am broadcast <INTENT>
....
    [--activity-single-top] [--activity-clear-task]
    [--activity-task-on-home]
    [--receiver-registered-only] [--receiver-replace-pending]
    [--selector]
    [<URI> | <PACKAGE> | <COMPONENT>]

# am start com.android.deskclock .Clock
Starting: Intent { act=android.intent.action.MAIN
cat=[android.intent.category.LAUNCHER] pkg=com.android.deskclock }
Warning: Activity not started, its current task has been brought to the front
```




```
# am start -a android.intent.action.MAIN -n
com.android.browser/.BrowserActivity
Starting: Intent { action=android.intent.action.MAIN
comp={com.android.browser/com.android.browser.BrowserActivity} }
Warning: Activity not started, its current task has been brought to the front

# am start -a android.intent.action.MAIN -n
com.android.settings/.Settings
Starting: Intent { action=android.intent.action.MAIN
comp={com.android.settings/com.android.settings.Settings} }
```

คำสั่งในการจัดการแพ็กเก็ต (package manager)

```
# pm list package
package:android
package:com.android.backupconfirm
package:com.android.bluetooth
package:com.android.browser
package:com.android.calculator2
....
package:com.android.wallpaper.holospiral
package:com.android.wallpaper.livepicker
package:com.example.myapp
package:com.svox.pico
package:jp.co.omronsoft.openwnn
```

เมื่อต้องการติดตั้งโปรแกรม (.apk) เพิ่ม สามารถใช้คำสั่ง pm install ได้ดังตัวอย่างข้างล่าง

```
# pm install /sdcard/com.twitter.android-1.apk
```

ตรวจสอบโปรแกรมที่ได้ติดตั้งไปก่อนหน้านี้

```
# pm list packages | grep twitter
```

แสดงไดเรกทอรี ที่เก็บโปรแกรมที่ได้ติดตั้งไป

```
# pm path com.twitter.android
```

คำสั่ง svc (service control) ใช้สำหรับดูสถานะการทำงานและควบคุมการทำงานของบริการภายใน

```
# svc help
Available commands:
    help     Show information about the subcommands
    power    Control the power manager
    data     Control mobile data connectivity
    wifi     Control the Wi-Fi manager
    usb      Control Usb state

# svc power
Control the power manager
usage: svc power stayon [true|false|usb|ac]
         Set the 'keep awake while plugged in' setting.

# svc data
Control mobile data connectivity
usage: svc data [enable|disable]
         Turn mobile data on or off.
       svc data prefer
          Set mobile as the preferred data network
# svc wifi
Control the Wi-Fi manager
usage: svc wifi [enable|disable]
         Turn Wi-Fi on or off.

       svc wifi prefer
          Set Wi-Fi as the preferred data network

# svc usb
Control Usb state
usage: svc usb setFunction [function]
         Set the current usb function.

       svc usb getFunction
          Gets the list of currently enabled functions
```

บทที่ 6 พื้นฐานการเขียนโปรแกรมภาษาจาวาบนแอนดรอยด์สำหรับนักพัฒนา

ระบบปฏิบัติการแอนดรอยด์ถือได้ว่าเป็นหนึ่งในระบบปฏิบัติการสำหรับอุปกรณ์ชนิดพกพาที่ประสบความสำเร็จสูงสุดเท่าที่เคยมีมาและยังเป็นตัวแทนสำคัญของวงการระบบสมองกลฝังตัวที่พิสูจน์ให้เห็นถึงศักยภาพของระบบสมองกลฝังตัวและบทบาทที่มีความสำคัญต่อมวลมนุษยชาติ ทั้งในโลกของการสื่อสาร โปรแกรมประยุกต์ที่หลากหลายและมอบโอกาสที่สำคัญทางด้านระบบสมองกลฝังตัวแก่เหล่านักพัฒนาทั่วทุกมุมโลกรวมทั้งบริษัทผู้ผลิตในอุตสาหกรรมอุปกรณ์อิเล็กทรอนิกส์ได้มีโอกาสสร้างสรรผลิตภัณฑ์ที่รวมเทคโนโลยีด้านต่างๆของระบบสมองกลฝังตัวไม่ว่าจะเป็นระบบปฏิบัติการ ระบบโปรแกรมระบบ โปรแกรมประยุกต์ และบอร์ดสมองกลให้สามารถทำงานได้ร่วมกันได้อย่างลงตัว และสามารถตอบสนองการใช้งานของผู้คนในปัจจุบัน สามารถตอบโจทย์ระดับภาคอุตสาหกรรม รวมถึงสามารถทำให้เกิดผลิตภัณฑ์ที่ยกระดับการแข่งขันให้กับประเทศเหล่านั้นได้อย่างที่ไม่เคยมีมาก่อนและเป็นที่ทราบกันดีว่าต้นน้ำเทคโนโลยีส่วนใหญ่จะเกิดขึ้นจากประเทศที่พัฒนาแล้ว ดังนั้นประเทศที่กำลังพัฒนาส่วนใหญ่จะต้องอยู่ในฐานะผู้ใช้มาโดยตลอดรวมทั้งคนที่ต้องซื้อเทคโนโลยีมาเพื่อพัฒนาประเทศของตน แต่อย่างไรก็ตามเมื่อยุคของซอฟท์แวร์เสรี (Open Source) ระบบปฏิบัติการแบบเปิด (Open Operating System) แหล่งความรู้อันมหาศาลในโลกอินเตอร์เน็ต รวมถึงการออกแบบพัฒนาฮาร์ดแวร์แบบเปิด (Open Hardware) ได้มาถึงจุดลงตัวในช่วงไม่กี่ปีที่ผ่านมา ก็ทำให้เกิดโอกาสที่สำคัญในการนำองค์ความรู้เหล่านี้มาประยุกต์เพื่อเป็นพื้นฐานในการต่อยอดการพัฒนาต่างๆได้อีกมากมายหลังจากนี้

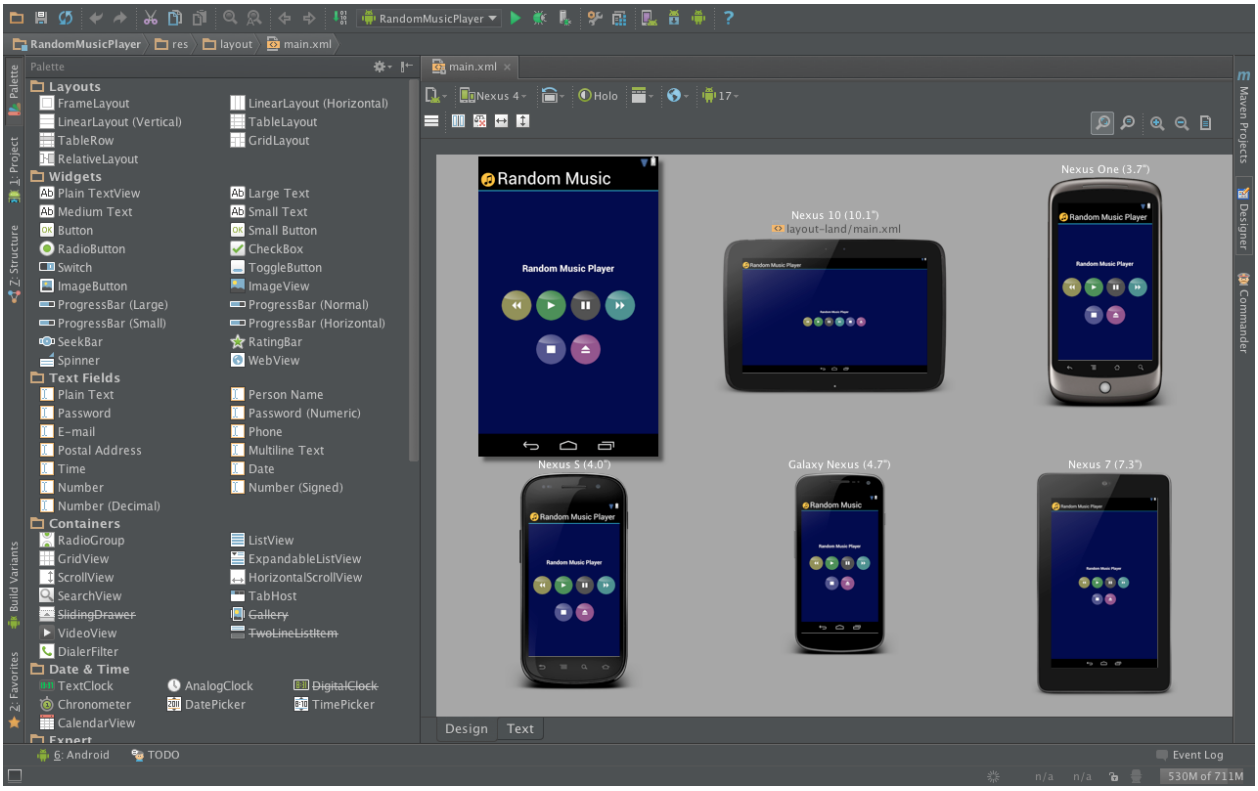


ดังนั้นถ้านักพัฒนาคนไทยพยายามทำความเข้าใจในศาสตร์ด้านระบบสมองกลฝังตัวอย่างจริงจังและนานเพียงพอทั้งยังสามารถมองเห็นถึงความสำคัญของการนำระบบสมองกลผังตัวที่มีระบบปฏิบัติการแอนดรอยด์เพื่อมาแก้ปัญหาระบบโครงสร้างพื้นฐานทางด้านอุตสาหกรรมในเมืองไทยอย่างแท้จริง รวมทั้งถ้าได้รับการสนับสนุนทางด้านผลิตภัณฑ์ที่เกิดจากการพัฒนาของคนไทยหน่วยงานองค์กรภาครัฐหรือภาคเอกชนด้วยแล้ว ผู้เขียนก็มีความเชื่อมั่นว่าในอีกไม่กี่ปีข้างหน้าประเทศไทยก็อาจมีโอกาสสร้างผลิตภัณฑ์ที่สามารถตอบโจทย์ความต้องการพื้นฐานของประเทศโดยไม่ต้องพึ่งพาเทคโนโลยีจากประเทศอื่นๆมากเกินไป ทั้งยังสามารถแข่งขันได้ในระดับนานาชาติได้อย่างภาคภูมิใจได้

ดังนั้นนักพัฒนาทางด้านระบบสมองกลฝังตัวจึงถือได้ว่าเป็นกลไกสำคัญของการขับเคลื่อนการพัฒนาเทคโนโลยีพื้นฐานของประเทศเพื่อให้เกิดการนำไปสู่การประยุกต์ใช้ในด้านการเกษตรกรรมและอุตสาหกรรมในด้านต่างๆได้อย่างทั่วถึง ดังนั้นนักพัฒนาจะต้องเข้าใจหลักการหลายๆอย่างโดยเฉพาะอย่างยิ่งพื้นฐานต่างๆของระบบปฏิบัติการลีนุกซ์ไม่ว่าจะเป็นการใช้คำสั่งตั้งแต่ระดับพื้นฐานจนถึงระดับสูง พื้นฐานการเขียนโปรแกรม พื้นฐานการเชื่อมต่อระหว่างระบบฮาร์ดแวร์และระบบโปรแกรมให้เป็นอย่างดี รวมถึงรู้แนวทางวิธีการแก้ไขปรับแต่งระบบปฏิบัติการลีนุกซ์และระบบปฏิบัติการแอนดรอยด์จากที่ได้กล่าวในบทที่ผ่านมา เพื่อให้นักพัฒนาสามารถเชื่อมโยงศาสตร์ความรู้ที่เกี่ยวข้องต่างๆ (เช่น คอมพิวเตอร์ อิเล็กทรอนิกส์ และระบบสื่อสารเครือข่าย) เข้าด้วยกัน

เครื่องมือพัฒนา Android Studio IDE

ในงาน Google I/O 2013 ที่ผ่านมาทางบริษัทกูเกิลก็ได้เปิดตัวเครื่องมือตัวใหม่ชื่อว่า Android Studio IDE ที่ถูกพัฒนาอยู่บนพื้นฐานของ IntelliJ IDEA เพื่อมาแทนที่เครื่องมือพัฒนาตัวเดิมที่ใช้โปรแกรม Eclipse IDE โดยข้อดีของ Android Studio IDE คือทำงานได้เร็วขึ้น เป็นโปรแกรมลักษณะ Live Layout, Live Coding (WYSIWYG Editor) ซึ่งจะทำให้นักพัฒนาสามารถเห็นภาพรวมของโปรแกรมที่กำลังพัฒนาอยู่บนอุปกรณ์แอนดรอยด์ที่มีขนาดหน้าจอแตกต่างกัน สามารถดูรูปภาพและข้อความที่เราอ้างอิงจากไฟล์อื่น (Preview) และมีเครื่องมือตรวจสอบโค้ดโปรแกรม (Lint tools) ครบถ้วน เช่นสามารถดูประสิทธิภาพการทำงานของโปรแกรม การรองรับเวอร์ชันรุ่นก่อนๆ เป็นต้น



รูปที่ 6.1 หน้าตาโปรแกรม Android Studio

วิธีการติดตั้ง ANDROID STUDIO IDE

สำหรับระบบปฏิบัติการ Windows:

ดาวน์โหลดไฟล์ EXE android-studio-bundle-<version>.exe ในกรณีที่กำลังติดตั้งอยู่มีการร้องขอให้ติดตั้ง JAVA เพิ่มเติมหรืออาจจะทำการตั้งค่าตัวแปรสภาพแวดล้อม (environment variables) โดยการเลือก Start menu > Computer > System Properties > Advanced System Properties เลือกแทบ Advanced > Environment Variables กดปุ่ม “add a new system variable” จากนั้นให้ใส่ path ของ JAVA_HOME เช่น C:\Program Files\Java\jdk1.7.0_21.

สำหรับระบบปฏิบัติการ MacOS X:

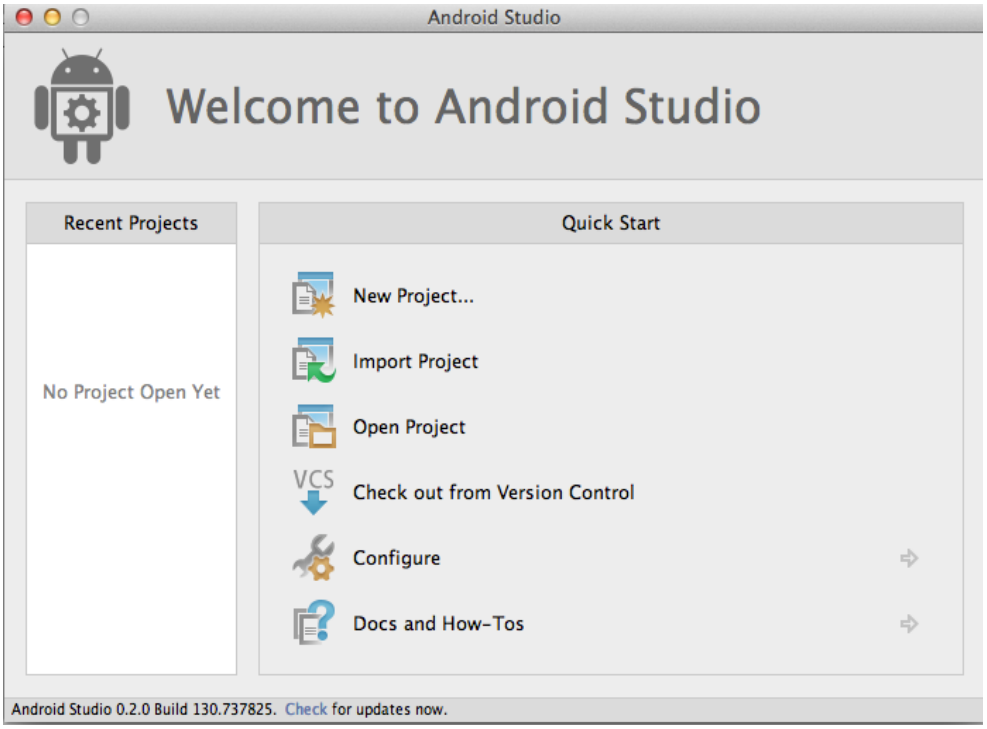
ดาวน์โหลดไฟล์ dmg android-studio-bundle-<version>.dmg ซึ่งปัญหาที่พบเจอบ่อยครั้งคือการขอสิทธิ์อนุญาต (Permissions) ให้ทำการตั้งค่าดังนี้ เลือกที่ System Preferences > Security & Privacy ที่แทบของ Allow applications downloaded ให้ทำการเลือก Anywhere

สำหรับระบบปฏิบัติการลีนุกซ์:

ดาวน์โหลดไฟล์ android-studio-bundle-<version>.tgz เมื่อดาวน์โหลดเสร็จเรียบร้อยก็ให้แตกไฟล์ แล้วรันคำสั่ง ./studio.sh เมื่อติดตั้งโปรแกรมเสร็จแล้วให้ตั้งค่าตัวแปรระบบในไฟล์ ~/.bashrc เช่น PATH เพื่อชี้ไปยังไดเรกทอรี android-studio/bin/

เริ่มต้นการใช้งานโปรแกรม

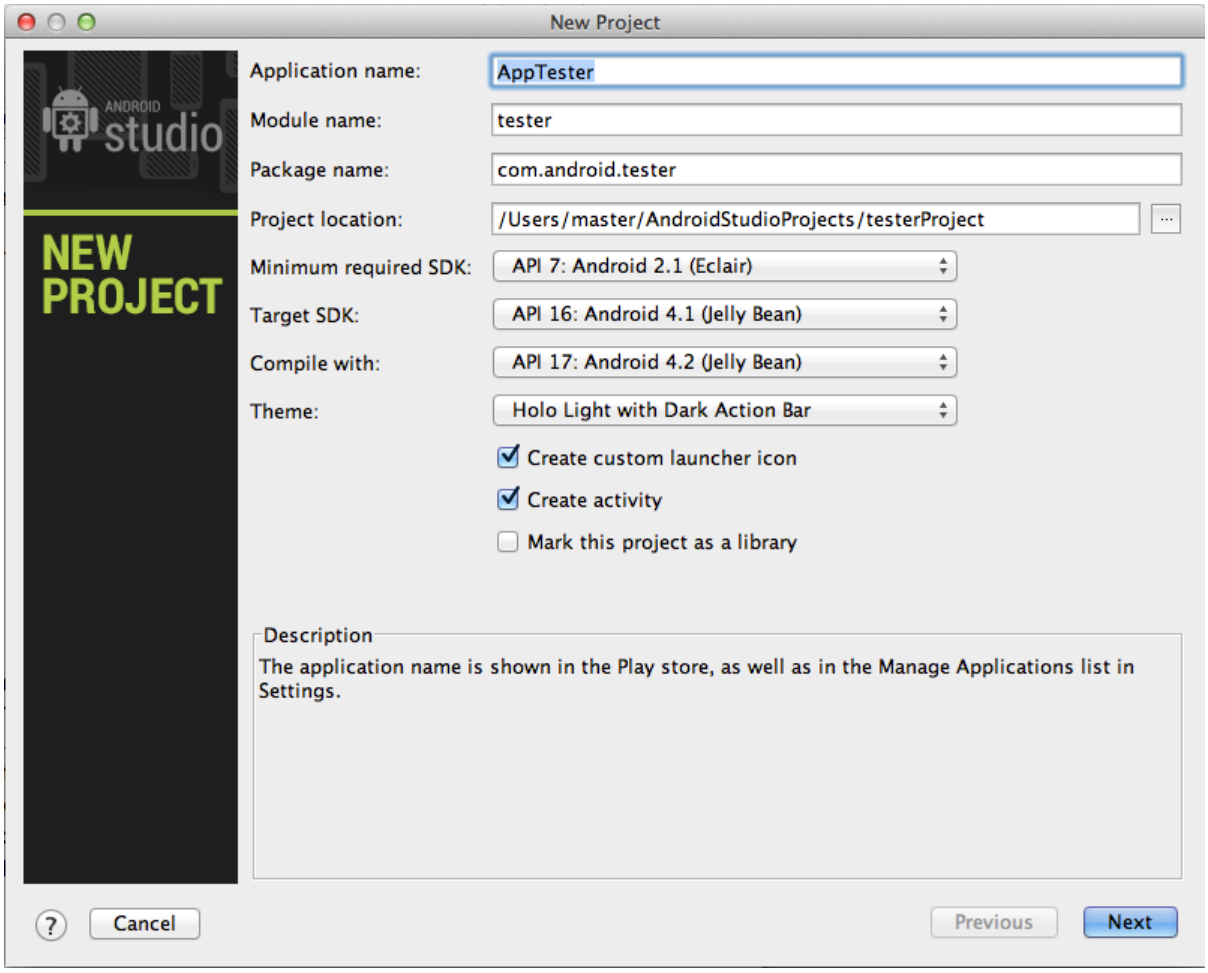
เมื่อเข้าโปรแกรม Android Studio ครั้งแรกก็จะแสดงหน้าต่างดังรูปข้างล่าง



รูปที่ 6.2 หน้าตาสำหรับสร้างหรือนำเข้าโปรแกรม

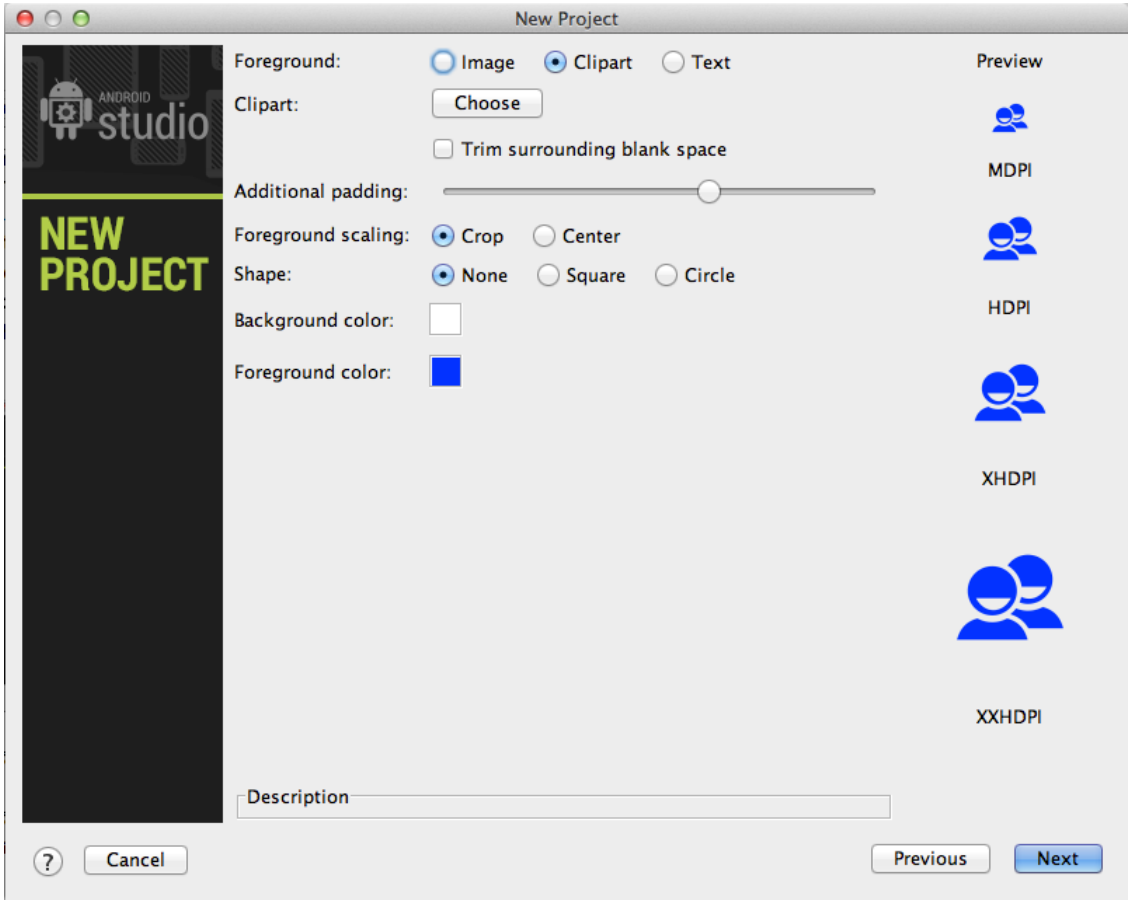
ทดสอบสร้าง Project ใหม่ (New Project...) ซึ่งเมื่อเข้าไปแล้วจะมีการตั้งค่าพื้นฐานต่างๆ ดังแสดงในรูปข้างล่าง เช่น

Application name:	*ชื่อของโปรแกรม*
Module name:	
Package name:	*ชื่อของ Package ที่ใช้งาน*
Project location:	*ที่อยู่ของโปรเจค*
Minimum required SDK:	*เวอร์ชั่นของ SDK ต่ำสุดที่สามารถใช้งานได้*
Target SDK:	*เวอร์ชั่นของ SDK ที่ต้องการใช้งาน*
Compile with:	*เวอร์ชั่นของ SDK ที่ทำการรันโปรแกรม*
Theme:	*ลักษณะธีมของโปรแกรม*



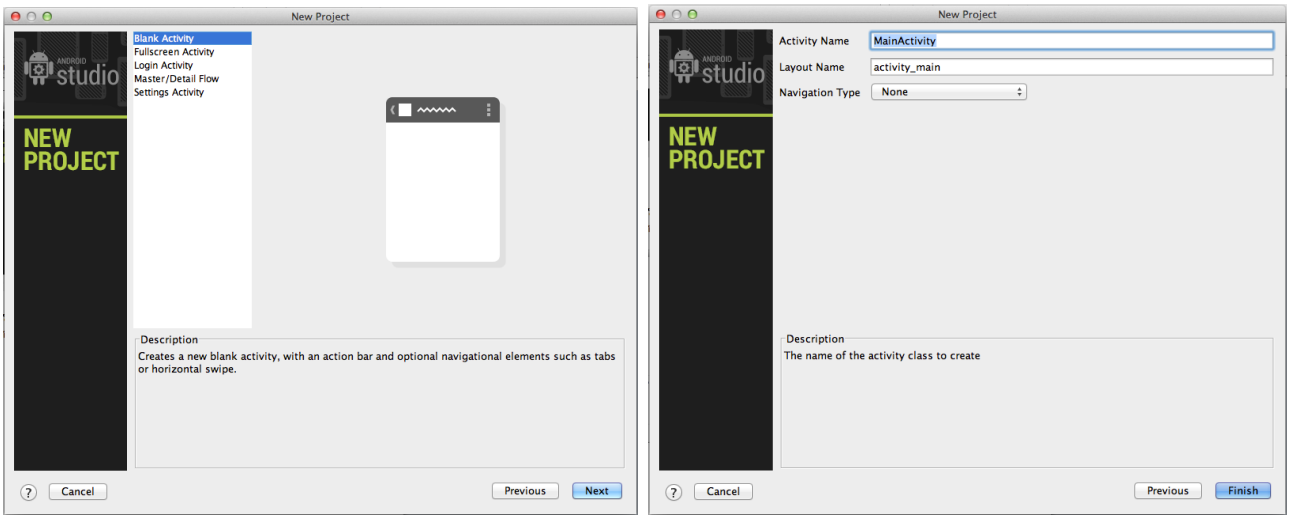
รูปที่ 6.3 ตั้งค่ารายละเอียดของโปรแกรม

ขั้นตอนถัดไปโปรแกรมจะให้ผู้ใช้ทำการเลือกไอคอน ที่ใช้เป็นไอคอนของโปรแกรม



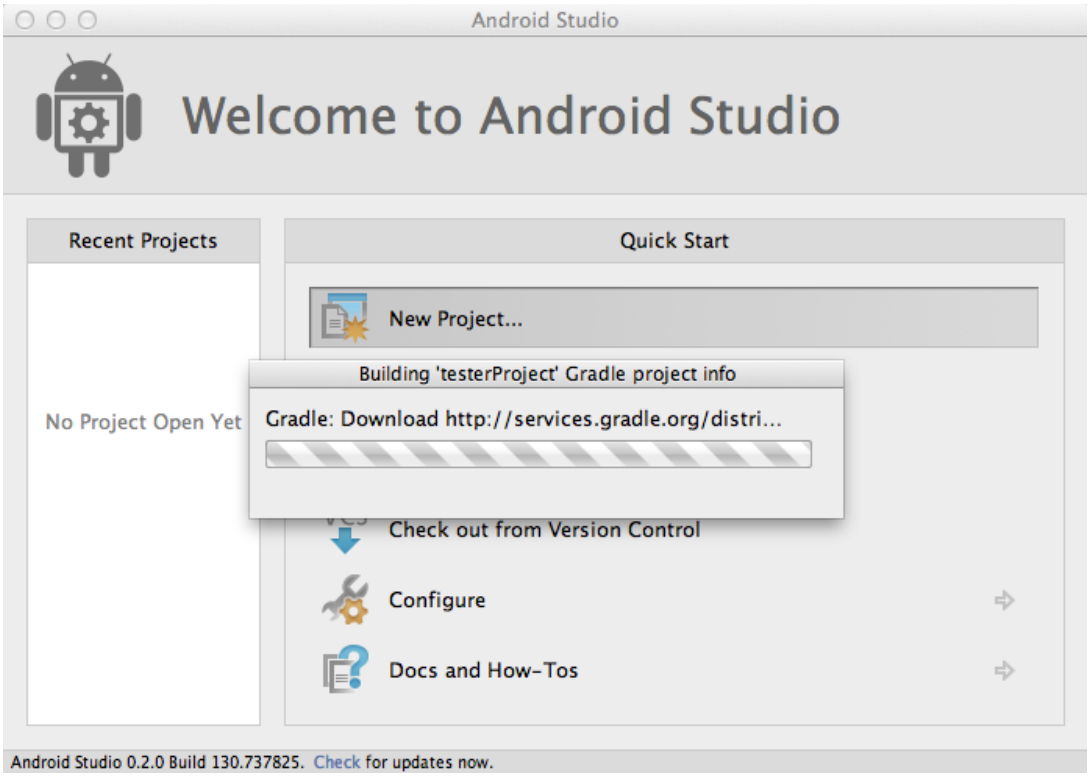
รูปที่ 6.4 กำหนดรูปไอคอนโปรแกรมและค่าทางกราฟฟิกต่างๆ

สำหรับในขั้นตอนนี้นักพัฒนาสามารถเลือกชนิดของโปรแกรมเบื้องต้นได้หลายแบบ ตัวอย่างเช่น Blank Activity, Fullscreen Activity, Login Activity, Master/Detail Flow หรือ Settings Activity โดยในที่นี้จะขอเลือก Blank Activity และตั้งชื่อ Activity หลักของโปรแกรมว่า MainActivity จากนั้นกดปุ่ม Finish เพื่อสร้างโค้ดโปรแกรม



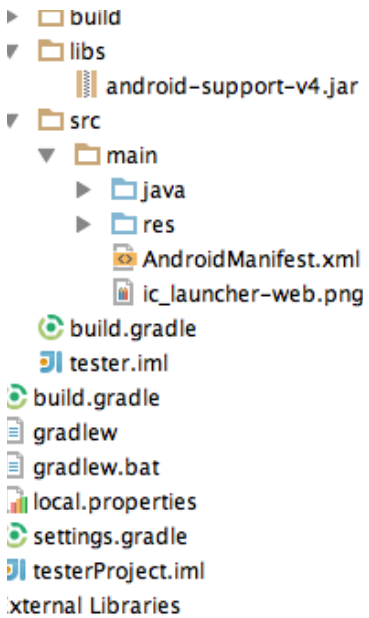
รูปที่ 6.5 เลือก template ของโปรแกรมและตั้งค่า Activity

โปรแกรม Android Studio จะทำการสร้างโปรเจค และจัดเตรียมไฟล์ที่สำคัญสำหรับโปรเจคตามที่ตั้งไว้



รูปที่ 6.6 ดาวน์โหลดองค์ประกอบผ่าน Gradle และสร้างโครงสร้างโปรเจ็ค

รายการโค้ดโปรแกรมและไฟล์ต่างๆ ดังแสดงในรูปข้างล่างนี้

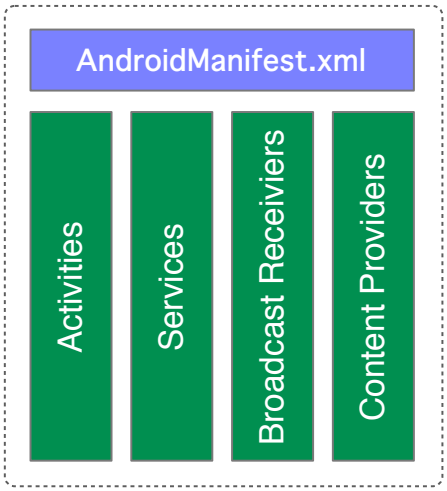


รูปที่ 6.7 แสดงโครงสร้างภายในโปรเจ็ค

ข้อมูลภายในไดเรกทอรี res (Resource) ของโปรแกรมจะประกอบด้วยไฟล์ประเภท XML ซึ่งไฟล์ที่เก็บข้อมูลของ Layout ทั้งหมดที่ใช้ในโปรแกรม และในไดเรกทอรี res จะเป็นที่เก็บไฟล์มัลติมีเดียต่างๆ เช่น ไฟล์ภาพ ไฟล์เสียง เมื่อทำการคอมไพล์ไฟล์โปรแกรม ข้อมูลทั้งหมดในไดเรกทอรี res ก็จะถูกอ้างอิงรวมกับไฟล์ที่ถูกสร้างขึ้นมาในระหว่างการคอมไพล์ที่ชื่อว่า R.java

การกำหนดคุณลักษณะในไฟล์ AndroidMainfest.xml

ไฟล์สำคัญไฟล์หนึ่งคือไฟล์ AndroidManifest.xml ซึ่งเป็นไฟล์ที่กำหนดคุณลักษณะต่างๆภายในโปรแกรมประยุกต์นั้นๆ ซึ่งประกอบไปด้วย Activities, Services, Broadcast Receivers และ Content Providers ดังรูปข้างล่าง



รูปที่ 6.8 องค์ประกอบภายในไฟล์ AndroidManifest.xml

นอกจากนั้นการพัฒนาโปรแกรมแอนดรอยด์บ่อยครั้งที่จะมีการขอเข้าถึงทรัพยากรต่างๆในระบบ ดังนั้นโปรแกรมเหล่านี้จะต้องมีการประกาศการร้องขอสิทธิ์ ไว้ภายในไฟล์ชื่อว่า AndroidManifest.xml ด้วย

ตัวอย่างไฟล์ AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.androidaudiostream"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
```

```
            android:name="com.example.androidaudiostream.ActivityAudioDecode"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

จากตัวอย่างของ XML ด้านบนเป็นตัวอย่างของไฟล์ AndroidManifest.xml ที่มีอีลีเมนต์ชื่อว่า manifest โดยหลักจะมีการตั้งค่าไว้ในสองตัวแปรคือ android:versionCode ซึ่งเป็นค่าที่ใช้ในการตรวจสอบขณะที่ทำการติดตั้งเพื่อให้ทราบว่าเป็นการติดตั้งแบบปรับปรุงรุ่นใหม่ (upgrade) หรือลดรุ่นลง (downgrade) ส่วนของตัวแปร android:versionName จะเป็นค่าระบุเวอร์ชั่นที่จะใช้แสดงในโปรแกรม

อีลีเมนต์ <application> จะระบุการตั้งค่าเกี่ยวกับฉลาก (label) และไอคอน (Icon) ของโปรแกรม

อีลีเมนต์ <activity> เป็นการกำหนดชื่อแอคทิวิตี้ที่จะสามารถใช้งานได้ภายในโปรแกรม ซึ่งจะต้องกำหนดตัว android:name เป็นชื่อของแพ็คเกจที่แอคทิวิตี้นั้นอยู่

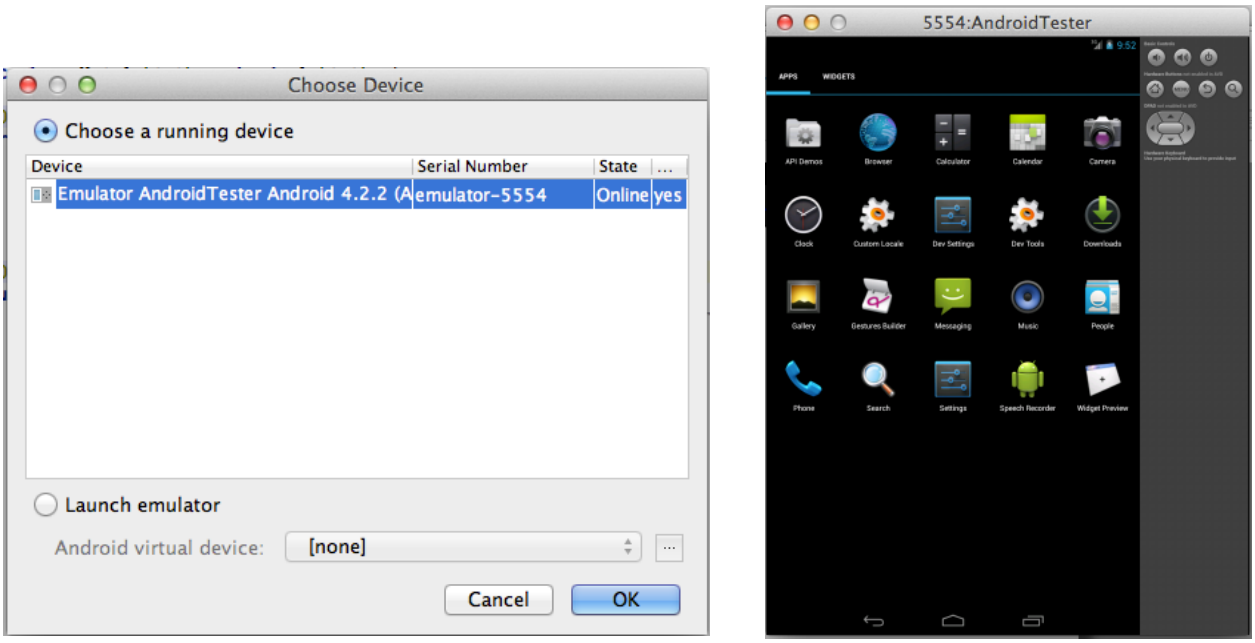
อีลีเมนต์ <intent-filter> จะเป็นการระบุให้ระบบปฏิบัติการแอนดรอยด์รู้ว่ามีการเรียกใช้งานคอมโพเน็นต์ตัวไหนบ้าง การประกาศอีลีเมนต์ <action> เป็นชื่อ android.intent.action.MAIN เป็นการบอกว่าแอคทิวิตี้นี้ทำงานเป็นแอคทิวิตี้หลักของโปรแกรม

การติดตั้งและเรียกใช้โปรแกรมบนอุปกรณ์แอนดรอยด์

เมื่อผู้เขียนโปรแกรมทำการเขียนโปรแกรมเรียบร้อยแล้วต้องการทดสอบโปรแกรมที่เขียนบนอุปกรณ์แอนดรอยด์ซึ่งจะมีวิธีการทดสอบได้ 2 ทางคือ

การรันโปรแกรมผ่าน Emulator

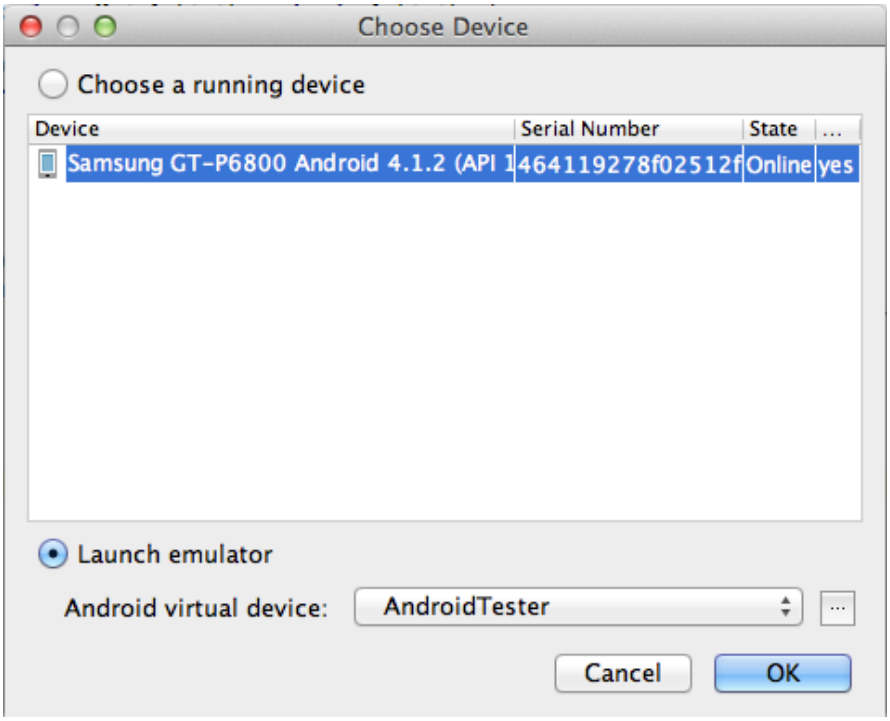
ในการทดสอบโปรแกรมผ่าน Emulator ก่อนอื่นผู้ใช้จะต้องทำการสร้าง Emulator เข้าไปอยู่ในรายการอุปกรณ์ก่อน หลังจากนั้นเมื่อผู้ใช้กดรันโปรแกรมก็จะปรากฏหน้าต่างแสดงรายชื่อของอุปกรณ์แล้วทำการเลือกอุปกรณ์ที่ต้องการรันโปรแกรมทดสอบ ดังตัวอย่างในรูปข้างล่าง



รูปที่ 6.9 แสดงการเรียกใช้ Android Emulator

การรันโปรแกรมบนอุปกรณ์จริงผ่าน ADB

ในการทดสอบโปรแกรมอีกวิธีหนึ่งคือการรันโปรแกรมผ่าน ADB เพื่อทำการติดตั้งโปรแกรมลงอุปกรณ์จริงดังรูปข้างล่าง วิธีการนี้จะทำให้ผู้เขียนโปรแกรมสามารถดูความสมบูรณ์ของโปรแกรมได้ดีกว่าการรันโปรแกรมบน Emulator



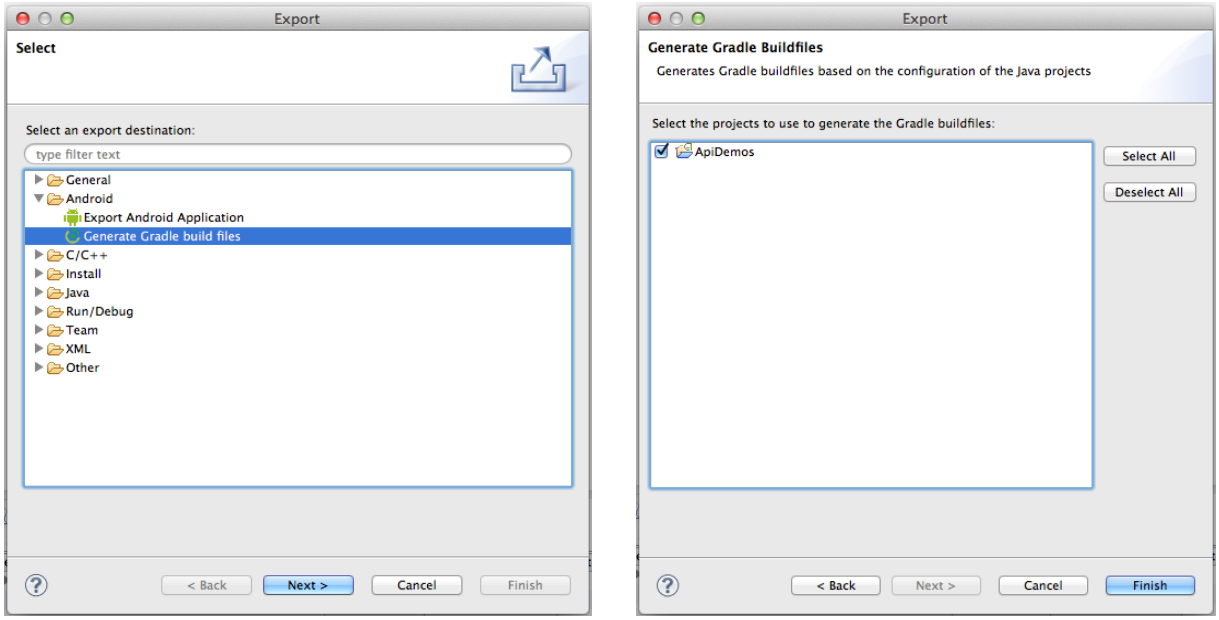
รูปที่ 6.10 แสดงอุปกรณ์ Android ที่เชื่อมต่ออยู่กับเครื่องคอมพิวเตอร์

การย้ายโค้ดโปรแกรมเดิม Eclipse IDE มาสู่ Android Studio

เนื่องจากนักพัฒนาในยุคแรกๆจะมีความคุ้นเคยกับการเขียนโปรแกรมด้วยเครื่องมือพัฒนา Eclipse IDE ดังนั้นเมื่อต้องการย้ายโค้ดโปรเจ็คเดิมที่เคยพัฒนาอยู่บน Eclipse IDE มาสู่ Android Studio IDE จะมีขั้นตอนการย้ายโค้ดโปรเจ็คดังนี้

ขั้นตอนที่ 1: Export โปรแกรมเดิมจาก Eclipse IDE

1. ทำการเปิดโปรเจคและอัพเดต ADT Plugin (เวอร์ชัน 22.0 หรือสูงกว่า)
2. เลือกเมนู File > Export
3. โปรแกรมจะแสดงหน้าต่างขึ้นมา ให้เลือก Generate Gradle build files ภายใน Android จากนั้นกดปุ่ม Finish



รูปที่ 6.11 แสดงหน้าต่างการสร้าง Gradle build files

ขั้นตอนที่ 2: Import โปรแกรมเข้าสู่ Android Studio

1. เปิดโปรแกรม Android Studio เลือกเมนู File > Import Project
2. จากนั้นเลือกไปยังที่อยู่โปรเจค เลือก build.gradle จากนั้นกดปุ่ม Finish
3. ให้เลือกตาม Dialog ที่แสดงจากนั้นกด OK

Apache Ant สำหรับการพัฒนาแอนดรอยด์

Apache Ant คือเครื่องมือสำหรับใช้ในการคอมไพล์และสร้างโปรแกรม JAVA โดยนักพัฒนาโปรแกรมสามารถสร้างสคริปท์ (Script) ในรูปแบบ XML ที่มีการระบุ packages ที่เกี่ยวข้องเพื่อสั่งให้โปรแกรม ant ทำการคอมไพล์และสร้างออกมาเป็นโปรแกรม .APK (Android Package) ได้เหมือนกับการสร้างไฟล์ Makefile และใช้คำสั่ง “make” ในระบบปฏิบัติการลีนุกซ์

เมื่อได้ทำการติดตั้งโปรแกรม Ant (http://ant.apache.org) สามารถทดสอบเขียนโปรแกรมพื้นฐาน (src\ant\Test.java) ดังตัวอย่างข้างล่าง

```
package ant;
class Test {
     public static void main(String args[])
     {
         System.out.println("Hello World from ANT");
     }
}
```

สคริปท์ไฟล์ src\ant\build.xml

```
<project basedir="." default="make">
<property name="root" value="../.." />
 <property name="classes" value="${root}/classes" />
 <property name="srcroot" value="${root}/src" />
 <path id="project.class.path">
  <pathelement location="${classes}" />
    </path>
 <target name="make">
  <javac debug="true" srcdir="${srcroot}" includes="ant/**/*.java"
destdir="${classes}">
  </javac>
 </target>
 <target name="run" depends="make" description="Run Test">
  <java classname="ant.Test" classpathref="project.class.path"  fork="true">
  </java>
 </target>
  <target name="clean" description="clean all classes">
  <delete dir="${classes}/ant"/>
  </target>
</project>
<?xml version="1.0" encoding="UTF-8"?>
```

คอมไพล์และสร้างโปรแกรมออกมาเป็น .APK

```
C:\...\src\ant>ant make
Buildfile: build.xml
make:
```

```
BUILD SUCCESSFUL
Total time: 0 seconds

C:\...\src\ant>ant run
Buildfile: build.xml
make:
run:
     [java] test ANT
BUILD SUCCESSFUL
Total time: 0 seconds

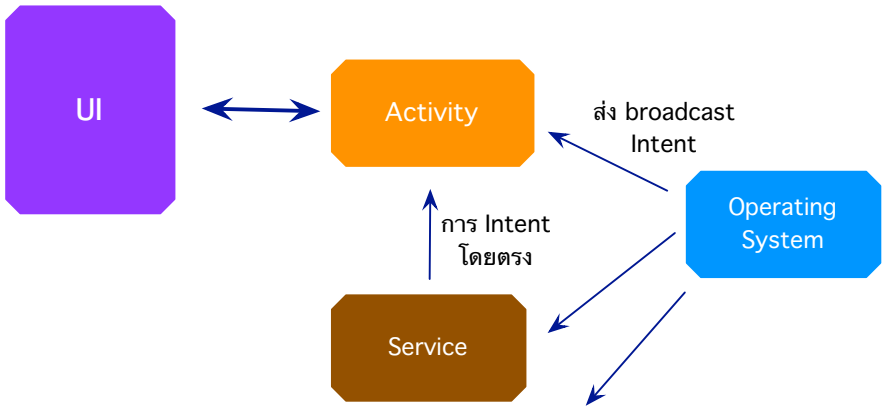
C:\...\src\ant>ant clean
Buildfile: build.xml
clean:
   [delete] Deleting directory C:\...\classes\ant
BUILD SUCCESSFUL
Total time: 0 seconds
```

ไฟล์โปรแกรม (.APK) จะถูกสร้างเก็บไว้ในไดเรกทอรี bin ผู้ใช้สามารถนำไฟล์ .APK ไปติดตั้งลงใน Android Emulator หรืออุปกรณ์แอนดรอยด์ โดยใช้คำสั่ง adb install <ชื่อโปรแกรม>.apk

ANDROID ACTIVITY

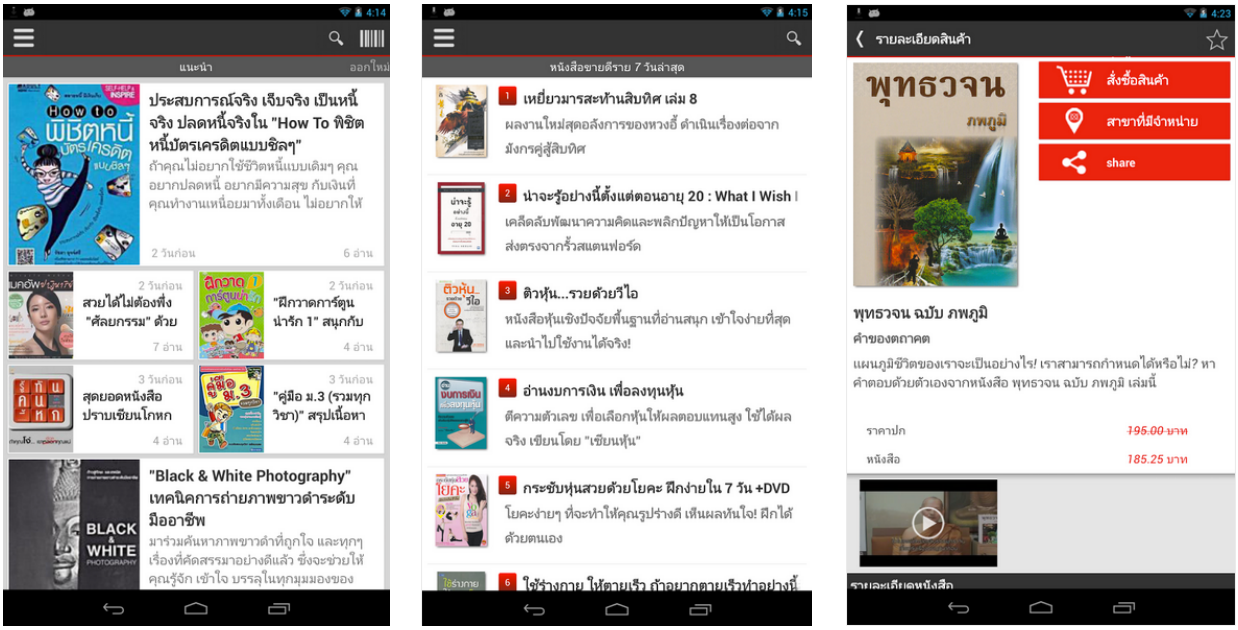
การพัฒนาโปรแกรมประยุกต์แอนดรอยด์ : แอคทิวิตี้ (Activity)

แอคทิวิตี้ (Activity) มีความสำคัญมากสำหรับโปรแกรมประยุกต์ เพราะเป็นส่วนที่ใช้ติดต่อกับผู้ใช้ผ่าน UI ดังแสดงในรูปข้างล่าง



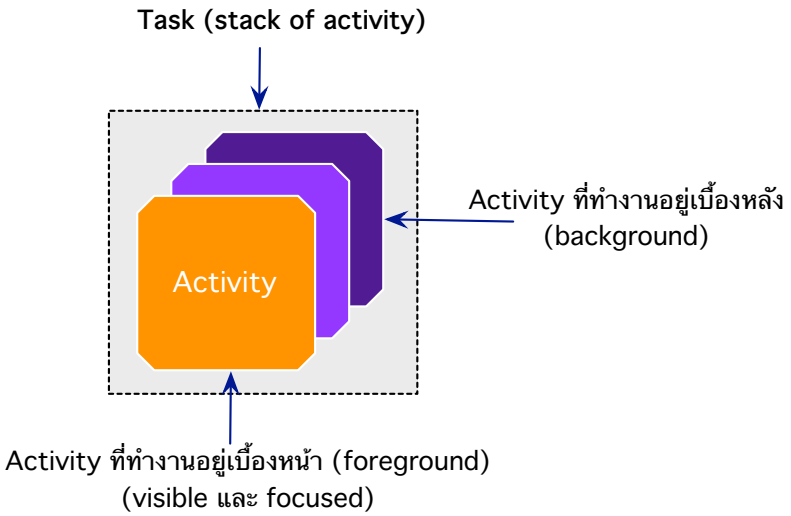
รูปที่ 6.12 แสดงกลไกการทำงานของโปรแกรมภายในแอนดรอยด์

แต่ละ activity จะแทนหนึ่งหน้าต่างที่จะแสดงขึ้นบนหน้าจอในขณะนั้นดังแสดงในรูปข้างล่าง



รูปที่ 6.13 การแสดงผลของแต่ละ activity

ซึ่งในแต่ละโปรแกรมประยุกต์สามารถสร้างแอคทิวิตี้ได้หลายตัวแต่จะต้องกำหนดแอคทิวิตี้เพียงตัวใดตัวหนึ่งเท่านั้นที่จะให้ทำงานเป็นแอคทิวิตี้หลักของโปรแกรมประยุกต์นั้น ซึ่งแต่ละแอคทิวิตี้สามารถที่จะเรียกให้ตัวแอคทิวิตี้อื่นขึ้นมาทำงานได้โดยที่แอคทิวิตี้ก่อนหน้าจะถูกเก็บไว้ในสแต็ก (Stack) ของระบบตามลำดับไปดังรูปข้างล่าง



รูปที่ 6.14 ขบวนการจัดการของ activities แต่ละตัวในขณะที่ทำงานอยู่

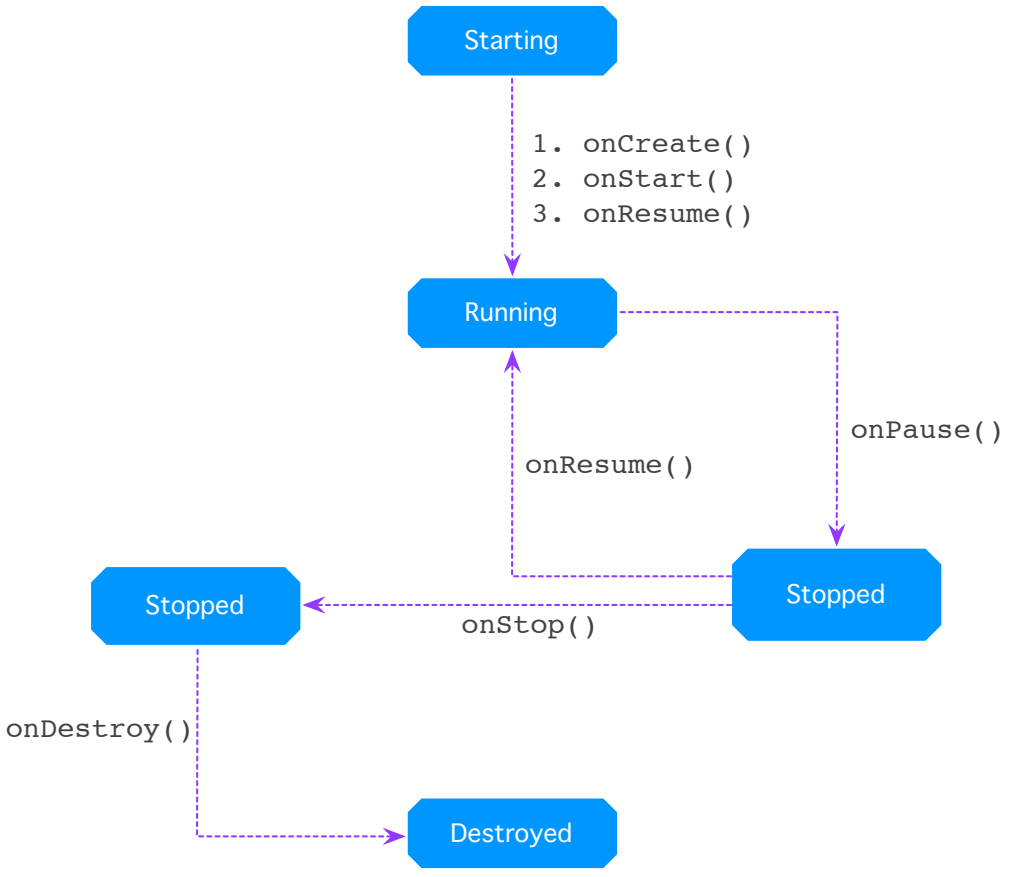
ภายในโปรแกรมประยุกต์จะมี 4 องค์ประกอบหลักได้แก่

1) Activity
2) Service
3) Intent
4) Broadcast Receiver

ตาราง 6.1 แสดงลักษณะการทำงานของ 4 องค์ประกอบหลักในโปรแกรม Android

ประเภท	ลักษณะการทำงาน	ตัวอย่างการทำงาน
Activity	โปรเซสที่ทำงานร่วมกับ User Interface	แสดงข้อความ, รูปภาพ, เล่นเกมส์ ฯลฯ
Service	โปรแกรมที่ทำงานอยู่เบื้องหลัง	เล่นเพลง, ระบบซิงค์ข้อมูล
Intent	Intent ชนิด Directed Intent	การแจ้งไปยังโปรเซสที่ระบุไว้
	Intent ชนิด Broadcast Intent	การแจ้งแบบกระจายไปยังทุกโปรเซส
	Intent Filter	รายการของ Intents ใน Activity หรือ Services
Broadcast Receiver	การขอเรียกใช้ข้อมูล	GPS, Internet, สมุดโทรศัพท์ ฯลฯ

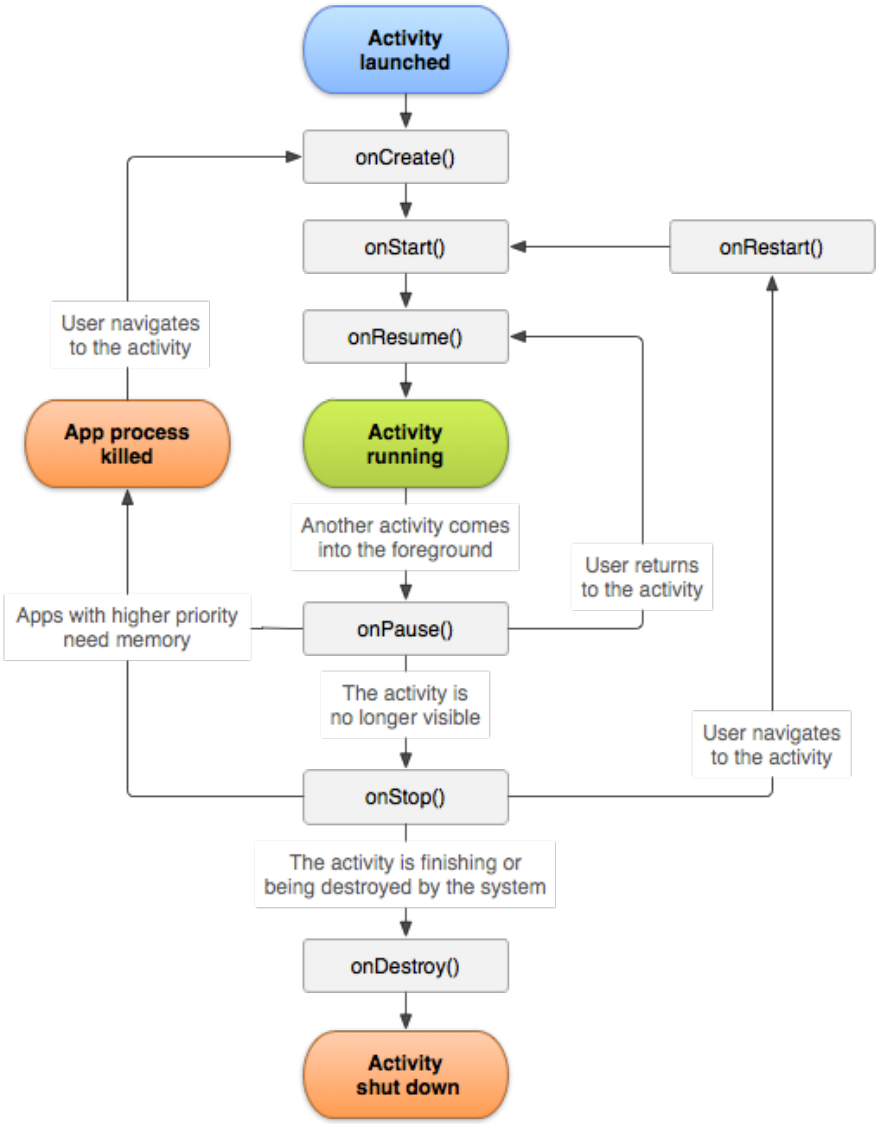
วงจรชีวิต (lifecycle) ของแอคทิวิตี้



รูปที่ 6.15 วงจรชีวิตของแอคทิวิตี้ (Activity)

ช่วงวงจรชีวิตของแอคทิวิตี้นั้นสามารถแบ่งออกได้เป็น 3 ช่วงการทำงาน ดังรูปข้างบน คือ

1. Entire Lifetime คือการเริ่มต้นและการปิดตัวของแอคทิวิตี้ ในช่วงการทำงานนี้จะมีการเรียกใช้ฟังก์ชันคือ onCreate() และเมื่อแอคทิวิตี้ถูกสั่งปิดจะมีการเรียกใช้ฟังก์ชันคือ onDestroy()
2. Visible Lifetime คือช่วงเวลาที่แอคทิวิตี้นั้น กำลังทำงานอยู่โดยหลังจากที่แอคทิวิตี้ถูกสร้างขึ้นมาแล้ว แอคทิวิตี้นั้น กำลังจะเริ่มต้นการทำงานจะมีการเรียกใช้ฟังก์ชัน onStart() และเมื่อแอคทิวิตี้นั้นถูกสั่งปิดจะต้องมีการปิดการทำงานแอคทิวิตี้ก่อนจะมีการเรียกใช้ฟังก์ชัน onStop()
3. Foreground Lifetime คือช่วงเวลาที่แอคทิวิตี้นั้น กำลังทำงานอยู่เบื้องหน้าเช่น เมื่อแอคทิวิตี้นั้นเริ่มต้นการทำงานจะถูกดันขึ้นมาด้านบนเพื่อติดต่อกับผู้ใช้งานโดยจะเรียกฟังก์ชัน onResume() และถ้าแอคทิวิตี้นั้นกำลังจะหยุดการทำงานชั่วคราวหรือถาวร ก็จะเรียกใช้ฟังก์ชัน onPause() หรือ onDestroy()



รูปที่ 6.16 รายละเอียดของวงจรชีวิตของแอคทิวิตี้

ตัวอย่างโค้ดการทำงานของแอคทิวิตี้

จากโค้ดตัวอย่างดังต่อไปนี้จะเป็นการอธิบายหลักการทำงานของแอคทิวิตี้ให้เข้าใจได้ง่ายขึ้น โดยจะมีการทำงานตาม Lifecycle ที่ได้อธิบายแล้วข้างต้น

```
public class ActivityA extends Activity {

      @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_a);
     Log.d("Embedded Debug”, onCreate - Called when the Activity is first created”);
    }

      @Override
    protected void onStart() {
         Log.d("Embedded Debug”, onStart - Called when the Activity is becoming
visible to the User”);
          super.onStart();
    }

      @Override
    protected void onRestart() {
        Log.d("Embedded Debug”, onRestart - Called after your activity has been
stopped, prior to it being started again”);
        super.onRestart();
    }

      @Override
    protected void onResume() {
        Log.d("Embedded Debug”, onResume - Called when the Activity will start in-
teracting with the User”);
        super.onResume();
    }

      @Override
    protected void onPause() {
        Log.d("Embedded Debug”, onPause - Called when the system is about to start
resuming a previous activity”);
        super.onPause();
    }

      @Override
    protected void onStop() {
        Log.d("Embedded Debug”, onStop - Called when the Activity is no longer visi-
ble to the User because another activity has been resumed and is covering this
one”);
        super.onStop();
    }

      @Override
    protected void onDestroy() {
```

```
        Log.d("Embedded Debug”, onDestroy - the final call you receive before your
activity is destroyed”);
        super.onDestroy();
    }

      public void startActivityB(View v) {
            Intent intent = new Intent(ActivityA.this, ActivityB.class);
            startActivity(intent);
      }

      public void startActivityC(View v) {
            Intent intent = new Intent(ActivityA.this, ActivityC.class);
            startActivity(intent);
      }

      public void finishActivityA(View v) {
            ActivityA.this.finish();
      }

}
```

ไฟล์ AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/dark_blue"
    android:padding="8dip"
    >

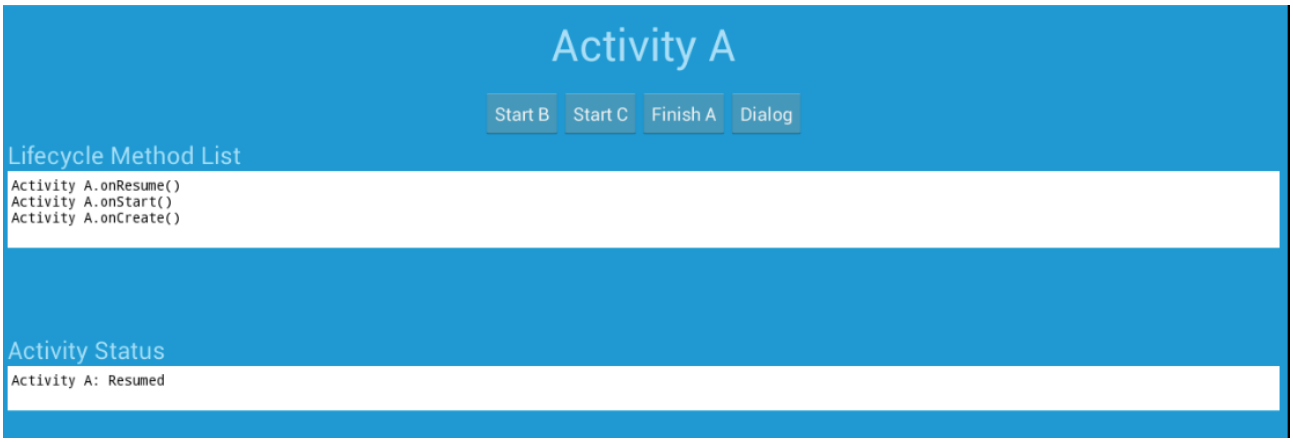
    <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        >
        <Button
            android:id="@+id/btn_start_b"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="@string/btn_start_b_label"
            android:onClick="startActivityB"
            />

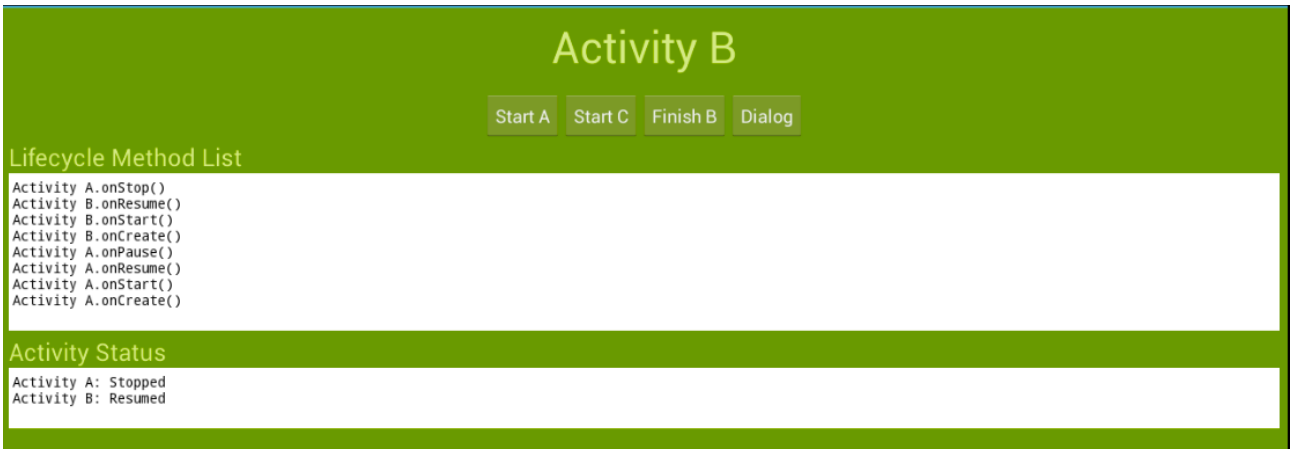
        <Button
            android:id="@+id/btn_start_c"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="@string/btn_start_c_label"
```

```
            android:layout_toRightOf="@id/btn_start_b"
            android:onClick="startActivityC"
            />
        <Button
            android:id="@+id/btn_finish_a"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="@string/btn_finish_a_label"
            android:layout_toRightOf="@id/btn_start_c"
            android:onClick="finishActivityA"
            />
        <Button
            android:id="@+id/btn_start_dialog"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="@string/btn_start_dialog_label"
            android:layout_toRightOf="@id/btn_finish_a"
            android:onClick="startDialog"
            />
    </RelativeLayout>
</LinearLayout>
```

จากตัวอย่างโค้ดโปรแกรมข้างต้น เป็นการแสดงตัวอย่างการเรียกใช้งานแอคทิวิตี้ โดยจะแสดงให้เห็นถึงวงจรชีวิตของแอคทิวิตี้ จากโค้ดโปรแกรมได้แสดงตัวอย่างของแอคทิวิตี้ A เมื่อรันโปรแกรม โปรแกรมจะทำการเริ่มต้นด้วยกันเรียกใช้งานแอคทิวิตี้ A ขึ้นมา จากนั้นจะทำการแสดง layout ที่ได้ออกแบบไว้ซึ่งภายใน layout ได้มีการตั้งค่าให้เรียกใช้งานฟังก์ชันต่างๆ เช่นการเรียกแอคทิวิตี้ B เรียกแอคทิวิตี้ C เป็นต้น ดังตัวอย่างโค้ดและผลการทำการของโปรแกรมในรูปข้างล่าง

```
android:onClick="startActivityB"
```





รูปที่ 6.17 ผลลัพธ์การรันโปรแกรม

USER INTERFACE

เมื่อเข้าใจหลักการทำการของแอคทิวิตี้แล้ว ในส่วนการติดต่อระหว่างแอคทิวิตี้กับผู้ใช้ก็มีส่วนสำคัญไม่น้อยเนื่องจากเป็นกราฟฟิกสำหรับใช้ในการติดต่อกับผู้ใช้ทั่วไป ซึ่งจะเรียกกันโดยทั่วไปว่า User Interface (UI) การสร้างหน้าตาของโปรแกรมหรือ UI ขึ้นมาได้นั้นมีเครื่องมืออยู่ต่างๆมากมาย แต่อย่างไรก็ตามการออกแบบและพัฒนาโปรแกรมเพื่อตอบสนองความต้องการของผู้ใช้งานให้ได้ดีที่สุดนั้นก็ยังต้องให้ความสนใจในการสร้างประสบการณ์การใช้งานโปรแกรมที่เรียกว่า User Experience (UX) ควบคู่ไปด้วย ดังนั้นนักพัฒนาจะต้องมีการศึกษาวิเคราะห์พฤติกรรมการใช้งานกับอุปกรณ์ไฮเทคโนโลยีเหล่านี้ไม่ว่าจะเป็นเทคโนโลยีหน้าจอสัมผัสและแสดงผล หน่วยประมวลผลกลาง หน่วยความจำ และการพัฒนาปรับปรุงระบบปฏิบัติการทางด้านระบบสมองกลฝังตัวควบคู่กันไปด้วยตัวอย่างเช่นการทำความเข้าใจหน่วยค่าการแสดงผลกราฟฟิกที่จะถูกขึ้นอยู่กับเทคโนโลยีหน้าจอสัมผัสและแสดงผล การออกแบบโปรแกรมทางด้านกราฟฟิกเพื่อรับค่าจากผู้ใช้โดยต้องให้เหมาะสมกับขนาดของหน้าจออุปกรณ์ที่ใช้งานโดยจะกล่าวถึงในรายละเอียดต่อไปนี้

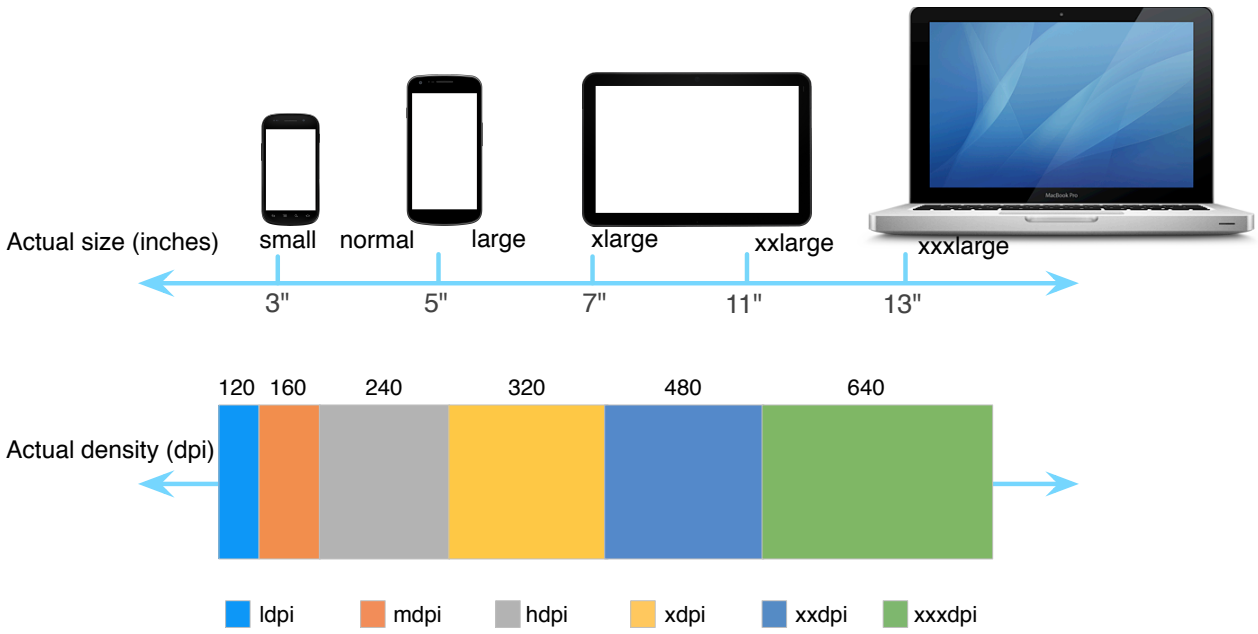
หน่วยค่าการแสดงผลกราฟฟิก

การทำความเข้าใจหลักการทางด้านการออกแบบกราฟฟิก นักพัฒนาส่วนใหญ่จะค่อนข้างมีปัญหาในเรื่องนี้พอสมควรเนื่องจากมีรายละเอียดอยู่ไม่น้อย ปัจจุบันอุปกรณ์ที่ใช้ระบบปฏิบัติการแอนดรอยด์นั้นมีอยู่มากมายหลากหลายรุ่นและมีความหลากหลายของสัดส่วนหน้าจอแสดงผล เช่นขนาดหน้าจอ 480x320, 800x480, 960x540, 1280x720 ซึ่งจะมีผลต่อการออกแบบหน้าตาของโปรแกรม ดังนั้นการกำหนดตำแหน่งการวางกราฟฟิกหรือองค์ประกอบต่างๆจะไม่สามารถกำหนดจุดสี (Pixel) แบบตายตัวได้เพราะหากมีการกำหนดค่าจุดสีตายตัวเมื่อเปิดโปรแกรมบนเครื่องหน้าจอที่มีความละเอียดแตกต่างกันก็จะทำให้การแสดงผลกราฟฟิกผิดเพี้ยนไปจากเดิมที่กำหนดไว้ได้

ดังนั้นทางผู้พัฒนาของบริษัทกูเกิลจึงได้คิดค้นหน่วยจุดสีใหม่ขึ้นมาเพื่อให้ความละเอียดหน้าจอที่แตกต่างกันเกิดความคลาดเคลื่อนน้อยที่สุดเรียกหน่วยนี้ว่าหน่วย “dp” (Density Independent Pixel) หรือจุดสีเสมือน (Virtual Pixel) ซึ่งเป็นหน่วยของความละเอียดหน้าจอที่สมมติขึ้นมาโดยอิงจากความละเอียด (Resolution), ขนาด (Size) และความหนาแน่นจุดสี (Pixel) (Density) ของหน้าจอบนอุปกรณ์นั้นซึ่งนักพัฒนาจะต้องใช้ค่า dp ในการกำหนดขนาดของข้อความ (Text) ขนาดของรูปภาพหรือขนาดของปุ่มต่างๆ โดยมีหลักการคำนวณค่า dp ดังนี้

1 dp = (ความหนาแน่นของจอ) / 160

แอนดรอยด์ได้แบ่งหน้าจอเป็นหลายขนาดโดยมีค่าความละเอียดของแต่ละหน้าจอ ดังแสดงในรูปข้างล่าง



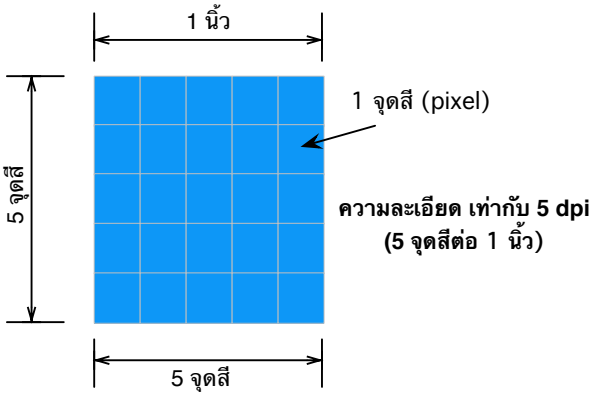
รูปที่ 6.18 เปรียบเทียบรายละเอียดของขนาดหน้าจอและค่า dpi

ล่าสุดบนระบบปฏิบัติการแอนดรอยด์ เวอร์ชัน 4.3 ได้ประกาศออกมาอย่างเป็นทางการแล้วว่าพร้อมจะรองรับค่าความละเอียดหน้าจอระดับ extra extra high dpi (xxhdpi) ขนาด 640 dpi ดังรูปข้างบน และในอนาคตอันใกล้นี้ค่าความละเอียดของหน้าจอบนอุปกรณ์แอนดรอยด์ก็จะเทียบเท่ากับค่าความละเอียดหน้าจอโทรทัศน์ที่ 4K ได้อย่างแน่นอน

- ldpi (low dpi) คือ หน้าจอที่มีขนาดของความหนาแน่น ไม่เกิน 120 dpi
- mdpi (medium dpi) คือ หน้าจอที่มีความหนาแน่น ตั้งแต่ 120 - 160 dpi เช่นรุ่น T-Mobile G1
- hdpi (high dpi) คือ หน้าจอที่มีความหนาแน่น ตั้งแต่ 160 - 240 dpi เช่นรุ่น Nexus S
- xdpi (extra high dpi) คือ หน้าจอที่มีความหนาแน่น 240 dpi ขึ้นไป (320 dpi บนรุ่น Galaxy Nexus/N4 หรือ 480 dpi บนรุ่น HTC One)

- xxdpi (extra extra high dpi) คือ หน้าจอที่มีความหนาแน่น 480 dpi ขึ้นไป
- xxxdpi (extra extra extra high dpi) คือ หน้าจอที่มีความหนาแน่น 640 dpi ขึ้นไป

ซึ่งค่า dpi (Dot Per Inch) หมายถึง จำนวนจุดสีที่มีได้ทั้งหมดในพื้นที่ขนาด 1 ตารางนิ้ว ดังแสดงในรูปด้านขวา ยกตัวอย่างเช่น จำนวนจุดในพื้นที่ขนาด 1 ตารางนิ้ว จะมีจุดสีเรียงกันอยู่จำนวนทั้งสิ้น 300 x 300 Pixel ซึ่งค่า dpi ยิ่งสูงก็จะมีผลต่อการแสดงผลโดยเฉพาะกับการใช้ในโปรแกรมเกมส์ในลักษณะ FPS หรือเกมส์ลักษณะ Shooting ที่ต้องการใช้การแสดงผลกราฟฟิกที่เปลี่ยนแปลงอย่างรวดเร็ว เป็นต้น



รูปที่ 6.19 แสดงการคิดค่าความละเอียด (dpi) ของหน้าจอ

สูตรการคำนวณค่า dp จากค่า dpi คือ

```
dp = pixel * (160 / dpi)
```

ตัวอย่างการคำนวณค่า dp ของเครื่องยี่ห้อ Samsung Galaxy Note 10.1 ที่มีขนาดหน้าจอแบบ mdpi ซึ่งเทียบค่า dpi เท่ากับ 160 (ตามรูปที่ 6.8) และเครื่องนี้มีค่าความละเอียดหน้าจออยู่ที่ 1280 x 800 Pixel ค่า dp จะมีค่าเท่ากับ

```
1280 * (160 / 160) =  1280 dp
 800 * (160 / 160) =   800 dp
```

ดังนั้นเครื่อง Samsung Galaxy Note 10.1 นี้จะมีค่าขนาดหน้าจอแบบ dp เท่ากับ **1280 x 800 dp**

ซึ่งเมื่อนำเครื่อง Samsung Nexus 10 ที่มีขนาดหน้าจอแบบ xhdpi ที่ค่า dpi เท่ากับ 320 dpi (ตามรูปที่ 6.8) และความละเอียดหน้าจอสูงถึง 2560 x 1600 Pixel ซึ่งจะเห็นว่ามีค่าสูงกว่าเครื่อง Samsung Galaxy Note 10.1 แต่เมื่อคำนวณค่า dp จะมีค่าเท่ากับ

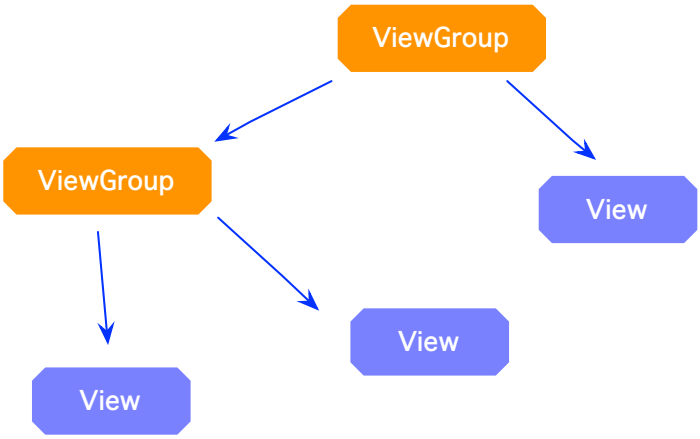
```
2560 * (160 / 320) = 1280 dp
1600 * (160 / 320) =  800 dp
```

จะเห็นได้ว่าค่า dp ของทั้ง 2 เครื่องมีค่า **1280 x 800 dp** เท่ากัน ดังนั้นนักพัฒนาก็ควรยึดค่า dp เป็นหลักในการวางรูปแบบกราฟฟิกที่จะเป็นหน้าตาโปรแกรม โดยที่ยังคงแสดงในสัดส่วนที่ถูกต้องเช่นเดิม

อ้างอิงส่วนหนึ่งมาจาก: http://www.akexorcist.com/2013/03/android-design-dp.html

View และ ViewGroup

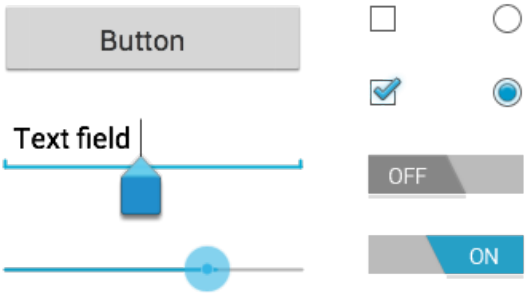
View คือส่วนติดต่อกับผู้ใช้พื้นฐานเนื่องจากเป็นองค์ประกอบพื้นฐานการของสร้าง UI ต่างๆภายในโปรแกรมตัวอย่างเช่น TextView, ImageView, SurfaceView, ViewGroup เป็นต้น สำหรับคลาส ViewGroup เป็นส่วนขยายของ View ที่สามารถมี View ได้หลายตัวโดยสามารถสร้างการควบคุมเชื่อมต่อไปยัง View ทั้งหมดได้



รูปที่ 6.20 แสดงความสัมพันธ์ของ ViewGroup และ View

Input Controls

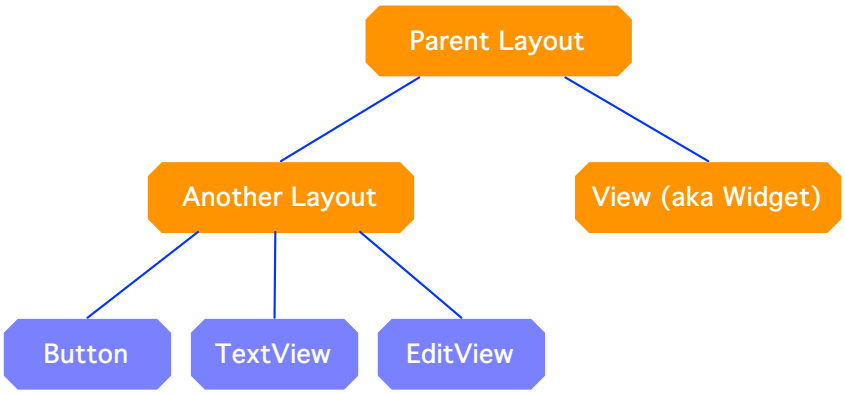
คือส่วนวัตถุ (Object) ที่ใช้ติดต่อกับผู้ใช้ ซึ่งสามารถโต้ตอบกับผู้ใช้ได้ ยกตัวอย่างเช่น ปุ่ม (Button), ช่องใส่ข้อความ (text fields), บาร์ตั้งค่า (seek bars), ปุ่มเลือก (checkboxes), ปุ่มขยาย (zoom buttons), ปุ่มกลับค่า (toggle buttons) เป็นต้น



รูปที่ 6.21 ตัวอย่าง Input Controls

Layout Containers

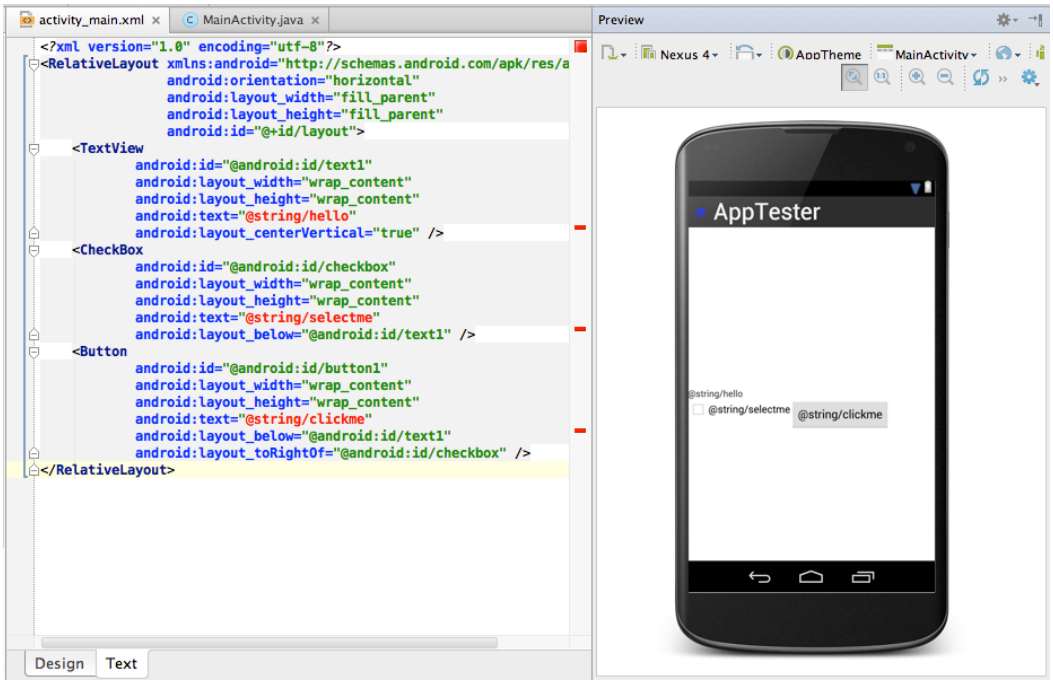
ถ้าเปรียบเทียบกับการเขียนโปรแกรมด้วย JAVA AWT หรือ JAVA Swing ตัว layout ก็เปรียบเสมือน JAVA Containers ส่วน View ก็จะเปรียบเสมือน JAVA Components หรือ Widgets นั่นเอง โดยความสัมพันธ์ระหว่าง layout และ view แสดงดังรูปข้างล่าง



รูปที่ 6.22 แสดงความสัมพันธ์ของ Layout และ components ต่างๆ

โดย layout แต่ละตัวจะสามารถบรรจุ View และ layout อื่นๆเข้าไปได้หลายๆตัว ได้แก่

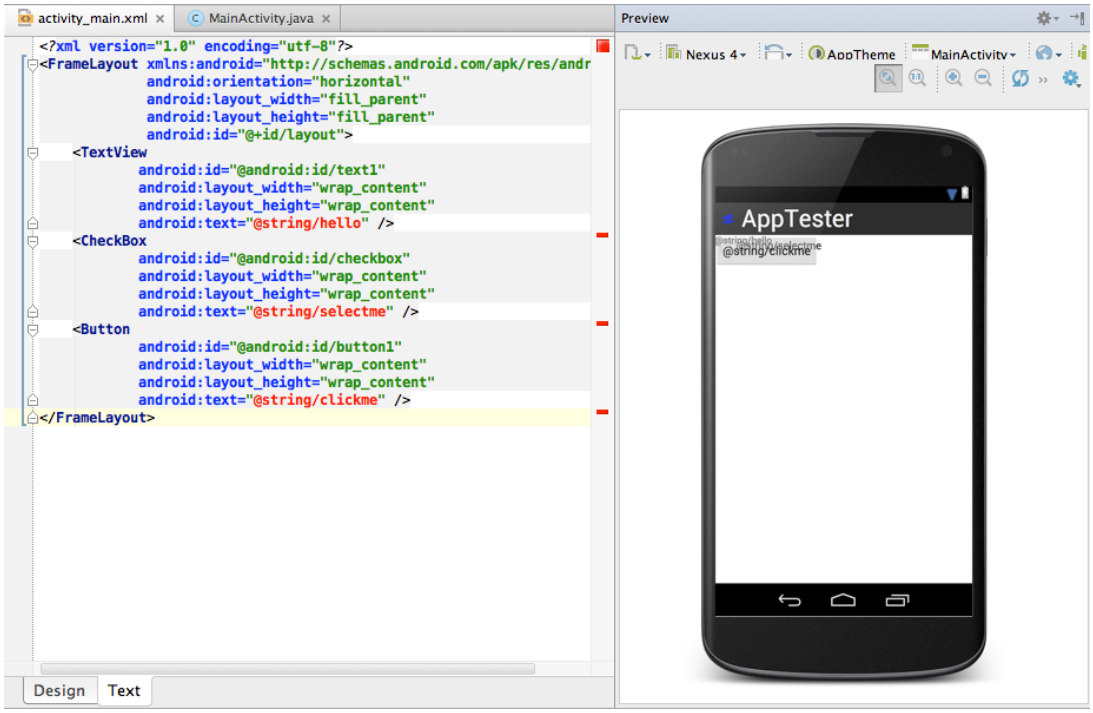
1. RelativeLayout ซึ่งเป็น layout ที่กำหนดไว้เป็นตัวพื้นฐานตั้งแต่เริ่มสร้างโปรเจค ซึ่งเป็น layout ที่มีการจัดเรียงตัววัตถุ (object) โดยมีการอ้างอิงตำแหน่งของวัตถุนั้นกับวัตถุอื่นภายใน layout หรืออ้างอิงกับตัว layout ที่วัตถุนั้นอยู่ภายใน โดยการอ้างอิงผ่านหมายเลข (android:id) ของวัตถุหรือ layout




```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/a
                android:orientation="horizontal"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:id="@+id/layout">
    <TextView
            android:id="@android:id/text1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/hello"
            android:layout_centerVertical="true" />
    <CheckBox
            android:id="@android:id/checkbox"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/selectme"
            android:layout_below="@android:id/text1" />
    <Button
            android:id="@android:id/button1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/clickme"
            android:layout_below="@android:id/text1"
            android:layout_toRightOf="@android:id/checkbox" />
</RelativeLayout>
```

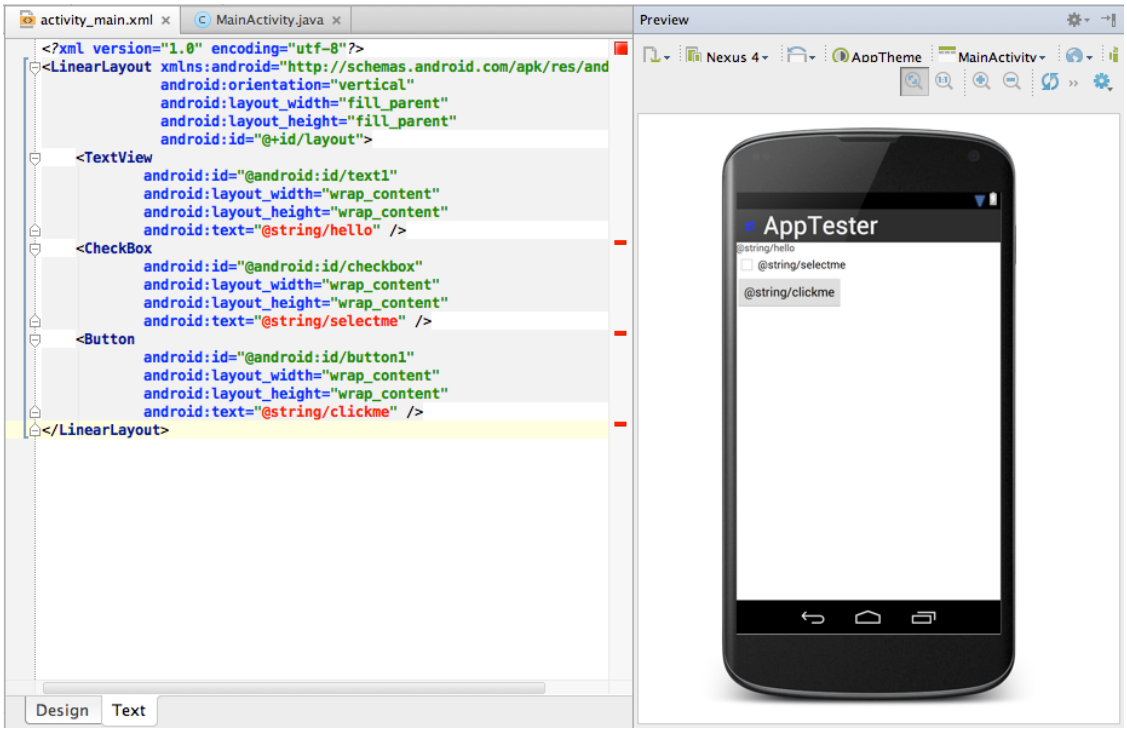
รูปที่ 6.23 การตั้งค่าแบบ RelativeLayout

2. FrameLayout เป็น layout ที่มีการจัดวางวัตถุเป็นชั้นๆ โดยจะเริ่มวางวัตถุเริ่มต้นจากมุมซ้ายบน (top-left) ของ layout เสมอและจะนำวัตถุที่วางทีหลังมาไว้ชั้นบนสุดของ layout

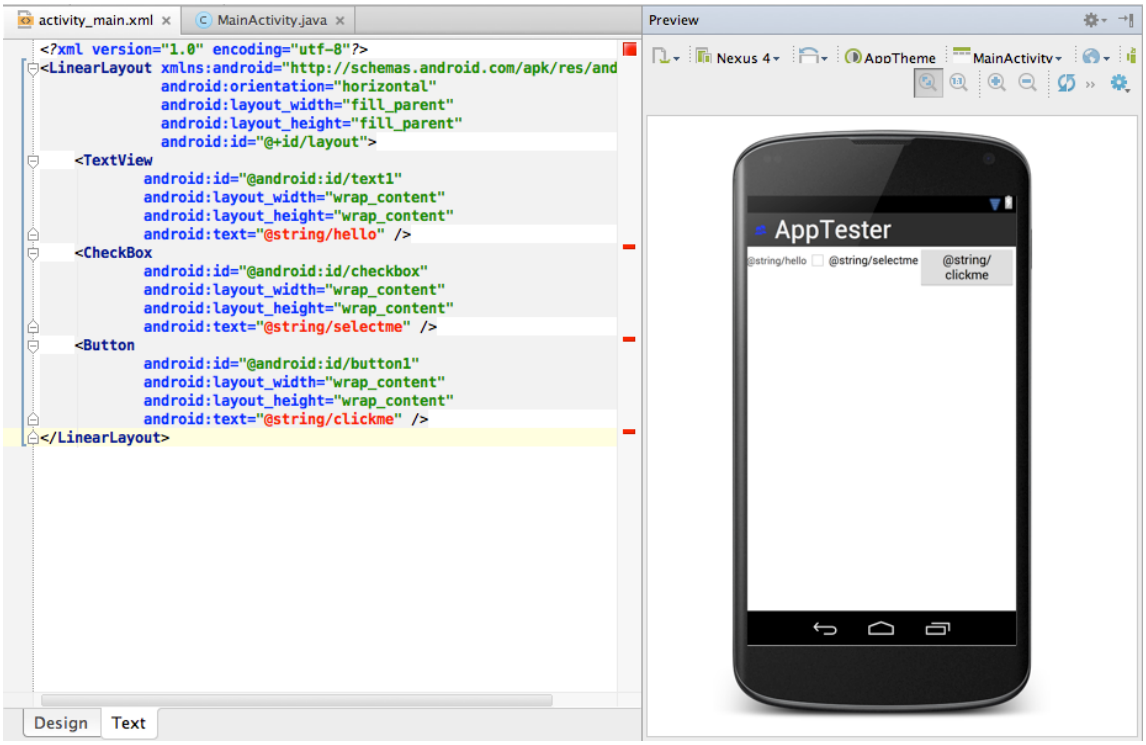


รูปที่ 6.24 การตั้งค่าแบบ FrameLayout

3. LinearLayout เป็น layout ที่มีการจัดเรียงวัตถุเป็นแนวเส้นตรง ซึ่งอาจจะเป็นแนวตั้งหรือแนวนอนก็ได้ โดยการกำหนดค่าใน android:orientation (horizontal/vertical)

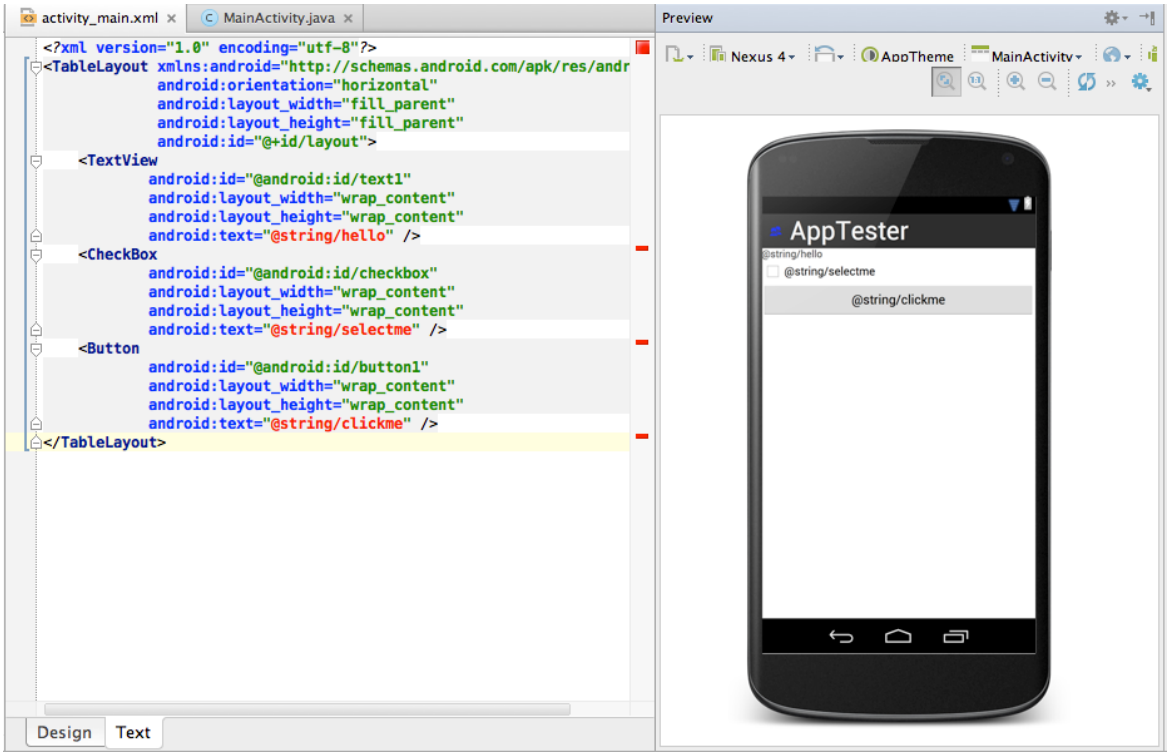


รูปที่ 6.25 การตั้งค่าแบบ LinearLayout แนวตั้ง

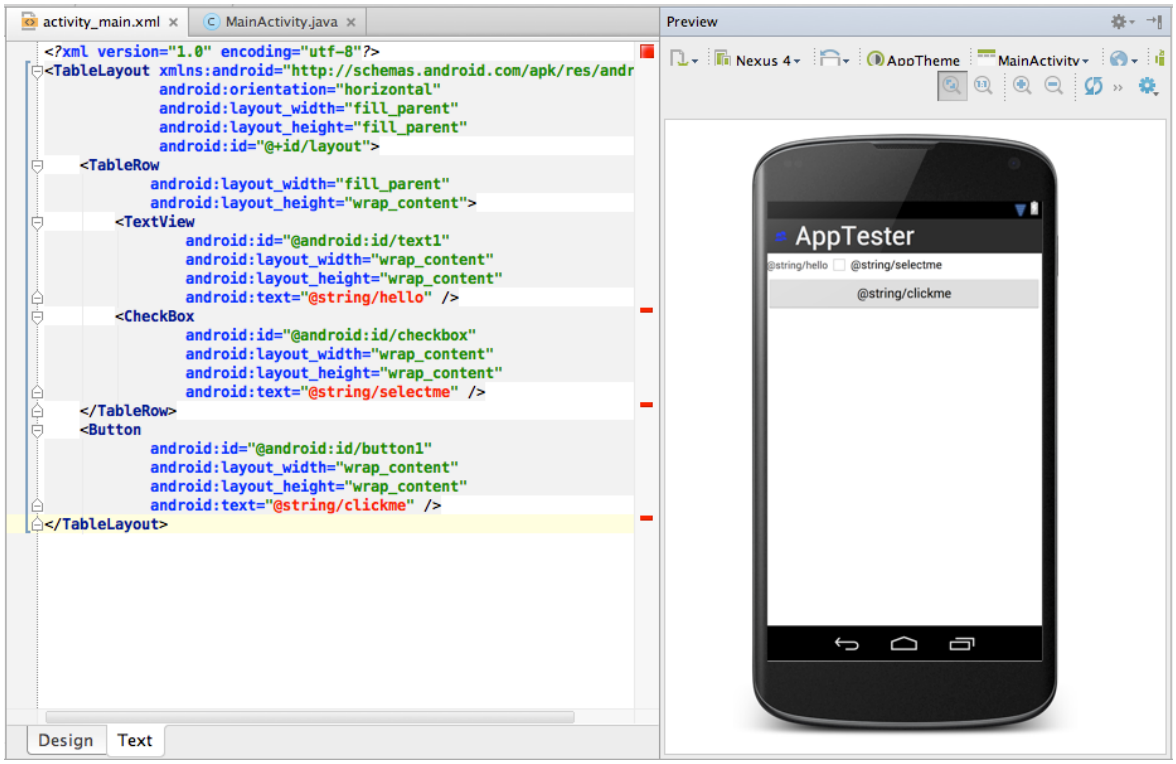


รูปที่ 6.26 การตั้งค่าแบบ LinearLayout แนวนอน

4. TableLayout เป็น layout ที่มีการจัดวางวัตถุในแบบตาราง โดยวัตถุแต่ละตัวจะถือเป็น 1 คอลัมน์ ซึ่งสามารถเพิ่มแถวได้โดยเพิ่ม XML tag ของ TableRow

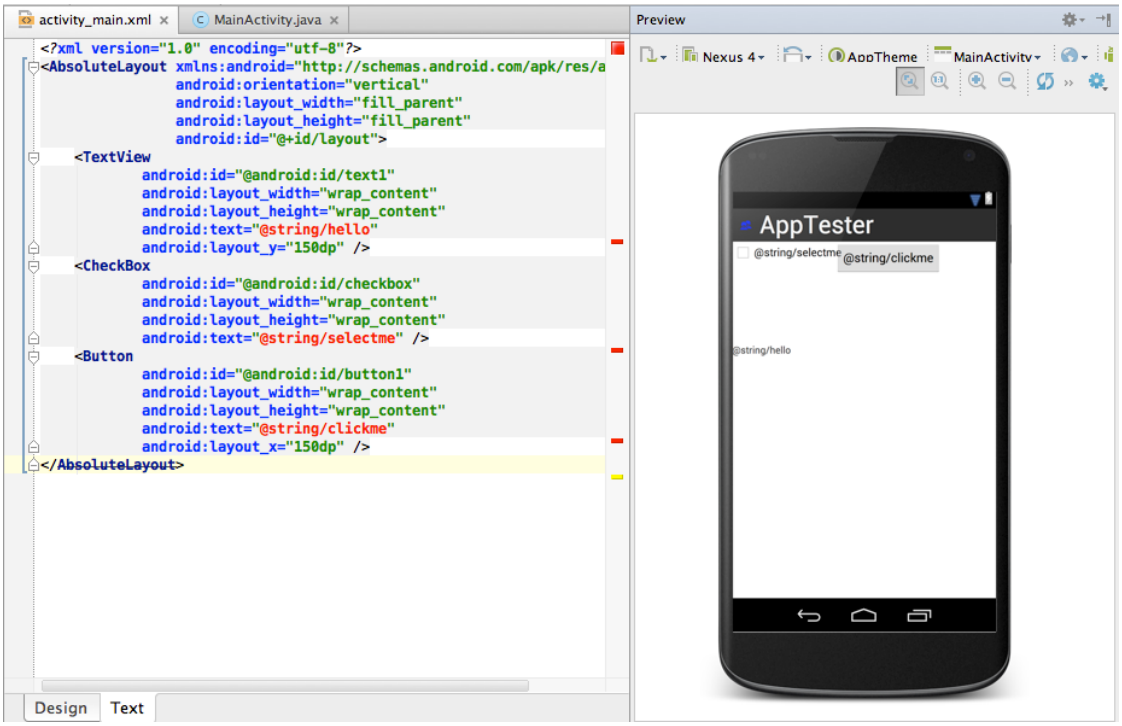


รูปที่ 6.27 การตั้งค่าแบบ TableLayout ไม่มีแถว



รูปที่ 6.28 การตั้งค่าแบบ TableLayout มีแถว

5. AbsoluteLayout เป็น layout ที่มีการจัดวางวัตถุตามตำแหน่งจริงบนหน้าจอ โดยจะกำหนดตำแหน่งของวัตถุได้ผ่านค่า android:layout_x และ android:layout_y



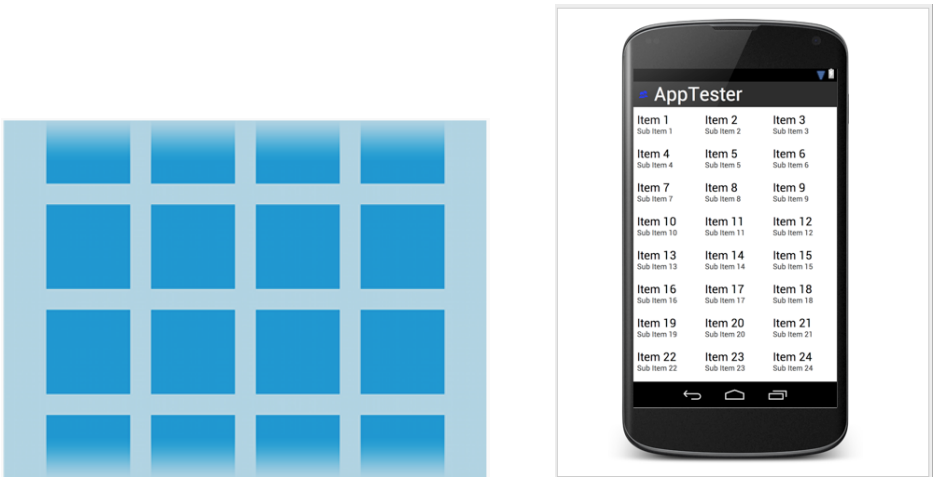
รูปที่ 6.29 การตั้งค่าแบบ AbsoluteLayout

6. ListView เป็นคลาสที่สืบทอดจากคลาส ViewGroup ที่มีมุมมองการแสดงแบบรายการ (list) ของไอเท็ม (item) โดยที่สามารถเลื่อนขึ้นลง (Scrolling) เพื่อดูเนื้อหาทั้งหมดได้ ซึ่งเนื้อหาของรายการแต่ละรายการจะถูกแทรกลงอัตโนมัติผ่านทางอะแดปเตอร์ (Adapter)



รูปที่ 6.30 การตั้งค่า ListView

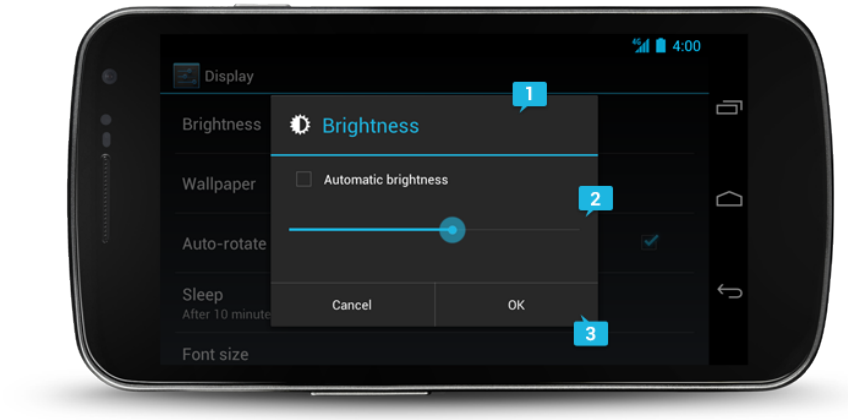
7. GridView เป็นอีกคลาสที่สืบทอดจากคลาส ViewGroup ที่มีมุมมองการแสดงแบบสองมิติ (แถว-คอลัมน์) ในลักษณะตาราง สามารถทำการเลื่อนขึ้นลงเพื่อดูข้อมูลทั้งหมดได้ ซึ่งรายการจะถูกแทรกลงอัตโนมัติผ่านทางอะแดปเตอร์เช่นเดียวกันแบบ ListView



รูปที่ 6.31 การตั้งค่า GridView

หน้าต่าง Dialog และ Alert

คือหน้าต่างตอบโต้ (Dialog) ที่ใช้ในการแจ้งเตือนหรือรับค่าจากผู้ใช้ ดังรูปข้างล่าง



รูปที่ 6.32 แสดง Dialog พื้นฐาน

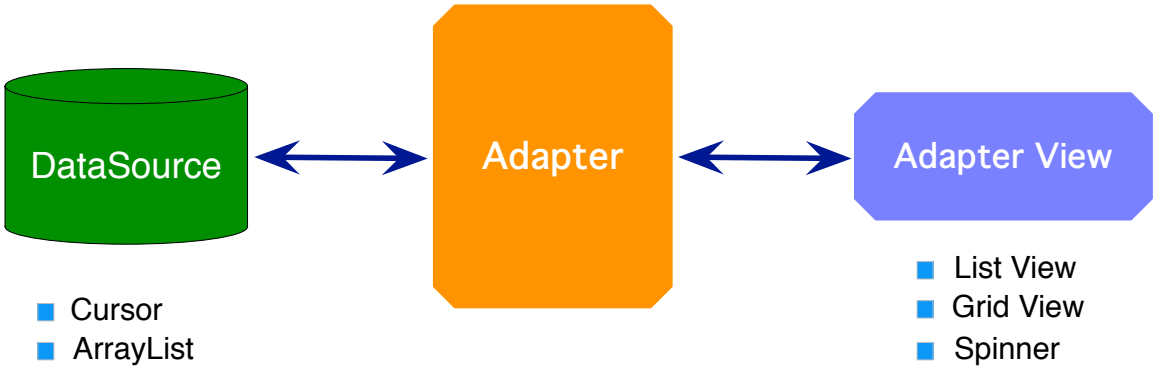
การแจ้งเตือนอีกแบบหนึ่งคือการแจ้งเตือนในลักษณะแสดงเป็นข้อความสั้นๆชั่วระยะเวลาหนึ่ง ให้ปรากฏบนหน้าจอ เรียกว่า Toast แต่จะไม่มีการโต้ตอบกับผู้ใช้แต่อย่างใด โดยใช้คำสั่ง `Toast.makeText(this, “text”, Toast.LENGTH_SHORT).show()` ดังรูปข้างล่าง



รูปที่ 6.33 แสดงการแสดงข้อความแบบ toast

ANDROID ADAPTER

อะแดปเตอร์ (Adapter) ทำหน้าที่ติดต่อระหว่าง AdapterView กับข้อมูลที่ต้องการให้แสดงใน ListView, GridView, spinner หรือ Gallery นอกจากนี้ Adapter ยังใช้ในการจัดการ View กับชุดข้อมูลโดยที่ AdapterView จะสามารถแสดงข้อมูลบางส่วนซึ่งถูกกำหนดโดย Adapter เช่นในกรณีมีข้อมูลมากและไม่สามารถแสดงได้ในครั้งเดียว



รูปที่ 6.34 แสดงความสัมพันธ์ของ Adapter

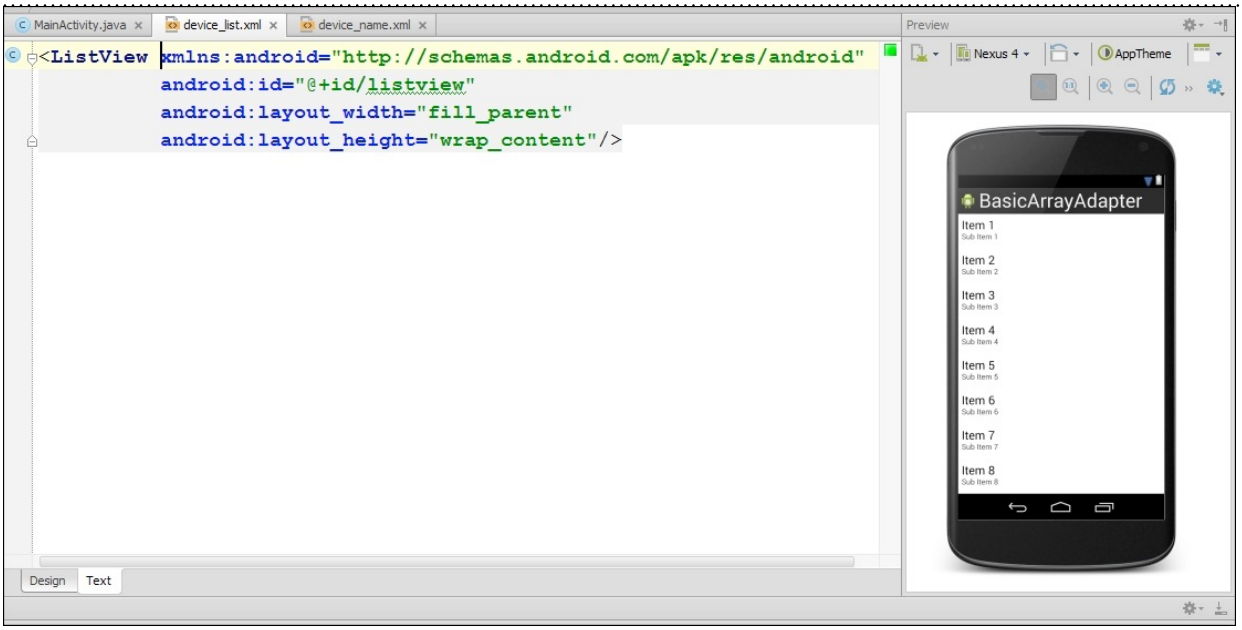
ตัวอย่างการใช้งาน Adapter

1. ArrayAdapter

ตัวอย่างนี้จะแสดงการเพิ่มค่าเข้าไปแสดงบน listview และลบค่าเมื่อมีการกดที่ตำแหน่ง list

โค้ดภายในไฟล์ device_list.xml

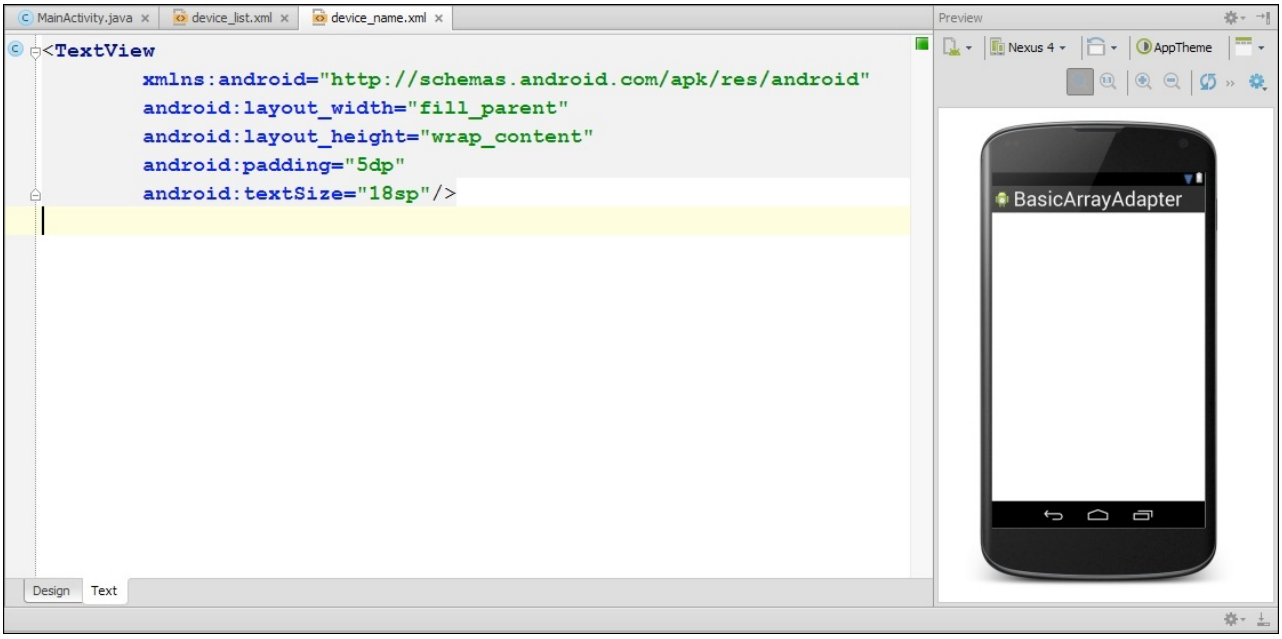
```
<ListView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/listview"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```



รูปที่ 6.35 แสดงรายละเอียดภายในไฟล์ device_list.xml

โค้ดภายในไฟล์ device_name.xml

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:padding="5dp"
    android:textSize="18sp" />
```



รูปที่ 6.36 แสดงรายละเอียดภายในไฟล์ device_name.xml

ซึ่งภายในไฟล์ MainActivity.java มีรายละเอียดดังนี้

```
package com.example.basicarrayadapter;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class MainActivity extends Activity {

	ListView listview;
	private ArrayAdapter<String> mNewDevicesArrayAdapter;
	String[] values = new String[] { "Android", "iPhone", "WindowsMobile",
		"Blackberry", "WebOS", "Ubuntu", "Windows7", "Max OS X", "Linux",
			"OS/2", "Ubuntu", "Windows7", "Max OS X", "Linux", "OS/2",
			"Ubuntu", "Windows7", "Max OS X", "Linux", "OS/2", "Android",
			"iPhone", "WindowsMobile" };

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.device_list);

		listview = (ListView) findViewById(R.id.listview);
		mNewDevicesArrayAdapter = new ArrayAdapter<String>(this,
				R.layout.device_name);
```

```
            for (int i = 0; i < values.length; ++i) {
                  mNewDevicesArrayAdapter.add(values[i]);
            }

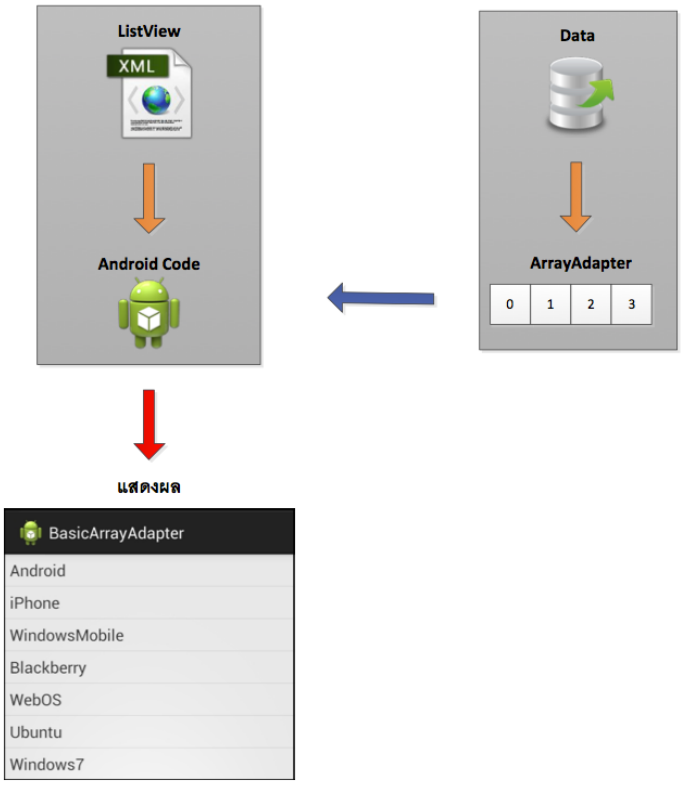
            listview.setAdapter(mNewDevicesArrayAdapter);

            listview.setOnItemClickListener(new AdapterView.OnItemClickListener() {
                  public void onItemClick(AdapterView<?> parent, View view,
                              int position, long id) {

                        String item = (String) parent.getItemAtPosition(position);
                        mNewDevicesArrayAdapter.remove(item);
                        mNewDevicesArrayAdapter.notifyDataSetChanged();
                  }
            });
      }
}
```

สำหรับการแสดงข้อมูลโดยใช้ ArrayAdapter จะมีขั้นตอนดังนี้

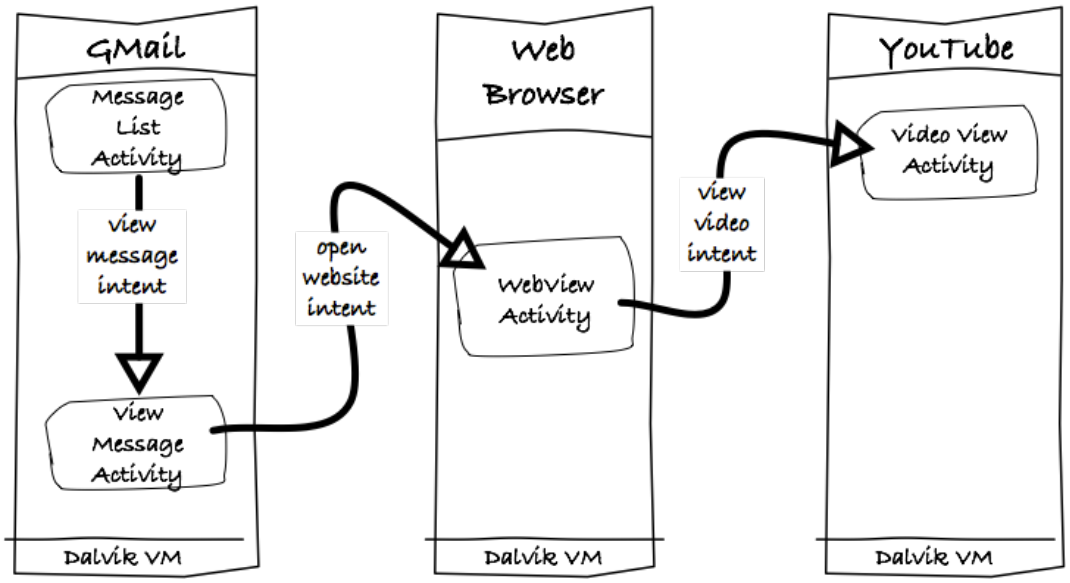
- ทำการอ้างอิง ID ของตัว ListView ที่เป็น xml เข้ากับโค้ดโปรแกรม
- ทำการอ้างอิง Layout ของ xml ที่จะแสดงค่าใน ListView
- ทำการเพิ่มข้อมูลลงใน ArrayAdapter
- ทำการเชื่อม ArrayAdapter เข้ากับ listview เพื่อแสดงข้อมูล



รูปที่ 6.37 แสดงการทำงานของโปรแกรม

ANDROID INTENT

Intent เป็นรูปแบบการส่งข้อมูลระหว่างโปรแกรมภายในระบบปฏิบัติการแอนดรอยด์ ซึ่งจะพบบ่อยในการใช้เพื่อเรียก Activity ตัวอื่น โดยสามารถส่งข้อมูลได้หลายรูปแบบ ทั้งยังเป็นช่องทางการส่งข้อมูลที่นิยมใช้มากในการเขียนโปรแกรมเช่นกัน ดังรูปข้างล่าง



รูปที่ 6.38 แสดงการใช้ intent แยกการทำงานของ Activity แต่ละตัว

ประเภทของ Intents

Intent แบ่งได้เป็น 2 ประเภทหลักดังนี้

1. Implicit Intent เป็น Intent ที่ระบบปฏิบัติการแอนดรอยด์ได้เตรียมไว้ให้แล้ว สามารถเรียกใช้งานได้ในทันที ตัวอย่างเช่น เว็บเบราว์เซอร์ เครื่องมือการค้นหา รายชื่อผู้ติดต่อในโทรศัพท์ การโทรออก และแผนที่ เป็นต้น ตัวอย่างการประกาศ Intent เพื่อเรียกเว็ปไซต์ค้นหาข้อมูล ดังตัวอย่างข้างล่าง

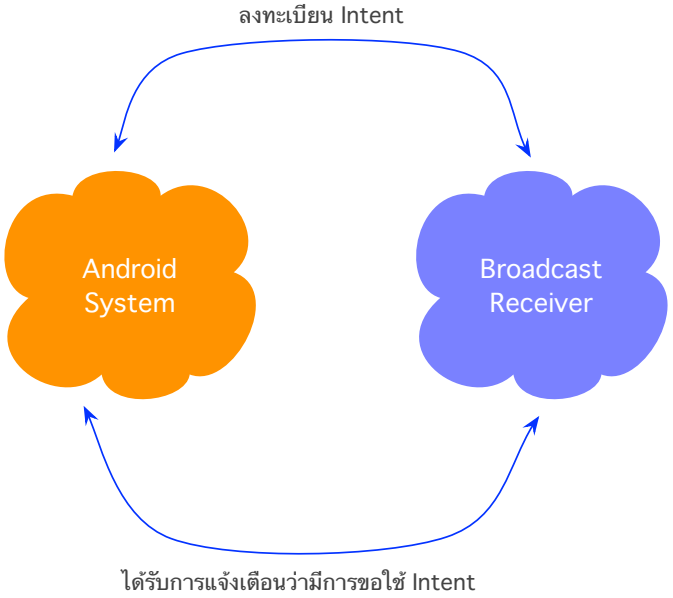
```
    Intent i = new Intent(Intent.ACTION_WEB_SEARCH);
```

2. Explicit Intent เป็น intent ที่ต้องสร้างขึ้นเองเพื่อกำหนดการทำงานเฉพาะอย่างตามที่ต้องการ ตัวอย่างการประกาศ Intent เพื่อเรียก activity ที่ต้องการเช่น ActivityTwo.class ดังตัวอย่างข้างล่าง

```
    Intent i = new Intent(this, ActivityTwo.class);
```

BROADCAST RECEIVER

Broadcast receivers เป็นส่วนที่จะทำงานก็ต่อเมื่อได้รับการตอบสนองหรือเกิดเหตุการณ์ตามที่โปรแกรมได้กำหนดเอาไว้ล่วงหน้าด้วยการลงทะเบียนเพื่อรับการแจ้งของระบบดังแสดงในรูปข้างล่าง โดยเงื่อนไขเหล่านี้จะสามารถตอบสนองในขณะที่ผู้ใช้กำลังเปิดโปรแกรมอยู่หรือแม้จะปิดโปรแกรมไปแล้วก็ตามที ตัวอย่างการทำงานเช่น ในขณะที่ผู้ใช้กำลังขับรถหรือไม่สะดวกในการรับสายแต่เมื่อมีคนโทรศัพท์เข้ามาตัวโปรแกรมก็ทำการตัดสายที่กำลังโทรฯเข้ามาทันทีและจะทำการส่งข้อความ (SMS) กลับไปหาคนที่กำลังโทรเข้ามาโดยอัตโนมัติ หรืออีกตัวอย่างเช่นในกรณีที่ตั้งค่าไว้ว่าเมื่อเครื่องบูทเสร็จเรียบร้อย (ACTION_BOOT_COMPLETED) ก็ให้ระบบปฏิบัติการทำการเปิดโปรแกรมที่ได้ลงทะเบียนใน Broadcast receiver ให้ทำขึ้นมาทันที



รูปที่ 6.39 แสดงการใช้ทำงานของ intent

ขั้นตอนการพัฒนาโปรแกรม Broadcast receivers

- ทำการตั้งค่าใน AndroidManifest.xml ตามที่ได้ลงทะเบียนไว้ หรือสามารถตั้งค่าแบบตามสถานการณ์ที่เปลี่ยนแปลงผ่านฟังก์ชัน Context.registerReceiver()
- เขียนคลาสขึ้นมาโดยสืบทอดต่อมาจากคลาส BroadcastReceiver
- ภายในจะมีฟังก์ชัน onReceive() สำหรับกำหนดการทำงานของโปรแกรมเมื่อเกิดเหตุการณ์ (event) ตามที่ได้ตั้งค่าไว้

System broadcasts

เหตุการณ์ของระบบ (system events) ที่ได้ถูกกำหนดเป็นค่าคงที่ไว้ภายในระบบปฏิบัติการแอนดรอยด์ ดังแสดงในตารางข้างล่างนี้

ตาราง 6.2 แสดงรายการเหตุการณ์ของระบบ

Event	Usage
Intent.ACTION_BATTERY_LOW	ระดับแบตเตอรี่อยู่ในระดับต่ำ
Intent.ACTION_BATTERY_OKAY	ระดับแบตเตอรี่อยู่ในระดับดี
Intent.ACTION_BOOT_COMPLETED	แอนดรอยด์บูทเสร็จเรียบร้อย
Intent.ACTION_DEVICE_STORAGE_LOW	ระดับหน่วยความจำอยู่ในระดับต่ำ
Intent.ACTION_DEVICE_STORAGE_OK	ระดับหน่วยความจำอยู่ในระดับดี
Intent.ACTION_HEADSET_PLUG	ตรวจสอบหูฟังว่าต่ออยู่หรือไม่
Intent.ACTION_LOCALE_CHANGED	เมื่อมีการเปลี่ยนแปลงภาษาของอุปกรณ์
Intent.ACTION_MY_PACKAGE_REPLACED	อัพเดทแทนโปรแกรมเดิม
Intent.ACTION_PACKAGE_ADDED	ติดตั้งโปรแกรมใหม่
Intent.ACTION_POWER_CONNECTED	ต่อสายไฟเข้ากับอุปกรณ์
Intent.ACTION_POWER_DISCONNECTED	ถอดสายไฟออกจากอุปกรณ์
KeyChain.ACTION_STORAGE_CHANGED	เกิดการเปลี่ยนแปลง keystore
BluetoothDevice.ACTION_ACL_CONNECTED	Bluetooth ACL เกิดการเชื่อมต่อ
AudioManager.ACTION_AUDIO_BECOMING_NOISY	เลือก audio output ที่มีสัญญาณรบวนน้อย

ตัวอย่างโปรแกรมตรวจสอบการเชื่อมต่อกับหูฟังโดยใช้เหตุการณ์ของระบบชื่อว่า ACTION_HEAD-SET_PLUG ซึ่งใช้ในการตรวจสอบการเชื่อมต่อชุดหูฟัง ซึ่งถ้าต่อหูฟังอยู่ก็จะส่งค่าเป็น “1” ถ้าไม่ก็จะส่งค่า “0” แทน

ตัวอย่างโปรแกรมทดสอบการบรอดท์แคสท์ เมื่อมีการเสียบหูฟังเข้าอุปกรณ์แอนดรอยด์

```
// MainActivity.java

package com.example.demoeebuubroadcastreceiver;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
public class MainActivity extends Activity {
     @Override
protected void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.activity_main);
          IntentFilter intentfilter = new IntentFilter(Intent.ACTION_HEADSET_PLUG);
```

```
          MyBroadCast mybroadcast = new MyBroadCast();
          registerReceiver(myboradcast,intentfilter);
     }


     @Override
public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
          getMenuInflater().inflate(R.menu.main, menu);
          return true;
     }
}
```

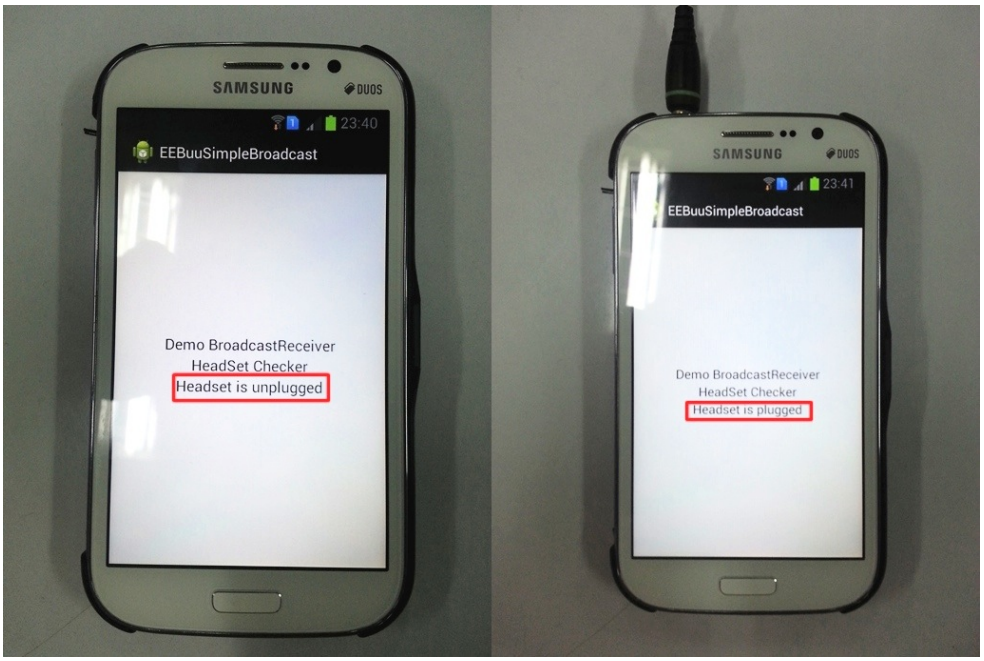
```
// MyBroadCast..java

package com.example.demoeebuubroadcastreceiver;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class MyBroadCast extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction().equals(Intent.ACTION_HEADSET_PLUG)) {

            int state = intent.getIntExtra("state", -1);
            switch (state) {
                case 0:
      Toast.makeText(context, "Headset is unplugged", Toast.LENGTH_LONG).show();
                    break;
                case 1:
   Toast.makeText(context, "Headset is plugged", Toast.LENGTH_LONG).show();
                    break;
                default:
    Toast.makeText(context, "Unknow", Toast.LENGTH_LONG).show();
            }
        }
    }
}
```

การแสดงผล (ด้านซ้ายไม่ได้ต่อหูฟัง ด้านขวาต่อหูฟัง)



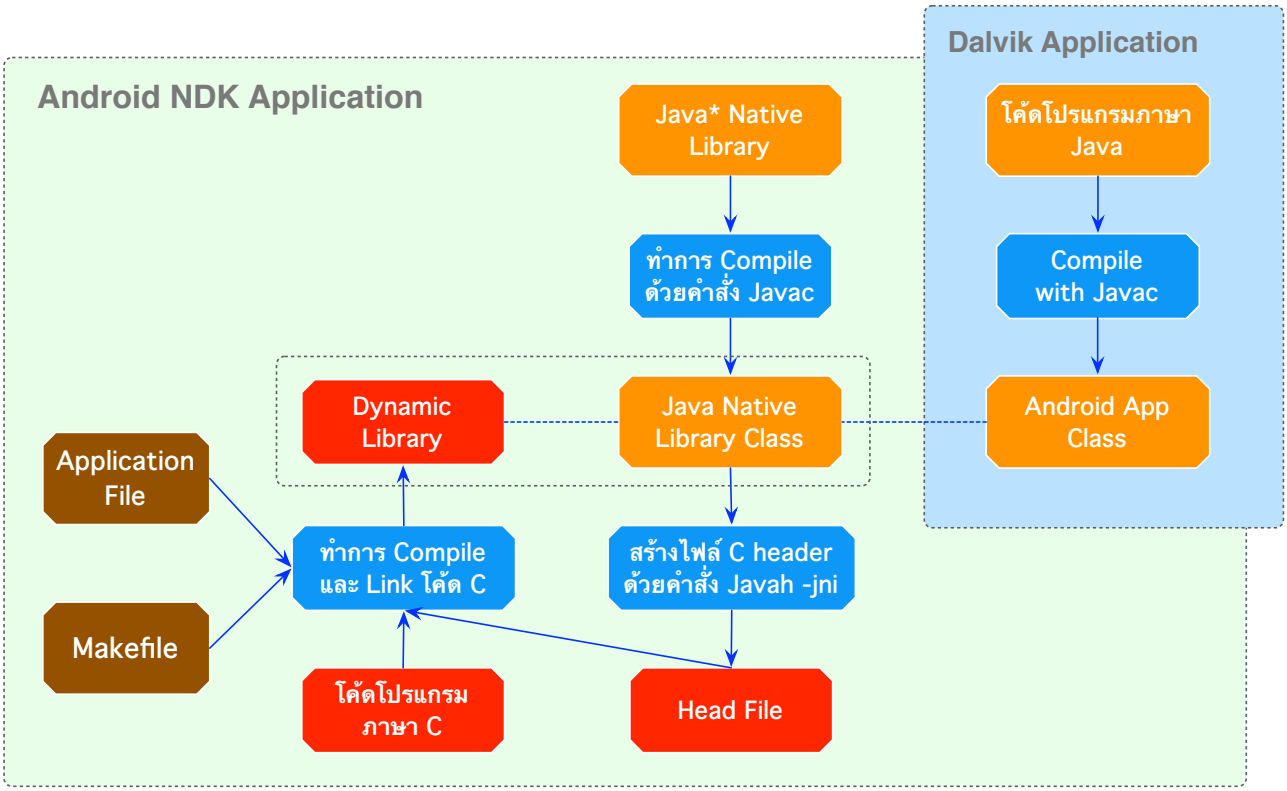
รูปที่ 6.40 แสดงการตรวจจับของโปรแกรมในการตรวจสอบการเสียบหู

การพัฒนาโปรแกรมด้วย Android Native Development Kit

พื้นฐาน ANDROID NDK

โปรแกรมแอนดรอยด์สามารถแยกออกได้เป็น 2 ประเภท ดังแสดงในรูปข้างล่าง ได้แก่

- โปรแกรมดาร์วิค (Dalvik applications) คือโปรแกรมที่รวมโปรแกรมภาษาจาวา และ SDK API ตามมาตราฐานของแอนดรอยด์ รวมทั้งการรวมเอาไฟล์ประกอบต่างๆ (resource files) ตัวอย่าง เช่น ไฟล์ XML ไฟล์รูปชนิด PNG รวมเข้าด้วยกันจนกลายเป็นไฟล์ APK
- โปรแกรมแอนดรอยด์ NDK (Android NDK applications) ซึ่งรวมโค้ดโปรแกรมภาษาจาวา ไฟล์ประกอบต่างๆ (resource files) รวมทั้งโค้ดโปรแกรมภาษา C/C++ (หรืออาจมีโค้ดโปรแกรมภาษาแอสเซมบลี้ (Assembly) โดยที่โค้ดโปรแกรมภาษา C/C++ จะถูกคอมไพล์เป็นไลบรารี (Dynamic linked library) ซึ่งมีนามสกุลไฟล์เป็น ".so" แล้วจะถูกเรียกโดยโค้ดโปรแกรมภาษาจาวา ตามกลไกของ JNI (JAVA Native Interface) และสุดท้ายทั้งหมดก็จะถูกรวมกลายมาเป็นไฟล์ APK



รูปที่ 6.41 แสดงกลไกการสร้างโปรแกรม NDK และ โปรแกรมแอนดรอยด์ทั่วไป

Android NDK เป็นอีกชุดเครื่องมือในการพัฒนาโปรแกรมประเภท Android NDK โดยนักพัฒนาจะเขียนโปรแกรมทั้งในส่วนของโปรแกรมภาษา C/C++ หรือ ภาษา Assembly (Native Language) และส่วนโปรแกรมภาษาจาวา ซึ่งทำให้สามารถเพิ่มประสิทธิภาพการทำงานโดยรวมของโปรแกรมประยุกต์ เหมาะในการพัฒนาโปรแกรมประยุกต์ประเภทการเข้า-ถอดรหัสสัญญาณวิดีโอ (Video encoding and decoding) การประมวลภาพ (Image processing) หรือการคำนวณทางคณิตศาสตร์ที่ซับซ้อนและต้องการความเร็วในการคำนวณโดยตรงจากหน่วยประมวลผลกลาง เป็นต้น เหตุผลหลักที่ทำให้การเขียนโปรแกรมด้วย Native language (C/C++, Assembly) สามารถเพิ่มประสิทธิภาพการทำงานของโปรแกรมประยุกต์ให้สามารถตอบสนองการทำงาน และการประมวลผลข้อมูล หรือการคำนวณทางคณิตศาสตร์ได้ดีกว่าการเขียนโปรแกรมด้วยภาษาจาวา เพียงอย่าง ได้แก่

- โค้ดโปรแกรม (C/C++ หรือ Assembly) จะถูกคอมไพล์จนมาเป็นรหัสไบนารี (binary code) เรียบร้อยแล้ว ซึ่งสามารถทำงานได้ทันทีในระดับระบบปฏิบัติการ แต่ในขณะที่โค้ดโปรแกรม JAVA เริ่มต้นจะถูกแปลให้กลายเป็น JAVA byte-code โดยตัว DVM (Dalvik Virtual Machine) ถึงแม้ว่าแอนดรอยด์ตั้งแต่รุ่น 2.2 หลังจากนั้นจะมีการเพิ่มการวิเคราะห์และปรับแต่งประสิทธิภาพให้กับ JAVA byte-code ด้วยเทคนิคการแปลงบ้างส่วนให้เป็นรหัสไบนารี (binary code) ด้วย JIT Compiler (Just-In-Time Compiler) แล้วก็ตาม แต่ก็ยังทำงานได้ช้ากว่าโค้ดโปรแกรม C/C++ อยู่ดี
- การพัฒนาโปรแกรมด้วย Android NDK จะช่วยให้นักพัฒนาสามารถเข้าถึงและเรียกใช้ความสามารถพิเศษบางอย่างของหน่วยประมวลผลกลางที่มีอยู่ในบางตัว ซึ่งไม่สามารถเรียกผ่าน Android SDK ได้

ตัวอย่างเช่น ตัว NEON ซึ่งเป็นหน่วยประมวลผลร่วมของ ARM CPU เป็นเทคโนโลยีแบบ SIMD (Single Instruction Multiple Data) ที่มีการยอมให้มีการประมวลผลส่วนของข้อมูลได้แบบขนาน (Parallel processing) ซึ่งเทคโนโลยีตัวนี้สามารถเพิ่มความเร็วในการประมวลผลทางด้านมัลติมีเดีย เช่น การทำ Video encoding/decoding การแสดงผลในด้านกราฟฟิกทั้งแบบสองมิติและสามมิติ การประมวลผลสัญญาณเสียงพูด เป็นต้น ดังตัวอย่างการใช้งานดังนี้

ไฟล์ Android.mk

```
LOCAL_PATH:= $(call my-dir)

include $(CLEAR_VARS)
LOCAL_MODULE    := neon_utils
LOCAL_SRC_FILES := neon_add.c
LOCAL_CFLAGS += -mfloat-abi=softfp -mfpu=neon -march=armv7
include $(BUILD_STATIC_LIBRARY)

NDK_PATH:=$(call my-dir)/../..

include $(CLEAR_VARS)
LOCAL_MODULE    := test_conditional_load
LOCAL_C_INCLUDES := $(NDK_PATH)/sources/cpufeatures
LOCAL_SRC_FILES := main.c
LOCAL_STATIC_LIBRARIES  :=  neon_utils cpufeatures

include $(BUILD_EXECUTABLE)
include $(NDK_PATH)/sources/cpufeatures/Android.mk
```

ตัวอย่างไฟล์โปรแกรม main.c

```
#include <stdio.h>
#include <cpu-features.h>

int main()
{
    int32_t int32_4[] = {2,3,4,5};

    if (android_getCpuFeatures() & ANDROID_CPU_ARM_FEATURE_NEON)
```

```
    {
        neon_add(int32_4);
        printf("NEON supported\n");
    }
    else
    {
        printf("NEON Not Supported\n");
    }
    printf("values = %d, %d, %d, %d\n", int32_4[0], int32_4[1], int32_4[2],
int32_4[3]);
    return 0;
}
```

ตัวอย่างไฟล์โปรแกรม neon_add.c

```
#include <arm_neon.h>
void neon_add(int32_t * ptr)
{
    int32x4_t vin = vld1q_s32(ptr);
    int32x4_t vout = vaddq_s32(vin, vin);
    vst1q_s32(ptr, vout);
}
```

ความสามารถในการนำโค้ดโปรแกรม native มาใช้ใหม่ (Code Reusability) โดยเฉพาะโปรแกรมแบบเปิด (Open source) และไลบรารีต่างๆ ซึ่งถูกพัฒนาด้วยโปรแกรมภาษา C/C++ ซึ่งถูกใช้อยู่แล้วภายในระบบปฏิบัติการลีนุกซ์ในงานประยุกต์อื่น นักพัฒนาเพียงทำความเข้าใจและปฏิบัติตามเงื่อนไขข้อตกลงทางลิขสิทธิ์การใช้งานโค้ดโปรแกรม ก็สามารถนำมาใช้ร่วมในการพัฒนา Android NDK ได้ทันที เช่น ไลบรารี zlib ที่ใช้สำหรับการบีบอัดข้อมูล หรือ ไลบรารี OpenMAX ที่สามารถนำมาใช้ในการพัฒนาโปรแกรมด้านมัลติมีเดีย สำหรับเล่นและจัดการไฟล์วิดีโอหรือไฟล์เสียง เป็นต้น

เริ่มต้นการพัฒนาโปรแกรม ANDROID NDK

ขั้นตอนการติดตั้ง Android NDK

ดาวน์โหลดโปรแกรมเครื่องมือ Android NDK เวอร์ชันล่าสุดได้จาก Android NDK และทำการติดตั้งและตั้งค่าตามคำแนะนำจากเว็ปไซต์ ที่ขึ้นอยู่กับระบบปฏิบัติการที่นักพัฒนาใช้งานอยู่ เช่น ระบบปฏิบัติการลีนุกซ์ หลังจากแตกไฟล์แล้วให้ตั้งค่าตำแหน่งของไดเรกทอรี Android NDK เป็นตัวแปรระบบภายในไฟล์ ~/.bashrc ดังตัวอย่างข้างล่าง

```
$ cat ~/.bashrc
```

```
...
export USE_CCACHE=0
export CCACHE_DIR=~/ccache
export JAVA_HOME=/usr/lib/jvm/jdk
export ANT_HOME=~/android/ant
export ANDROID_SDK_HOME=~/android/sdk
export ANDROID_NDK_HOME=~/android/ndk
export AOSP_HOME=~/aosp
export
PATH=$HOME/bin:$JAVA_HOME/bin:$ANT_HOME/bin:$ANDROID_SDK_HOME/tools:$ANDROID_
SDK_HOME/platform-tools:$ANDROID_NDK_HOME:$PATH
```

ขั้นตอนการพัฒนาโปรแกรม

ขั้นตอนการพัฒนาโปรแกรม Android NDK มีด้วยกัน 2 ส่วนใหญ่ๆคือ

1.การเขียนโปรแกรม native ด้วยภาษา C/C++ เพื่อสร้างเป็นไลบรารี (Dynamic link library)

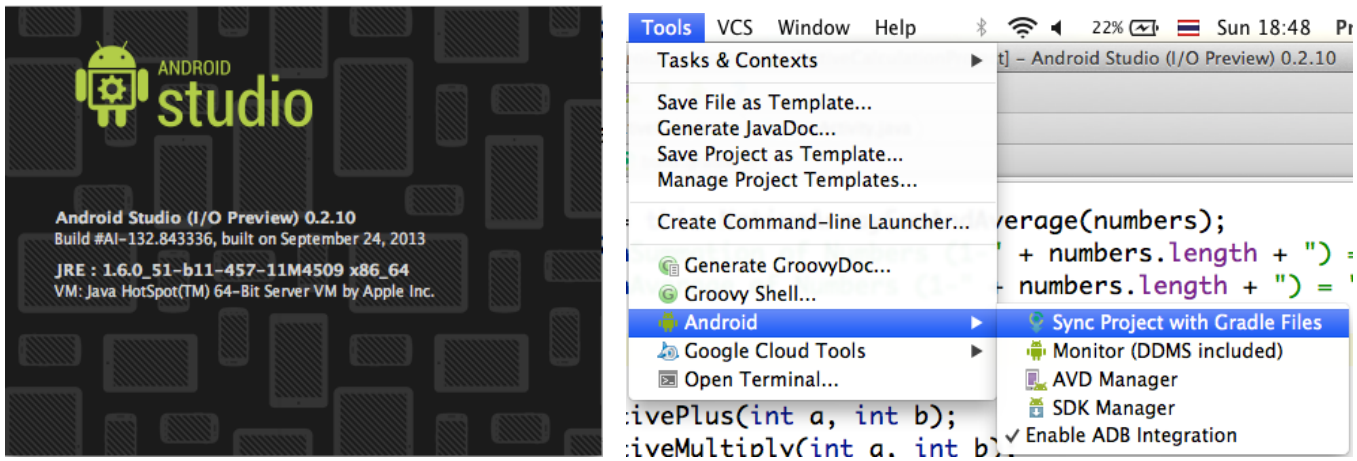
2.การเขียนโปรแกรมประยุกต์เพื่อเรียกใช้งานไลบรารี ด้วยภาษาจาวา

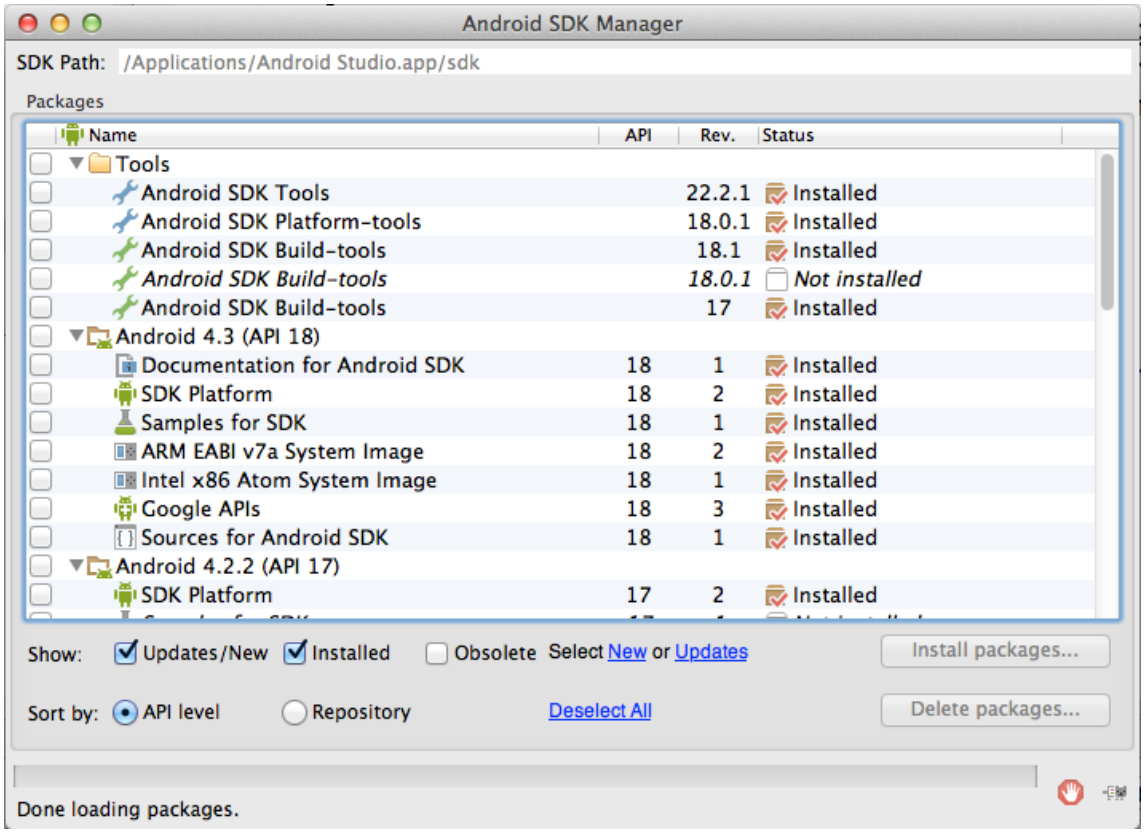
ตัวอย่างการพัฒนาโปรแกรม native ในขั้นตอนนี้จะใช้ Android NDK รุ่น r7b และ GNU make รุ่น 3.81 ดังขั้นตอนข้างล่างนี้

```
$ ls -al ~/android-ndk-r7b/
total 320
drwxr-xr-x@ 19 sriborrirux  staff     646 Mar 17  2012 .
drwxr-xr-x+ 94 sriborrirux  staff    3196 Oct  6 02:12 ..
-rw-r--r--@  1 sriborrirux  staff    6148 Mar 17  2012 .DS_Store
-rw-r-----@  1 sriborrirux  staff    1306 Dec 19  2011 GNUmakefile
-rw-r-----@  1 sriborrirux  staff    1360 Dec 19  2011 README.TXT
-rw-r--r--@  1 sriborrirux  staff       4 Jan 25  2012 RELEASE.TXT
drwxr-xr-x@  6 sriborrirux  staff     204 Jan 25  2012 build
drwxr-xr-x@ 29 sriborrirux  staff     986 Jan 25  2012 docs
-rw-r-----@  1 sriborrirux  staff     201 Dec 19  2011 documentation.html
-rwxr-x---@  1 sriborrirux  staff    2985 Dec 19  2011 ndk-build
-rw-r-----@  1 sriborrirux  staff     279 Dec 19  2011 ndk-build.cmd
-rwxr-x---@  1 sriborrirux  staff   21832 Dec 19  2011 ndk-gdb
-rwxr-xr-x@  1 sriborrirux  staff  103928 Jan 25  2012 ndk-stack
drwxr-x---@  8 sriborrirux  staff     272 Jan 25  2012 platforms
drwxr-xr-x@  3 sriborrirux  staff     102 Jan 25  2012 prebuilt
drwxr-x---@ 17 sriborrirux  staff     578 Oct  6 02:09 samples
drwxr-xr-x@  5 sriborrirux  staff     170 Dec 18  2010 sources
drwxr-xr-x@ 10 sriborrirux  staff     340 Jan 25  2012 tests
drwxr-xr-x@  4 sriborrirux  staff     136 Jan 25  2012 toolchains

$ ndk-build --version
GNU Make 3.81
Copyright (C) 2006  Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
This program built for i386-apple-darwin11.3.0
```

ส่วนการพัฒนาโปรแกรมประยุกต์ในขั้นตอนที่ 2 นั้นจะทำการพัฒนาด้วยโปรแกรม Android Studio ที่ทางบริษัทกูเกิล ได้พัฒนาต่อยอดมาจาก InteliJ IDEA เพื่อมาแทนที่โปรแกรม Eclipse IDE (ที่ต้องติดตั้ง Android ADT Plugin เพิ่มเติม) ซึ่งตัวโปรแกรม Android Studio ก็ได้รวมเครื่องมือสำหรับการพัฒนาและเครื่องมือจัดการต่างๆสำหรับโปรแกรมแอนดรอยด์ไว้ครบถ้วนดังแสดงในรูปข้างล่าง





รูปที่ 6.42 หน้าต่าง Android SDK Manager

ทางบริษัทกูเกิลจะมีการปรับปรุงให้โปรแกรม Android Studio สามารถรองรับการพัฒนาโปรแกรม Android NDK ได้ประมาณต้นปี พ.ศ. 2557 เป็นต้นไป แต่อย่างไรก็ตามเนื้อหาต่อไปนี้ จะเป็น

ขั้นตอนการปรับแต่งบางอย่างซึ่งได้ทำในช่วงปลายปี พ.ศ. 2556 เพื่อทำให้โปรแกรมประยุกต์ที่เขียนบน Android Studio สามารถติดต่อกับโปรแกรม native C/C++ ได้

ตัวอย่างโปรแกรม HELLO WORLD ด้วย ANDROID NDK

นักพัฒนาสามารถเรียนรู้โปรแกรมตัวอย่างได้จากไดเรกทอรี **~/android-ndk-r7b/samples/**

```
$ ls -al
total 0
drwxr-x---@ 17 sriborrirux  staff  578 Oct  6 02:09 .
drwxr-xr-x@ 19 sriborrirux  staff  646 Mar 17  2012 ..
drwxr-x---@  7 sriborrirux  staff  238 Oct  1  2011 bitmap-plasma
drwxr-x---@  7 sriborrirux  staff  238 Jan 25  2012 hello-gl2
drwxr-x---@ 10 sriborrirux  staff  340 Oct  5 21:26 hello-jni
drwxr-x---@  8 sriborrirux  staff  272 Jul 14  2011 hello-neon
drwxr-x---@  4 sriborrirux  staff  136 Jul 14  2011 module-exports
drwxr-x---@  6 sriborrirux  staff  204 Jan 25  2012 native-activity
drwxr-x---@  8 sriborrirux  staff  272 Oct  1  2011 native-audio
drwxr-x---@ 10 sriborrirux  staff  340 Nov  3  2011 native-media
drwxr-x---@  7 sriborrirux  staff  238 Oct  1  2011 native-plasma
drwxr-x---@  7 sriborrirux  staff  238 Oct  1  2011 san-angeles
drwxr-x---@  3 sriborrirux  staff  102 Dec 18  2010 test-libstdc++
drwxr-x---@  8 sriborrirux  staff  272 Jul 14  2011 two-libs
```

ในที่นี้จะหยิบยกตัวอย่างโปรแกรมในไดเรกทอรี hello-jni มาปรับใช้กับโปรแกรม Android Studio โดยให้ทำตามขั้นตอนดังนี้

ขั้นตอนที่ 1: คอมไพล์โปรแกรม **hello-jni.c**

```
$ cd samples/hello-jni/
$ ls
AndroidManifest.xml  default.properties  jni  res  src  tests
$ cd jni/
```

เปิดไฟล์ hello-jni.c เพื่อแก้ไขชื่อฟังก์ชันใหม่จาก

`Java_com_example_hellojni_HelloJni_stringFromJNI`

เป็น

`Java_com_example_hellondk_MainActivity_stringFromJNI`

```
$ vim hello-jni.c

#include <string.h>
#include <jni.h>

Java_com_example_hellondk_MainActivity_stringFromJNI( JNIEnv* env,
            jobject thiz ) {
```

```
    return (*env)->NewStringUTF(env, "Hello from JNI !");
}
```

ทำการคอมไพล์โดยใช้คำสั่ง ndk-build

```
$ ndk-build
Gdbserver      : [arm-linux-androideabi-4.4.3] libs/armeabi/gdbserver
Gdbsetup       : libs/armeabi/gdb.setup
Compile thumb  : hello-jni <= hello-jni.c
SharedLibrary  : libhello-jni.so
Install        : libhello-jni.so => libs/armeabi/libhello-jni.so
$ cd ..
```

เมื่อถอยออกมาจากไดเรกทอรี jni/ จะเห็นไดเรกทอรีที่ถูกสร้างใหม่จากการคอมไพล์ชื่อ libs ดังแสดงในคำสั่ง ls ข้างล่างนี้

```
$ ls
AndroidManifest.xml   res  default.properties
libs    src    jni     obj    tests
```

ขั้นตอนที่ 2: การเตรียมไลบรารี เพื่อให้ใช้ได้กับ Android Studio

ในขั้นตอนนี้จะแตกต่างจากการพัฒนาด้วยโปรแกรม Eclipse IDE คือจะต้องมีการปรับเปลี่ยนบางอย่างพิเศษ ได้แก่ การเตรียมไลบรารีในรูปแบบ .jar แทนที่จะเป็น .so เพื่อให้ Android Studio อ้างอิงผ่านการตั้งค่าในไฟล์ build.gradle ได้ การเตรียมไลบรารีจะมี 5 ขั้นตอนย่อยดังนี้

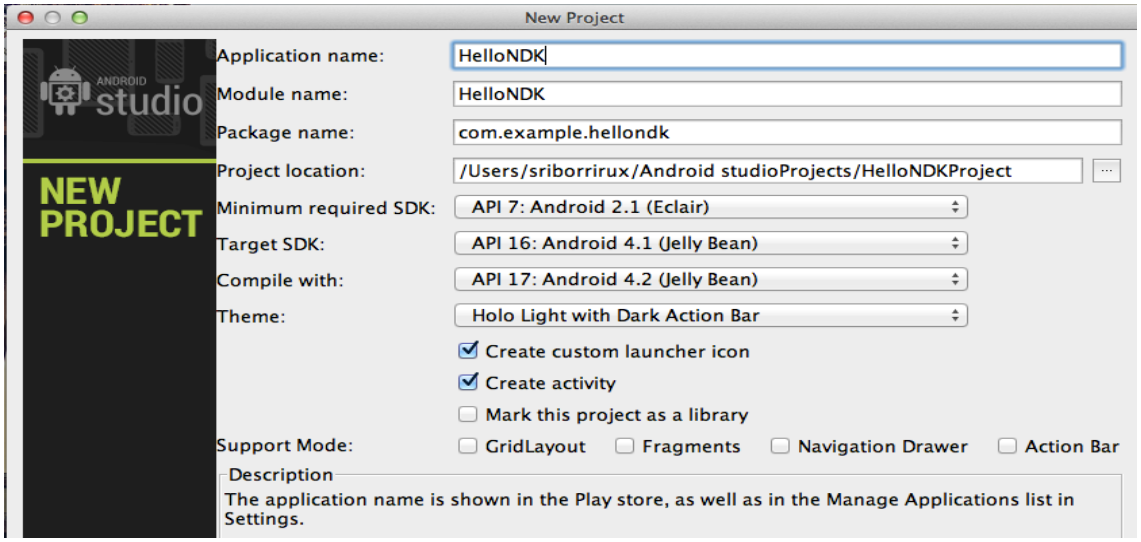
```
$ mv libs lib    <---- 1. เปลี่ยนชื่อไดเรกทอรี libs ให้เป็น lib
$ zip -r native-libs.zip lib/  <---- 2. ทำการแพ็กไดเรกทอรี lib ให้เป็นไฟล์ไลบารี
  adding: lib/ (stored 0%)
  adding: lib/armeabi/ (stored 0%)
  adding: lib/armeabi/gdb.setup (deflated 54%)
  adding: lib/armeabi/gdbserver (deflated 39%)
  adding: lib/armeabi/libhello-jni.so (deflated 48%)
(reverse-i-search)`mv': mv libs lib

$ mv native-libs.zip native-libs.jar <---- 3. เปลี่ยนชื่อนามสกุลไฟล์ให้เป็น .jar
$ ls
AndroidManifest.xml   lib   res   default.properties
src   jni   obj   tests   native-libs.jar
$ mv lib libs <---- 4. เปลี่ยนชื่อไดเรกทอรีกลับมาเป็น libs เช่นเดิม
$ mv native-libs.jar libs/ <---- 5. ย้ายไฟล์ไลบารีกลับไปอยู่ในไดเรกทอรี libs
```

สำหรับการสร้างโปรแกรมภาษาจาวา ด้วย Android Studio เพื่อเรียกใช้ไลบรารี hello-jni นั้น สามารถทำตามขั้นตอนดังนี้

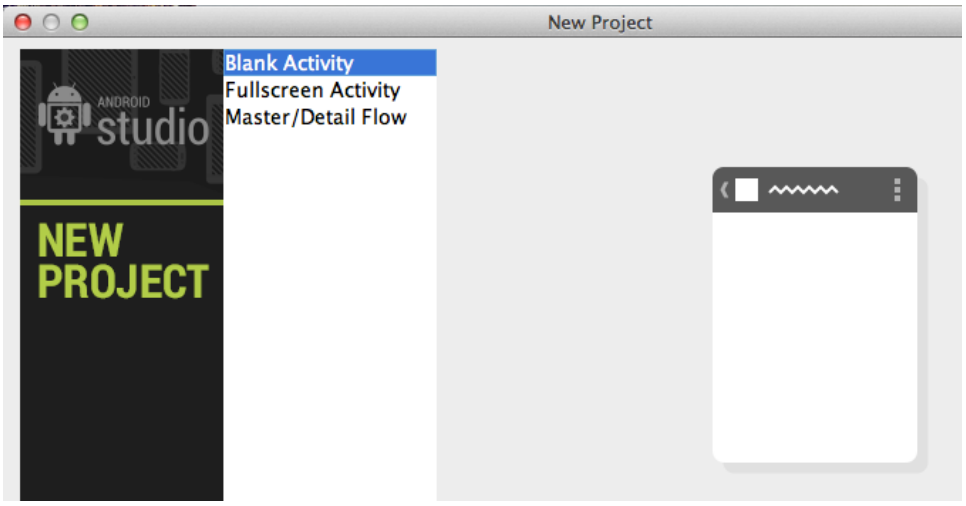
ขั้นตอนที่ 1: สร้างโปรเจ็คใหม่ด้วย Android Studio

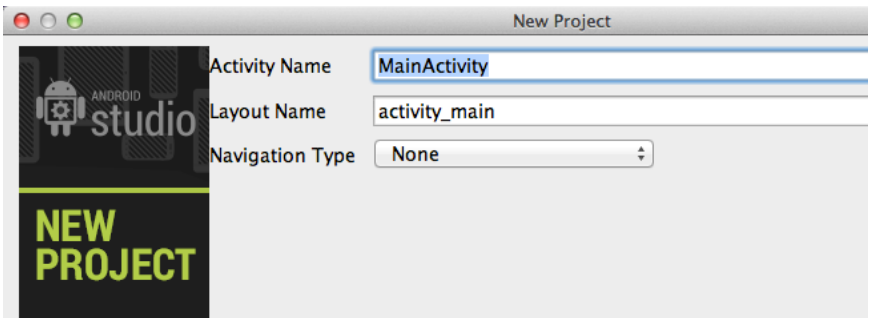
เปิดโปรแกรม Android Studio แล้วเลือกเมนู File -> New Project แล้วตั้งชื่อ Application name ว่า “HelloNDK” และตั้งชื่อ Package name ว่า “com.example.hellondk”



รูปที่ 6.43 ตั้งค่ารายละเอียดของโปรแกรม NDK

เลือกชนิดโปรแกรมเป็น Blank Activity แล้วตั้งชื่อ Activity ว่า “MainActivity”

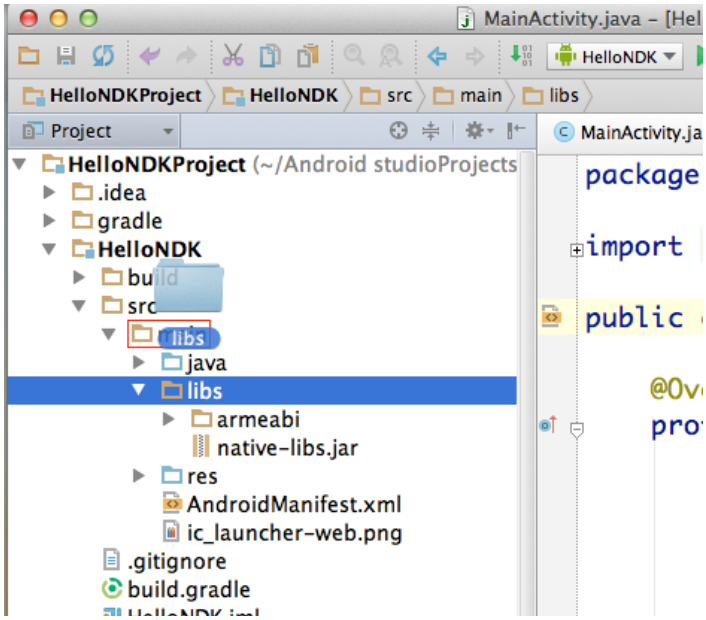




รูปที่ 6.44 กำหนด template ให้โปรแกรมและตั้งค่า Activity

ขั้นตอนที่ 2: นำเข้าไลบรารี hello-jni มาวางไว้ในโปรเจ็ค

โดยการใช้เมาส์ลากไดเรกทอรี libs/ ที่อยู่ภายใต้ไดเรกทอรี `~/android-ndk-r7b/samples/hello-jni/` แล้วมาวางไว้ในโปรเจ็คภายใต้ src/main



รูปที่ 6.45 การนำเข้าไดเรกทอรี libs

ขั้นตอนที่ 3: แก้ไขไฟล์ build.gradle

ภายในโปรเจ็คจะปรากฏไฟล์ชื่อว่า build.gradle ซึ่งเป็นไฟล์ที่ใช้ในการตั้งค่าเพิ่มเติมสำหรับคอมไพล์ ให้เพิ่ม `task nativeLibsToJar, tasks.withType (Compile)` และ `dependencies` เข้าไปในไฟล์ build.gradle ให้เหมือนข้างล่างนี้

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:0.5.+'
    }
}
apply plugin: 'android'
repositories {
    mavenCentral()
}

android {
    compileSdkVersion 17
    buildToolsVersion "17.0.0"
    defaultConfig {
        minSdkVersion 7
```

```
        targetSdkVersion 16
    }
}

task nativeLibsToJar(
    type: Zip,
    description: 'create a jar archive of the native libs') {
    destinationDir file("src/main/libs/native-libs")
    baseName 'native-libs'
    extension 'jar'
    from fileTree(dir: 'src/main/libs', include: '**/*.so')
    into 'lib/'
}

tasks.withType (Compile) {
    compileTask -> compileTask.dependsOn (nativeLibsToJar)
}

dependencies {
    compile fileTree(dir: "src/main/libs/native-libs", include:
'native-libs.jar')
}
```

ขั้นตอนที่ 4: เขียนโปรแกรมภาษาจาวา เพื่อให้ทำงานตามที่ต้องการ

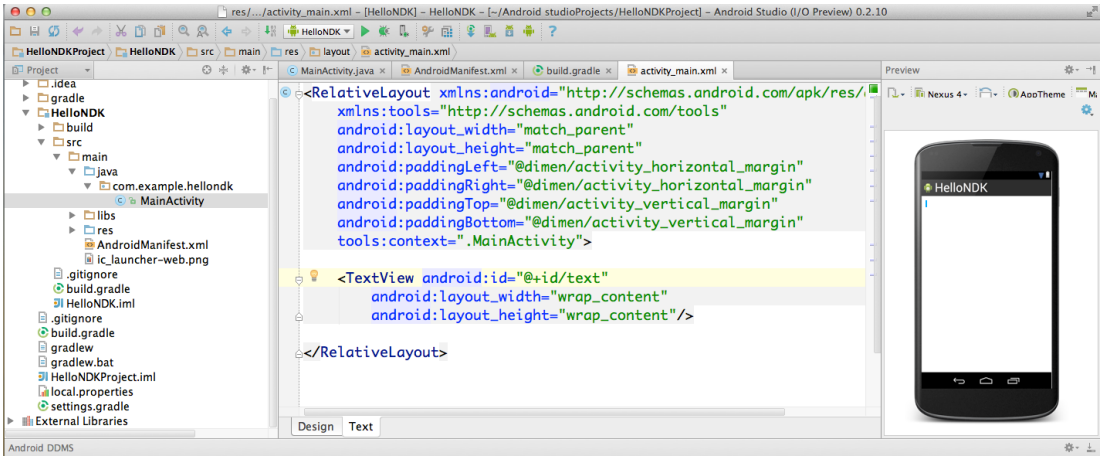
ให้แก้ไขไฟล์ MainActivity.java และไฟล์ activity_main.xml ให้มีรายละเอียดดังนี้

```
// MainActivity.java
package com.example.hellondk;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.TextView;
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView tv = new TextView(this);
        tv =(TextView)findViewById(R.id.hello);
        tv.setText(stringFromJNI());
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
```

```
    public native String  stringFromJNI();
 static {
        System.loadLibrary("hello-jni");
        }
}
```

แก้ไขไฟล์ activity_main.xml โดยการเพิ่ม android:id="@+id/text" และ

```
        android:textSize="40dp"
        android:text="No Message..."
```

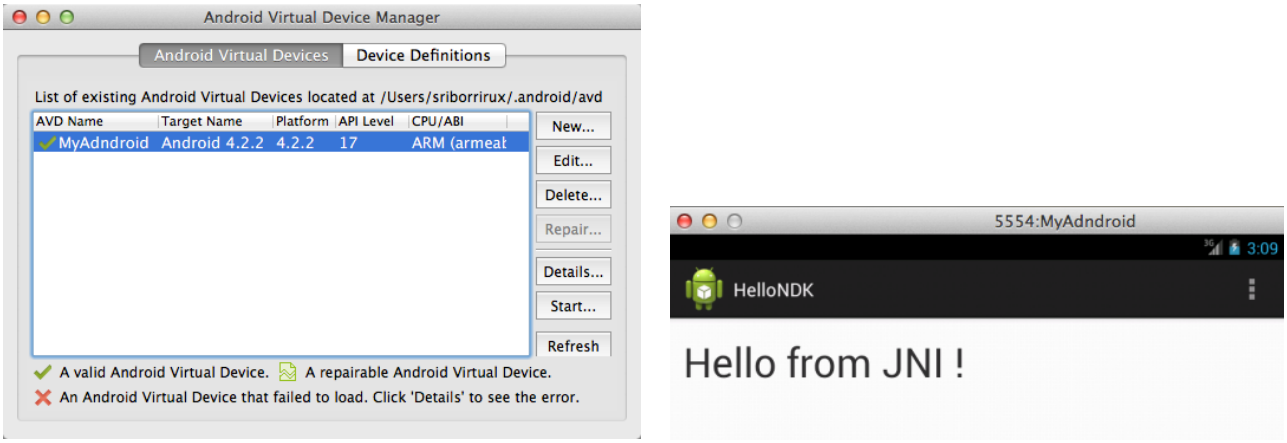


รูปที่ 6.46 แสดงรายละเอียดภายในไฟล์ activity_main.xml

```
// activity_main.xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="40dp"
        android:text="No Message..."/>
</RelativeLayout>
```

ขั้นตอนที่ 5: รันโปรแกรมประยุกต์เพื่อเรียกใช้ไลบรารี

ทำการสร้าง Android Virtual Device และรันโปรแกรม HelloNDK ดังแสดงในรูปข้างล่างนี้

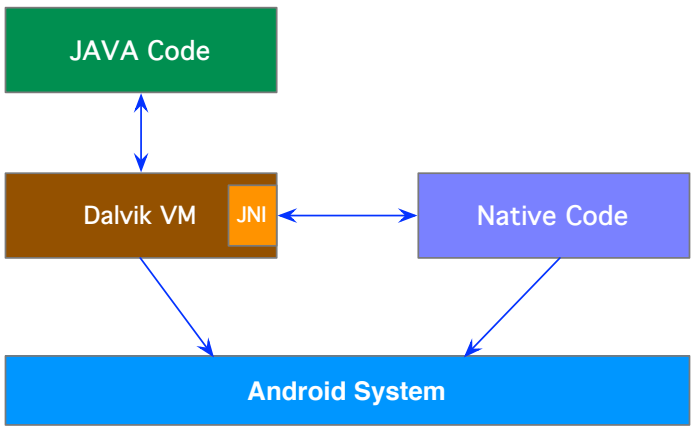


รูปที่ 6.47 ผลลัพธ์การรันโปรแกรม

พื้นฐานการพัฒนา JAVA NATIVE INTERFACE (JNI)

Java Native Interface หรือเรียกย่อๆว่า JNI เป็นส่วนหนึ่งที่อยู่ในชุดเครื่องมือพัฒนา JDK (Java Software Development Kit) ซึ่ง JNI นี้จะทำหน้าที่เป็นส่วนเชื่อมต่อระหว่างโปรแกรม native (C/C++) และโปรแกรม JAVA ทำให้โปรแกรม JAVA สามารถเรียกโค้ดโปรแกรม หรือฟังก์ชันของโปรแกรมที่ถูกพัฒนาด้วยโปรแกรมภาษา C/C++ หรือแม้แต่ภาษาอื่นๆได้ นอกจากนั้นโปรแกรมภาษาอื่นก็ยังสามารถเรียกฟังก์ชันภายในโปรแกรมภาษาจาวาได้เช่นกัน โดยเรียกผ่าน Invocation API ที่อยู่ภายใน JVM (Java virtual machine)

สำหรับระบบปฏิบัติการแอนดรอยด์โค้ดโปรแกรม JAVA (JAVA Code) ก็จะถูกรันบน Dalvik ซึ่งเป็นเครื่องเสมือน (Virtual Machine) คล้ายกับ (แต่ไม่เหมือน) JVM ของบริษัท Oracle แต่ตัว Dalvik นั้นได้ถูกพัฒนาใหม่ทั้งหมดโดยนักพัฒนาของกูเกิลเอง ในขณะที่โค้ดโปรแกรม native (Native Code) จะถูกคอมไพล์เป็นโค้ดไบนารี ซึ่งพร้อมถูกเรียกได้ทันทีในระดับระบบปฏิบัติการ ดังนั้น JNI จึงทำตัวเองเหมือนสะพานเชื่อมกันระหว่างโลกสองโลกนี้ ดังรูปแสดงความสัมพันธ์ระหว่าง JAVA code, Dalvik VM, native code, และ ระบบปฏิบัติการแอนดรอยด์ ข้างล่างนี้



รูปที่ 6.48 แสดงการสื่อสารระหว่างโปรแกรม Java และ Native C

จากรูปข้างบนทั้ง Dalvik VM และ Native Code ต่างก็ทำงานอยู่บนระบบปฏิบัติการแอนดรอยด์ ดังนั้น ระบบปฏิบัติการจะต้องเตรียมสภาพแวดล้อมสำหรับการเรียกคำสั่งระหว่างกัน (execution environment) และจะเห็นว่า JNI เป็นส่วนหนึ่งที่อยู่ใน Dalvik VM ซึ่งจะอำนวยความสะดวกให้ Native Code ให้สามารถเข้าถึงตัวแปรภายในคลาส (Data fields) และสมาชิกฟังก์ชันภายในคลาส (Member functions) ได้จากการกำหนดสิทธิ์การเข้าถึง (Access Permissions) คลาสภายในโค้ด JAVA

ดังนั้นการแลกเปลี่ยนข้อมูลระหว่างโปรแกรม JAVA และโปรแกรม native (C/C++) นั้นจึงเป็นประเด็นสำคัญที่นักพัฒนาควรทำความเข้าใจ ซึ่งใน JNI นั้นก็ได้มีการกำหนดตัวแปรที่สามารถรองรับตัวแปรมาตราฐานของแต่ละภาษา ดังแสดงในตารางข้างล่างนี้

ตาราง 6.3 แสดงประเภทของตัวแปรในภาษาจาวา เทียบกับตัวแปรใน JNI

Java Type	JNI	Java array Type	JNI
byte	jbyte	byte[]	jbyteArray
short	jshort	short[]	jshortArray
int	jint	int[]	jintArray
long	jlong	long[]	jlongArray
float	jfloat	float[]	jfloatArray
double	jdouble	double[]	jdoubleArray
char	jchar	char[]	jcharArray
boolean	jboolean	boolean[]	jbooleanArray
void	jvoid	Object[]	jobjectArray
Object	jobject		
Class	jclass		
String	jstring		
array	jarray		
Throwable	jthrowable		

นอกจากนั้น JNI ยังมีการกำหนดค่า (definition) ที่จำเป็นพื้นฐานไว้ให้เบื้องต้น เพื่อความสะดวกในการใช้งาน เช่น

```
#define JNI_FALSE  0
#define JNI_TRUE   1

typedef jint jsize;
```

เมื่อนักพัฒนาได้สร้างฟังก์ชันภายในโค้ดโปรแกรม native จะมีขบวนการหนึ่งที่สำคัญไม่น้อยไปกว่ากัน คือการลงทะเบียน native ฟังก์ชัน (Native Method Registration) ซึ่งโดยทั่วไป JNI เองจะสามารถค้นหาสำรวจชื่อฟังก์ชันภายในโปรแกรม native โดยอัตโนมัติอยู่แล้วดังตัวอย่างโปรแกรม HelloNDK ก่อนหน้านี้ แต่การพัฒนาโปรแกรมให้มีระบบและทำให้ JNI ไม่ต้องเสียเวลาเพื่อทำการค้นหาสำรวจชื่อฟังก์ชันทั้งหมดของโปรแกรม native ก่อนจะเริ่มทำการเรียกใช้ นักพัฒนาสามารถสร้างโปรแกรม native โดยให้ทำการลงทะเบียนชื่อฟังก์ชันภายในทั้งหมดให้กับ JNI ที่อยู่ภายใน Dalvik VM โดยการเรียกใช้ฟังก์ชันของ JNI ชื่อ **`RegisterNatives(JNIEnv *env, jclass clazz, const JNINativeMethod *methods, jint nMethods);`** ดังตัวอย่างข้างล่าง

```
...
JNINativeMethod nm[2];
nm[0].name = "NativePlus";
nm[0].signature = "(II)I";
nm[0].fnPtr = NativePlus;
nm[1].name = "NativeMultiply";
nm[1].signature = "(II)I";
nm[1].fnPtr = NativeMultiply;

jclass cls = (*env)->FindClass(env, "com/example/NativeApp/Activity");
// Register methods with env->RegisterNatives.
(*env)->RegisterNatives(env, cls, nm, 2);
```

อาร์กิวเม้นต์ cls (`clazz`) ในฟังก์ชัน **`RegisterNatives(..)`** จะถูกอ้างอิงไปยังคลาสที่เก็บรายการฟังก์ชันของโปรแกรม native ที่ได้ถูกลงทะเบียนแล้ว และตัวแปรแบบโครงสร้าง (data structure) nm (`JNINativeMethod`) จะเก็บรายละเอียดของฟังก์ชันในโปรแกรม native ที่ลงทะเบียนทั้งหมด โดยมีรายละเอียดดังนี้

```
typedef struct {
  char *name;          <--- ชื่อฟังก์ชัน (method name)
  char *signature;     <--- ตัวอธิบายคุณลักษณะของฟังก์ชัน (descriptor)
  void *fnPtr;         <--- ตัวชี้ไฟล์ยังตำแหน่งของฟังก์ชัน (function pointer)
} JNINativeMethod;
```

รูปแบบทั่วไปของการกำหนด method signature คือ

```
method-name(argument-types)return-type
```

JVM จะใช้อักขระเพื่อเป็นตัวแทนประเภทของตัวแปรในภาษาจาวา ดังแสดงในตารางข้างล่างนี้

ตาราง 6.4 อักขระเพื่อแทนสัญลักษณ์ให้กับตัวแปรของภาษาจาวา

JVM Signature	Java Type
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double
V	void
Lclass	ชื่อคลาส
[type	ชนิดตัวแปร
(arg-type)ret-type	ชนิดตัวแปรของอากิวเม้นต์ และการคืนค่ากลับ

ตัวอย่างการใช้งาน

ฟังก์ชัน	Signature
void fun1()	fun1()V
int fun2(int n, long l)	fun2(IJ)I
long fun3(int n, String s, int[] arr)	fun3(ILjava/lang/String;[I)J
boolean fun4(int[] arr)	fun4([I)Z
double fun5(String s, int n)	fun5(Ljava/lang/String;I)D
void fun6(int n, String[] arr, char c)	fun6(I[Ljava/lang/String;C)V
long fun7(int n, String s, int[] arr)	fun7(ILjava/lang/String;[I)J

ตัวอย่างการสร้างและเรียกใช้ JNI Methods

การพัฒนาโปรแกรม native (C/C++) โดยที่มีการสร้างฟังก์ชันแล้วทำการลงทะเบียนไปยัง Dalvik VM ด้วยฟังก์ชัน **RegisterNatives** ของ JNI เพื่อให้โปรแกรมประยุกต์ภาษาจาวา บนแอนดรอยด์สามารถเรียกใช้งานได้ มีขั้นตอนการพัฒนาดังนี้

ขั้นตอนที่ 1: สร้างโปรแกรม native และสร้างฟังก์ชันสำหรับการคำนวณทางคณิตศาสตร์

สร้างไดเรกทอรีชื่อ NativeCalculation และไดเรกทอรีย่อยชื่อ jni/ ซึ่งใช้ในการเก็บโค้ดโปรแกรม (.c, .h) และ Android.mk โดยใช้คำสั่งดังนี้

```
$ mkdir NativeCalculation
$ mkdir NativeCalculation/jni
$ cd NativeCalculation/jni
```

```
$ vim Android.mk
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE    := NativeCalculation
LOCAL_SRC_FILES := NativeCalculation.c
LOCAL_LDLIBS := -llog
include $(BUILD_SHARED_LIBRARY)
```

```
$ vim NativeCalculation.h
#include <jni.h>
#ifndef _Included_foo_Sqrt
#define _Included_foo_Sqrt
#ifdef __cplusplus
extern "C" {
#endif
#undef foo_Sqrt_EPSILON
#define foo_Sqrt_EPSILON 0.05

#ifdef __cplusplus
}
#endif
#endif
```

```
$ vim NativeCalculation.c
#include <jni.h>
#include <android/log.h>
#include <stdio.h>
#include <stdlib.h>
#include "NativeCalculation.h"

jint NativePlus(JNIEnv *pEnv, jobject pObj, jint pa, jint pb) {
      return pa+pb;
}

jint NativeMultiply(JNIEnv *pEnv, jobject pObj, jint pa, jint pb) {
      return pa*pb;
}

jdouble NativeSqrt(JNIEnv *pEnv, jobject pObj, jdouble pa) {
      jdouble x0 = 10.0, x1 = pa, diff;
      do {
```

```
            x1 = x0 - (((x0 * x0) - pa) / (x0 * 2));
            diff = x1 - x0;
            x0 = x1;
      } while (labs(diff) > 0.05);

      return x1;
}

jdoubleArray NativeArraySumAndAverage(JNIEnv *pEnv, jobject pObj, jintArray rets) {
      jboolean isCopy;
    // Convert rets to C's jint[]
    jint *intArr = (*pEnv)->GetIntArrayElements(pEnv, rets, &isCopy);
      __android_log_print(ANDROID_LOG_INFO, "NativeArraySumAndAverage", "a new copy
is created: %d", isCopy);
    if (NULL == intArr) return NULL;

      jsize len = (*pEnv)->GetArrayLength(pEnv, rets);
      int i;
      jint sum = 0;
      jdouble average = 0.0;

     for (i = 0; i < len; ++i) {
       __android_log_print(ANDROID_LOG_INFO, "NativeArraySumAndAverage of ", "%d =
%d", i, intArr[i]);
          sum += intArr[i];
      }
      average = (jdouble)sum / len;
      (*pEnv)->ReleaseIntArrayElements(pEnv, rets, intArr, 0);

      jdouble outCArray[] = {sum, average};


// Convert the C's Native jdouble[] to JNI jdoublearray, and then return
   // allocate array[2]
   jdoubleArray returnJNIArray = (*pEnv)->NewDoubleArray(pEnv, 2);

   if (NULL == returnJNIArray) return NULL;
   // copy outCArray to returnJNIArray
   (*pEnv)->SetDoubleArrayRegion(pEnv, returnJNIArray, 0 , 2, outCArray);

   return returnJNIArray;
}

JNIEXPORT jint JNICALL JNI_OnLoad(JavaVM* pVm, void* reserved)
{
    JNIEnv* env;
    if ((*pVm)->GetEnv(pVm, (void **)&env, JNI_VERSION_1_6)) {
             return -1;
      }
    JNINativeMethod nm[4];
    nm[0].name = "NativePlus";
    nm[0].signature = "(II)I";
```

```
    nm[0].fnPtr = NativePlus;
    nm[1].name = "NativeMultiply";
    nm[1].signature = "(II)I";
    nm[1].fnPtr = NativeMultiply;
    nm[2].name = "NativeSqrt";
    nm[2].signature = "(D)D";
    nm[2].fnPtr = NativeSqrt;
    nm[3].name = "NativeArraySumAndAverage";
    nm[3].signature = "([I)[D";
    nm[3].fnPtr = NativeArraySumAndAverage;

    jclass cls = (*env)->FindClass(env, "com/example/NativeCalculation/
MainActivity");
    // Register methods with env->RegisterNatives.
    (*env)->RegisterNatives(env, cls, nm, 4);
    return JNI_VERSION_1_6;
}
```

ขั้นตอนที่ 2: คอมไพล์โปรแกรม native และเตรียมไลบรารี เพื่อให้ใช้ได้กับ Android Studio

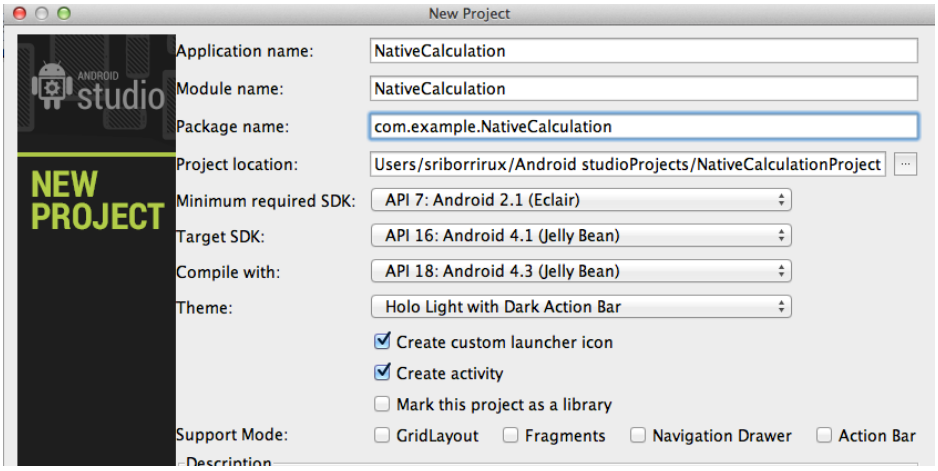
```
$ ndk-build
Compile thumb  : NativeCalculation <= NativeCalculation.c
SharedLibrary  : libNativeCalculation.so
Install        : libNativeCalculation.so =>
libs/armeabi/libNativeCalculation.so

$ ls
Android.mk      NativeCalculation.c   NativeCalculation.h
$ cd ..
$ mv libs lib
$ zip -r NativeCalculation.zip lib/
  adding: lib/ (stored 0%)
  adding: lib/android-support-v4.jar (deflated 14%)
  adding: lib/armeabi/ (stored 0%)
  adding: lib/armeabi/libNativeCalculation.so (deflated 47%)
$ mv NativeCalculation.zip NativeCalculation.jar
$ mn lib libs
$ mv NativeCalculation.jar libs/
```

สำหรับการสร้างโปรแกรมภาษาจาวา ด้วย Android Studio เพื่อเรียกใช้ไลบรารี `NativeCalculation` นั้น สามารถทำตามขั้นตอนดังนี้

ขั้นตอนที่ 1: สร้างโปรเจ็คใหม่ด้วย Android Studio

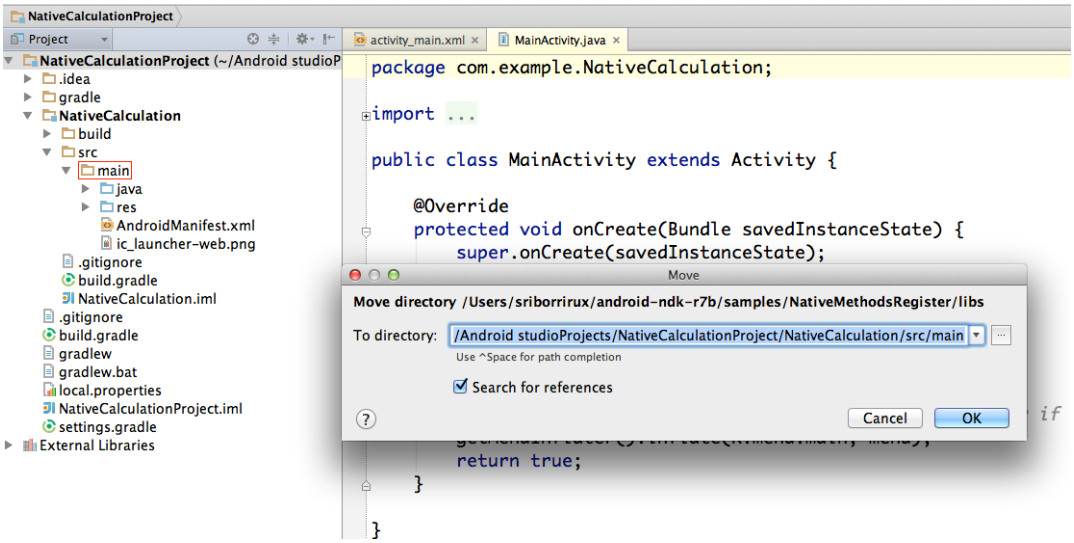
เลือกเมนู File -> New Project แล้วตั้งชื่อ Application name ว่า “NativeCalculation” และตั้งชื่อ Package name ว่า “com.example.NativeCalculation” ดังรูป

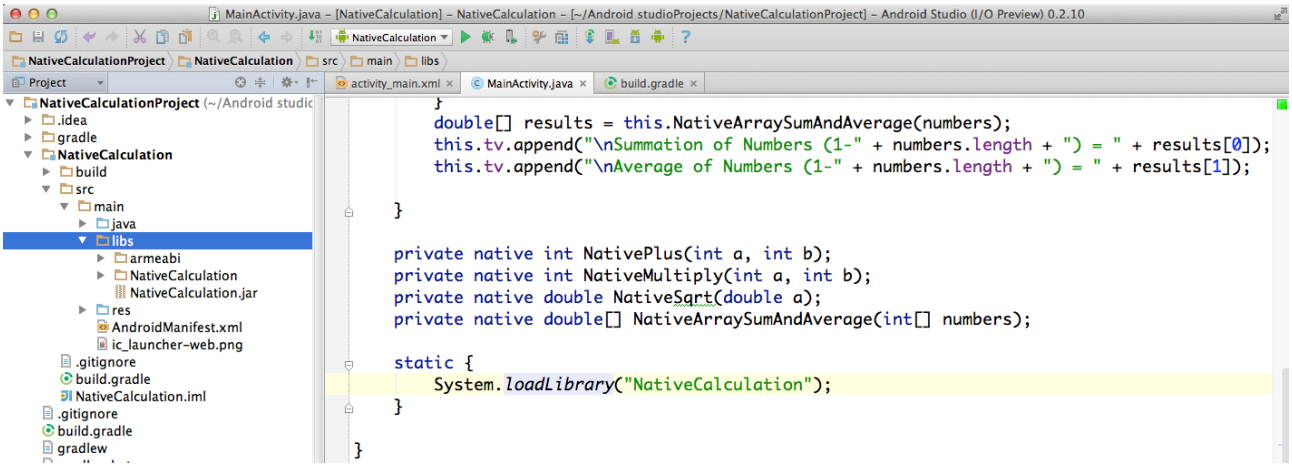


รูปที่ 6.49 การตั้งค่าโปรแกรม Calculation NDK

ขั้นตอนที่ 2: นำเข้าไลบรารี NativeCalculation มาวางไว้ในโปรเจ็ค

โดยการใช้เมาส์ลากไดเรกทอรี libs/ แล้วนำมาวางไว้ในโปรเจ็คภายใต้ไดเรกทอรี src/main ดังแสดงในรูปข้างล่าง





รูปที่ 6.50 การนำเข้าไดเรกทอรี lib และระบุรายละเอียดในไฟล์ MainActivity.java

ขั้นตอนที่ 3: แก้ไขไฟล์ build.gradle

ภายในโปรเจ็ค จะปรากฏไฟล์ชื่อว่า build.gradle ซึ่งเป็นไฟล์ใช้ในการตั้งค่าเพิ่มเติมสำหรับคอมไพล์ ให้เพิ่ม `task nativeLibsToJar, tasks.withType (Compile)` และ `dependencies` เข้าไปในไฟล์ build.gradle ให้เหมือนข้างล่างนี้

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:0.5.+'
    }
}
apply plugin: 'android'

repositories {
    mavenCentral()
}

android {
    compileSdkVersion 18
    buildToolsVersion "18.1.0"

    defaultConfig {
        minSdkVersion 7
        targetSdkVersion 16
    }
}

task nativeLibsToJar(
        type: Zip,
        description: 'create a jar archive of the native libs') {
    destinationDir file("src/main/libs/NativeCaculation")
    baseName 'NativeCaculation'
    extension 'jar'
    from fileTree(dir: 'src/main/libs', include: '**/*.so')
    into 'lib/'
}

tasks.withType (Compile) {
    compileTask -> compileTask.dependsOn (nativeLibsToJar)
}

dependencies {
    compile fileTree(dir: "src/main/libs/NativeCaculation", include:
'NativeCaculation.jar')
}
```

ขั้นตอนที่ 4: เขียนโปรแกรมภาษาจาวา เพื่อให้ทำงานตามที่ต้องการ

ให้แก้ไขไฟล์ MainActivity.java และไฟล์ activity_main.xml ให้มีรายละเอียดดังนี้

```
package com.example.NativeCalculation;
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.widget.TextView;

public class MainActivity extends Activity {
    private TextView tv;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        this.tv = new TextView(this);
        this.tv =(TextView)findViewById(R.id.calculation_text);
        callingNative();
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
    private void callingNative() {
        int a = 2342, b = 452;
        double c = 2353.0;
        int d = NativePlus(a, b);
        this.tv.setText(a + "+" + b + "=" + d);
        d = NativeMultiply(a, b);
        this.tv.append("\n" + a + "x" + b + "=" + d);
        double e = NativeSqrt(c);
        this.tv.append("\nSqrt(" + c + ")" + "=" + e);
        int numbers[] = new int[10000];
        for (int i=0; i < numbers.length ; ++i) {
            numbers[i] = i;
        }
        double[] results = this.NativeArraySumAndAverage(numbers);
        this.tv.append("\nSummation of Numbers (1-" + numbers.length + ") = " + re-
sults[0]);
        this.tv.append("\nAverage of Numbers (1-" + numbers.length + ") = " + re-
sults[1]);
    }
    private native int NativePlus(int a, int b);
    private native int NativeMultiply(int a, int b);
    private native double NativeSqrt(double a);
    private native double[] NativeArraySumAndAverage(int[] numbers);
    static {
        System.loadLibrary("NativeCalculation");
    }
}
```

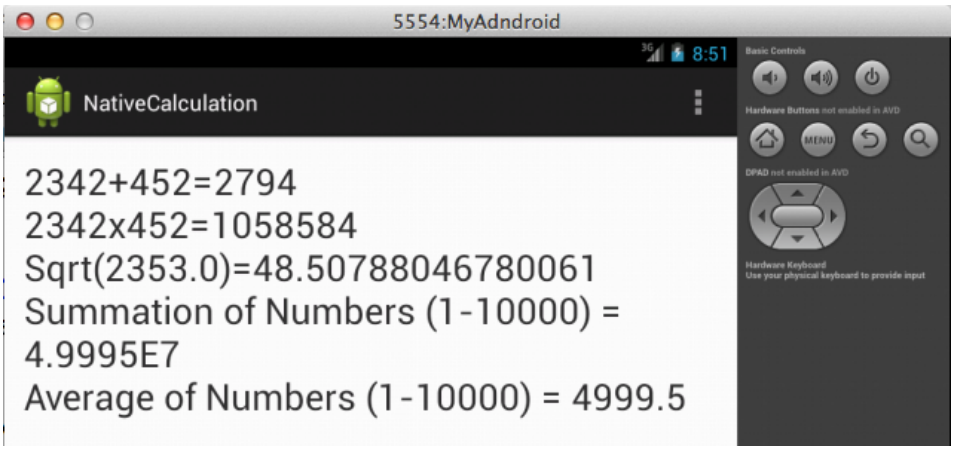
แก้ไขไฟล์ activity_main.xml โดยการเพิ่ม android:id="@+id/calculation_text" และ

```
        android:textSize="40dp"
        android:text="No Message..."
```

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">
    <TextView android:id="@+id/calculation_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="30dp"
        android:text="No Message..." />
</RelativeLayout>
```

ขั้นตอนที่ 5: รันโปรแกรมประยุกต์เพื่อเรียกใช้ไลบรารี

ผลการรันโปรแกรม NativeCalculation ซึ่งเป็นการเรียกใช้ฟังก์ชันในการบวกเลข คูณเลข หารากที่สอง บวกเลขในอาเรย์ และหาค่าเฉลี่ยของเลขในอาเรย์ แสดงผลลัพธ์ดังรูปข้างล่างนี้



รูปที่ 6.51 แสดงผลลัพธ์การรันโปรแกรมคำนวณทางคณิตศาสตร์

ตัวอย่างการพัฒนา ANDROID NDK MULTI-THREADING

เพื่อให้การทำงานในระดับ Native code สามารถดึงศักยภาพทางด้าน multi-processing technique ที่เป็นพื้นฐานสำคัญที่มากับระบบปฏิบัติการลีนุกซ์ตั้งแต่แรกด้วยการนำ Posix Thread ที่ทำให้โปรเซสสามารถแบ่งงานออกเป็นงานย่อยๆ (thread) แล้วแยกกันประมวลผล ซึ่งเรียกว่า multi-

threading programming เพื่อให้โปรแกรมประยุกต์ภาษาจาวา สามารถเรียกใช้งาน posix thread ได้ โดยตัวอย่างข้างล่างนี้จะแสดงให้เห็นถึงการใช้ mutex เพื่อทำให้เทรดแต่ละตัวทำงานได้สอดประสานไม่ขัดแย้งหรือแย้งใช้ทรัพยากรในเวลาเดียวกัน ดังตัวอย่างการพัฒนาในบทที่ 4 โดยการนำฟังก์ชัน pthread_mutex ที่เกี่ยวข้องมาใช้ ตัวอย่างเช่น

```
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

ซึ่งการเรียกใช้ mutex เพื่อให้เทรดนำไปเป็นเงื่อนไขในการขอเข้าใช้ทรัพยากรของระบบนั้น โดยมีฟังก์ชันให้เรียกใช้ได้ดังนี้

```
   int pthread_mutex_lock(pthread_mutex_t *mutex);
   int pthread_mutex_unlock(pthread_mutex_t *mutex);
   int pthread_mutex_trylock(pthread_mutex_t *mutex);
   int pthread_mutex_lock_timeout_np(pthread_mutex_t *mutex, unsigned msecs);
```

ขั้นตอนการพัฒนาโปรแกรม native และโปรแกรมประยุกต์ มีขั้นตอนดังนี้

ขั้นตอนที่ 1: สร้างโปรแกรม native และสร้างฟังก์ชันเทรด

สร้างไดเรกทอรีชื่อ NativeThreadMutex และไดเรกทอรีย่อยชื่อ jni/ ซึ่งใช้ในการเก็บโค้ดโปรแกรม (.c, .h) และ Android.mk โดยใช้คำสั่งดังนี้

```
$ mkdir NativeThreadsMutex
$ mkdir NativeThreadsMutex/jni
$ cd NativeThreadsMutex/jni
```

```
$ vim NativeThreadsMutex.cpp
#include <jni.h>
#include <unistd.h>
#include <pthread.h>
#include "mylog.h"

void jni_start_threads();
void *run_by_thread1(void *arg);
void *run_by_thread2(void *arg);

void jni_start_threads_dead();
void *run_by_thread1dead(void *arg);
void *run_by_thread2dead(void *arg);

jint JNI_OnLoad(JavaVM* pVm, void* reserved)
{
      JNIEnv* env;
      if (pVm->GetEnv((void **)&env, JNI_VERSION_1_6) != JNI_OK) {
```

```
                return -1;
        }
        JNINativeMethod nm[2];
        nm[0].name = "jni_start_threads";
        nm[0].signature = "()V";
        nm[0].fnPtr = (void*)jni_start_threads;
        nm[1].name = "jni_start_threads_dead";
        nm[1].signature = "()V";
        nm[1].fnPtr = (void*)jni_start_threads_dead;
        jclass cls = env->FindClass("com/example/NativeThreadsMutex/MainActivity");
        // Register methods with env->RegisterNatives.
        env->RegisterNatives(cls, nm, 2);
        return JNI_VERSION_1_6;
}

//pthread_mutex_t mux1 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mux1;
pthread_mutex_t mux2;
void jni_start_threads() {
        pthread_t th1, th2;
        int threadNum1 = 1, threadNum2 = 2;
        int ret;
        pthread_mutex_init(&mux1, NULL);
        pthread_mutex_init(&mux2, NULL);
        ret = pthread_create(&th1, NULL, run_by_thread1, (void*)&threadNum1);
        if(ret) {
                LOGE(1, "cannot create the thread %d thread: %d", threadNum1, ret);
        }
        LOGI(1, "thread 1 started");
        ret = pthread_create(&th2, NULL, run_by_thread2, (void*)&threadNum2);
        if(ret) {
                LOGE(1, "cannot create the thread %d: %d", threadNum2, ret);
        }
        LOGI(1, "thread 2 started");
        ret = pthread_join(th1, NULL);
        LOGI(1, "thread 1 end %d", ret);
        ret = pthread_join(th2, NULL);
        LOGI(1, "thread 2 end %d", ret);
        pthread_mutex_destroy(&mux1);
        pthread_mutex_destroy(&mux2);
}

int cnt = 0;
int THR = 10;
void *run_by_thread1(void *arg) {
        int* threadNum = (int*)arg;
        while (cnt < THR) {
                pthread_mutex_lock(&mux1);
                while ( pthread_mutex_trylock(&mux2) ) {
                   pthread_mutex_unlock(&mux1);  //avoid deadlock
                   usleep(50000);                    //if failed to get mux2, release
mux1 first
```

```
                pthread_mutex_lock(&mux1);
            }
            ++cnt;
            LOGI(1, "thread %d: cnt = %d", *threadNum, cnt);
            pthread_mutex_unlock(&mux1);
            pthread_mutex_unlock(&mux2);
            sleep(1);
      }
}

void *run_by_thread2(void *arg) {
      int* threadNum = (int*)arg;
      while (cnt < THR) {
            pthread_mutex_lock(&mux2);
            while ( pthread_mutex_trylock(&mux1) ) {
               pthread_mutex_unlock(&mux2);  //avoid deadlock
               usleep(50000);                    //if failed to get mux2, release
mux1 first
               pthread_mutex_lock(&mux2);
            }
            ++cnt;
            LOGI(1, "thread %d: cnt = %d", *threadNum, cnt);
            pthread_mutex_unlock(&mux2);
            pthread_mutex_unlock(&mux1);
            sleep(1);
      }
}

void jni_start_threads_dead() {
      pthread_t th1, th2;
      int threadNum1 = 1, threadNum2 = 2;
      int ret;
      pthread_mutex_init(&mux1, NULL);
      pthread_mutex_init(&mux2, NULL);
      ret = pthread_create(&th1, NULL, run_by_thread1dead, (void*)&threadNum1);
      if(ret) {
            LOGE(1, "cannot create the thread %d thread: %d", threadNum1, ret);
      }
      LOGI(1, "thread 1 started");
      ret = pthread_create(&th2, NULL, run_by_thread2dead, (void*)&threadNum2);
      if(ret) {
            LOGE(1, "cannot create the thread %d: %d", threadNum2, ret);
      }
      LOGI(1, "thread 2 started");
      ret = pthread_join(th1, NULL);
      LOGI(1, "thread 1 end %d", ret);
      ret = pthread_join(th2, NULL);
      LOGI(1, "thread 2 end %d", ret);
      pthread_mutex_destroy(&mux1);
      pthread_mutex_destroy(&mux2);
}
```

```
void *run_by_thread1dead(void *arg) {
	int* threadNum = (int*)arg;
	while (cnt < THR) {
		pthread_mutex_lock(&mux1);
		usleep(50);
		pthread_mutex_lock(&mux2);
		++cnt;
		LOGI(1, "thread %d: cnt = %d", *threadNum, cnt);
		pthread_mutex_unlock(&mux2);
		pthread_mutex_unlock(&mux1);
		sleep(1);
	}
}

void *run_by_thread2dead(void *arg) {
	int* threadNum = (int*)arg;
	while (cnt < THR) {
		pthread_mutex_lock(&mux2);
		usleep(50);
		pthread_mutex_lock(&mux1);
		++cnt;
		LOGI(1, "thread %d: cnt = %d", *threadNum, cnt);
		pthread_mutex_unlock(&mux1);
		pthread_mutex_unlock(&mux2);
		sleep(1);
	}
}
```

```
$ vim mylog.h

#ifndef COOKBOOK_LOG_H
#define COOKBOOK_LOG_H

#include <android/log.h>

#define LOG_LEVEL 4
#define LOG_TAG "NativeThreadsMutex"

#define LOGU(level, ...) if (level <= LOG_LEVEL) {__android_log_print(ANDROID_LOG_UNKNOWN,
LOG_TAG, __VA_ARGS__);}
#define LOGD(level, ...) if (level <= LOG_LEVEL) {__android_log_print(ANDROID_LOG_DEFAULT,
LOG_TAG, __VA_ARGS__);}
#define LOGV(level, ...) if (level <= LOG_LEVEL) {__android_log_print(ANDROID_LOG_VERBOSE,
LOG_TAG, __VA_ARGS__);}
#define LOGDE(level, ...) if (level <= LOG_LEVEL) {__android_log_print(ANDROID_LOG_DEBUG,
LOG_TAG, __VA_ARGS__);}
#define LOGI(level, ...) if (level <= LOG_LEVEL) {__android_log_print(ANDROID_LOG_INFO,
LOG_TAG, __VA_ARGS__);}
#define LOGW(level, ...) if (level <= LOG_LEVEL) {__android_log_print(ANDROID_LOG_WARN,
LOG_TAG, __VA_ARGS__);}
#define LOGE(level, ...) if (level <= LOG_LEVEL) {__android_log_print(ANDROID_LOG_ERROR,
LOG_TAG, __VA_ARGS__);}
#define LOGF(level, ...) if (level <= LOG_LEVEL) {__android_log_print(ANDROID_LOG_FATAL,
LOG_TAG, __VA_ARGS__);}
```

```
#define LOGS(level, ...) if (level <= LOG_LEVEL) {__android_log_print(ANDROID_LOG_SILENT,
LOG_TAG, __VA_ARGS__);}

#endif
```

```
$ vim Android.mk
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE    := NativeThreadsMutex
LOCAL_SRC_FILES := NativeThreadsMutex.cpp
LOCAL_LDLIBS    := -llog
include $(BUILD_SHARED_LIBRARY)
```

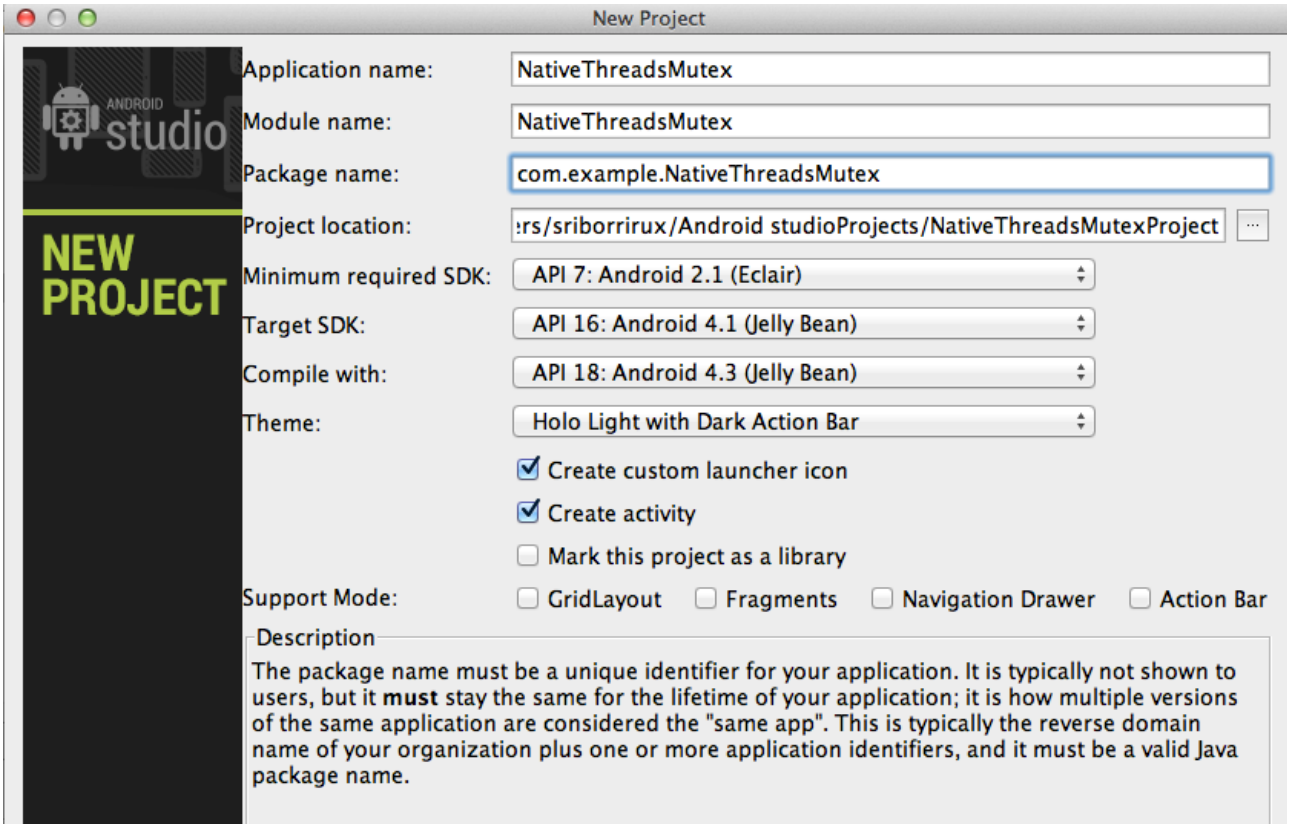
ขั้นตอนที่ 2: คอมไพล์โปรแกรม native และเตรียมไลบรารี เพื่อให้ใช้ได้กับ Android Studio

```
$ ndk-build
Compile++ thumb  : NativeThreadsMutex <= NativeThreadsMutex.cpp
SharedLibrary  : libNativeThreadsMutex.so
Install        : libNativeThreadsMutex.so =>
libs/armeabi/libNativeThreadsMutex.so
$ cd ..
$ mv libs lib
$ zip -r NativeThreadsMutex.zip lib/
  adding: lib/ (stored 0%)
  adding: lib/android-support-v4.jar (deflated 14%)
  adding: lib/armeabi/ (stored 0%)
  adding: lib/armeabi/libNativeThreadsMutex.so (deflated 49%)
$ mv NativeThreadsMutex.zip NativeThreadsMutex.jar
$ mv lib libs
$ mv NativeThreadsMutex.jar libs/
```

สำหรับการสร้างโปรแกรมภาษาจาวา ด้วย Android Studio เพื่อเรียกใช้ไลบรารี `NativeThreads-Mutex` นั้น สามารถทำตามขั้นตอนดังนี้

ขั้นตอนที่ 1: สร้างโปรเจ็คใหม่ด้วย Android Studio

เลือกเมนู File -> New Project แล้วตั้งชื่อ Application name ว่า “NativeThreadsMutex” และตั้งชื่อ Package name ว่า “com.example.NativeThreadsMutex”



รูปที่ 6.52 การตั้งค่าโปรแกรม NativeThreadsMutex NDK

ขั้นตอนที่ 2: ทำการรันโปรแกรม และตรวจสอบการทำงานผ่านโปรแกรม adb

เมื่อรันโปรแกรม ผลลัพธ์จะแสดงอยู่ใน logcat ดังนั้นเพื่อดูผลลัพธ์ จะใช้คำสั่ง adb logcat ดังแสดงข้างล่างนี้

```
$ adb logcat -v time NativeThreadsMutex:I *:S
10-05 19:19:45.469 I/NativeThreadsMutex( 2836): thread 1 started
10-05 19:19:45.469 I/NativeThreadsMutex( 2836): thread 2 started
10-05 19:19:45.479 I/NativeThreadsMutex( 2836): thread 2: cnt = 1
10-05 19:19:45.479 I/NativeThreadsMutex( 2836): thread 1: cnt = 2
10-05 19:19:46.488 I/NativeThreadsMutex( 2836): thread 2: cnt = 3
10-05 19:19:46.492 I/NativeThreadsMutex( 2836): thread 1: cnt = 4
10-05 19:19:47.489 I/NativeThreadsMutex( 2836): thread 2: cnt = 5
10-05 19:19:47.499 I/NativeThreadsMutex( 2836): thread 1: cnt = 6
10-05 19:19:48.490 I/NativeThreadsMutex( 2836): thread 2: cnt = 7
10-05 19:19:48.499 I/NativeThreadsMutex( 2836): thread 1: cnt = 8
10-05 19:19:49.489 I/NativeThreadsMutex( 2836): thread 2: cnt = 9
10-05 19:19:49.499 I/NativeThreadsMutex( 2836): thread 1: cnt = 10
10-05 19:19:50.499 I/NativeThreadsMutex( 2836): thread 1 end 0
10-05 19:19:50.499 I/NativeThreadsMutex( 2836): thread 2 end 0
```

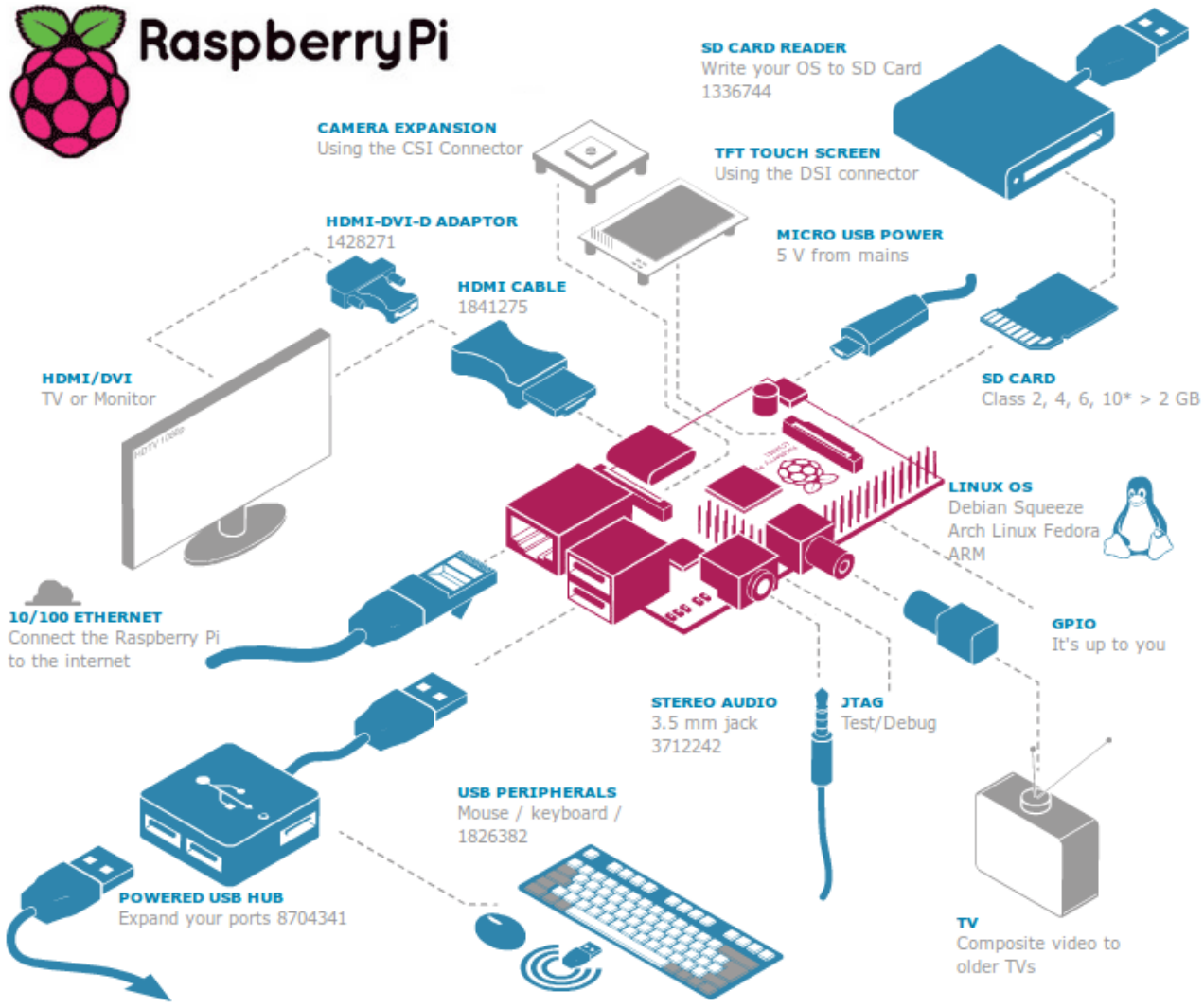
บทที่ 7 การพัฒนาโปรแกรมเพื่องานประยุกต์บนระบบสมองกลฝังตัว

ตัวอย่างการพัฒนาโปรแกรมบนบอร์ด Raspberry Pi

ในบทนี้จะแสดงตัวอย่างการประยุกต์ทางด้านระบบสมองกลฝังตัว โดยเนื้อหาในส่วนแรกจะเป็นตัวอย่างการพัฒนาโปรแกรมสำหรับบอร์ดสมองกลฝังตัวซึ่งเป็นบอร์ดที่ได้รับความนิยมมากในปัจจุบันชื่อว่าบอร์ด Raspberry Pi ซึ่งเป็นบอร์ดสมองกลฝังตัวที่ทำหน้าที่เหมือนคอมพิวเตอร์ขนาดเล็ก (Single-board computer) ที่พัฒนาโดยนาย Eben Upton ซึ่งทำงานอยู่บริษัท Broadcom และเป็นนักพัฒนาหลักภายใต้องค์กรไม่แสวงหากำไรชื่อว่า Raspberry Pi Foundation ในประเทศอังกฤษ โดยมีวัตถุประสงค์หลักคือต้องการสร้างบอร์ดตัวนี้ขึ้นมาเพื่อส่งเสริมการเรียนการสอนทางด้านวิทยาศาตร์และคอมพิวเตอร์ในโรงเรียน

ซึ่งความตั้งใจเดิมของโครงการ Raspberry Pi คือเพื่อต้องการเพียงจะทำเป็นบอร์ดไมโครคอนโทรลเลอร์ที่มีความสามารถคล้ายกับบอร์ด Arduino แต่ในที่สุดก็ได้ตัดสินใจออกแบบพัฒนาให้เป็นระดับบอร์ดคอมพิวเตอร์ขนาดเล็กแทน เพื่อต้องการให้ครูและนักเรียนสามารถพัฒนาโปรแกรมภายในบอร์ด Raspberry Pi ได้ทันที โดยไม่ต้องใช้เครื่องคอมพิวเตอร์เป็นเครื่อง Host สำหรับพัฒนาโปรแกรมเหมือนบอร์ดสมองกลฝังตัวทั่วไป ทางองค์กร Raspberry Pi ได้ทำข้อตกลงความร่วมมือทางลิขสิทธ์ร่วมกับบริษัท Element14 และบริษัท RS Components ซึ่งเป็นบริษัทยักษ์ใหญ่ด้านการสั่งซื้ออุปกรณ์อิเล็กทรอนิกส์แบบออนไลน์ เพื่อให้เป็นผู้ผลิตและผู้แทนจำหน่ายบอร์ด Raspberry Pi
รายละเอียดทางด้านเทคนิคดังนี้

1. ใช้ System-on-Chip จากค่าย Broadcom รุ่น BCM2835 ซึ่งมีหน่วยประมวลผลกลาง ARM1176JZF-S ความเร็วสัญญาณนาฬิกา 700 MHz สามารถถูก Over Clock ได้ถึง 1 GHz
2. หน่วยความจำ SDRAM ขนาด 256 MB ในรุ่น A และ SDRAM 512 MB ในรุ่น B
3. สามารถเก็บข้อมูลและบูต Image file ระบบปฏิบัติการได้จาก SD Card
4. มีUSB 2.0 ให้ใช้งาน
5. มีช่องเสียบ Video แบบ Composite RCA และ HDMI
6. มีหัวแจ๊ก Audio ขนาด 3.5 มิลลิเมตร
7. มี Ethernet 10/100
8. มี GPIO 17 พินให้ใช้งาน
9. มีแหล่งจ่ายไฟเลี้ยง +3.3 V และ +5V
10. ขนาดของบอร์ด 85.60 มิลลิเมตร x 53.98 มิลลิเมตร
11. น้ำหนัก 45 กรัม



รูปที่ 7.1 แสดงความสามารถในการเชื่อมต่อกับอุปกรณ์ภายนอกต่างๆกับบอร์ด Raspberry Pi

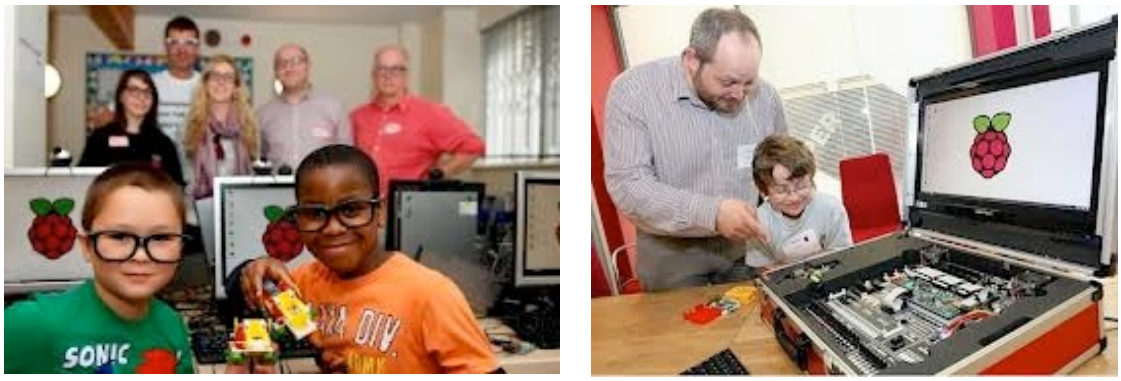
บอร์ด Raspberry Pi สามารถเชื่อมต่อกับอุปกรณ์ต่างได้หลายหลากดังแสดงในรูปข้างบน ซึ่งเหมาะสำหรับนักพัฒนาระบบสมองกลฝังตัวที่ไม่ต้องการเครื่อง Host เพื่อใช้ในการพัฒนาโปรแกรมแต่จะสามารถพัฒนาโปรแกรมด้วยโปรแกรม Arduino IDE เพื่อใช้ในการโปรแกรมภาษา C เพื่อเป็นโปรแกรมเฟิร์มแวร์ (firmware) สำหรับไมโครคอนโทรเลอร์ ATMEL ที่ใช้ในบอร์ด Arduino หรือโปรแกรมภาษาอื่นๆ เช่น ภาษาPython ด้วยโปรแกรม Python IDE ภาษาเพิร์ล (Perl) ภาษาเบสิก (Basic) หรือ ภาษา C++ เป็นต้น

บอร์ด Raspberry Pi ยังรองรับระบบปฏิบัติการที่เป็นที่นิยมอยู่ในปัจจุบัน ตัวอย่างเช่น ระบบปฏิบัติการ Debian GNU/Linux, Fedora, FreeBSD, NetBSD, Plan 9, Raspbian OS, RISC OS และ Slackware Linux นอกจากนั้นบอร์ดนี้ก็ยังเหมาะกับการนำไปประยุกต์ทำเป็นเครื่อง Media server หรือ Set-top-box สำหรับงานทางด้านมัลติมีเดีย ดังตัวอย่างโปรเจค OpenELEC (Open Embedded

Linux Entertainment Center) ที่มีการนำระบบปฏิบัติการลีนุกซ์ มาปรับแต่งใส่โปรแกรม XBMC media center แล้วนำมาติดตั้งในบอร์ด Raspberry Pi

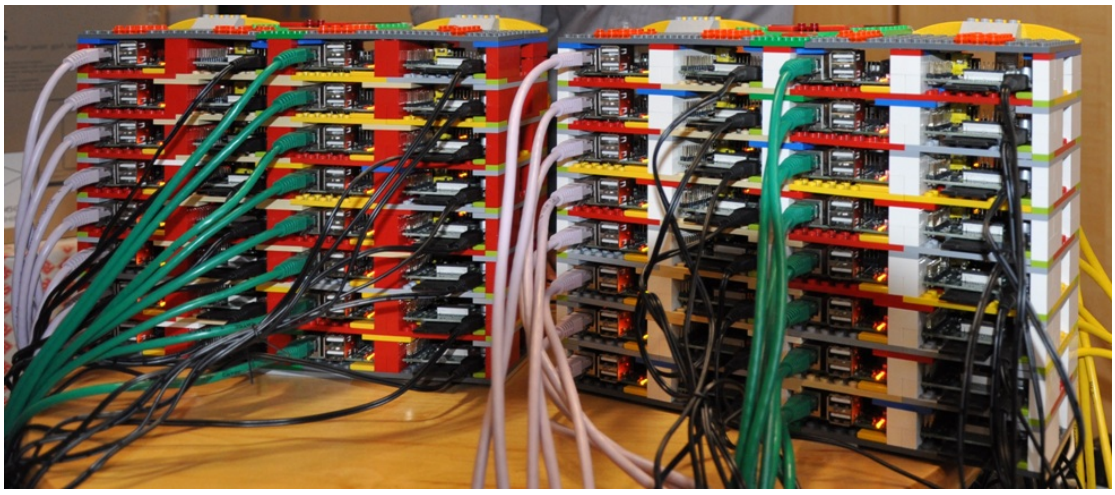
การประยุกต์ใช้งานบอร์ด RaspBerry Pi

ในด้านการศึกษา บอร์ด Raspberry Pi สามารถใช้เป็นบอร์ดสร้างแรงบันดาลใจให้กับเด็ก หรือนักศึกษา ที่มีความสนใจในการนำไปสร้างเป็นของเล่น หรือใช้ในการควบคุมอุปกรณ์ภายนอกต่างได้



รูปที่ 7.2 การนำ Raspberry Pi ไปใช้ในการสอนภายในโรงเรียน

สำหรับการประยุกต์ในงานด้านอุตสาหกรรม บอร์ด Raspberry Pi เองก็ได้รับความนิยมไม่น้อยจากเหล่านักพัฒนา เนื่องจากมีความยืดหยุ่นในการเลือกใช้ภาษาโปรแกรมที่จะนำมาพัฒนาโปรแกรมลงภายในบอร์ดและสามารถติดตั้งไลบรารีพิเศษเพิ่มเติมลงไปได้ ตัวอย่างเช่น บอร์ด Data Acquisition (DAQ), Mini Super Computer ดังรูปข้างล่าง

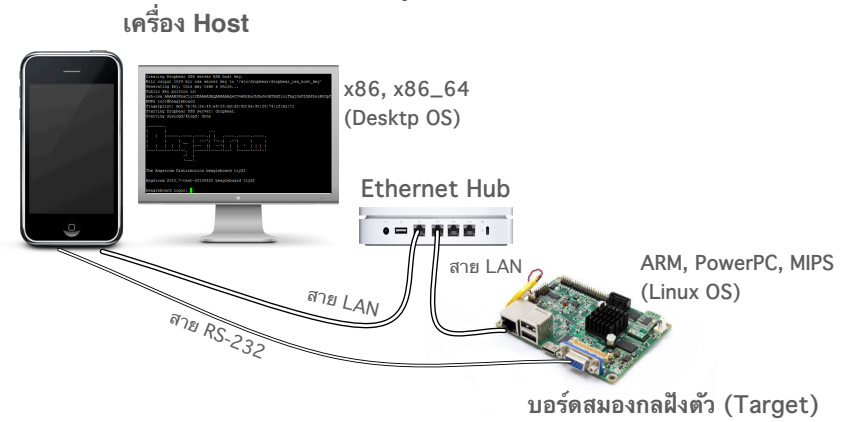


รูปที่ 7.3 การนำ Raspberry Pi ไปประยุกต์ในการประมวลผลขนาน

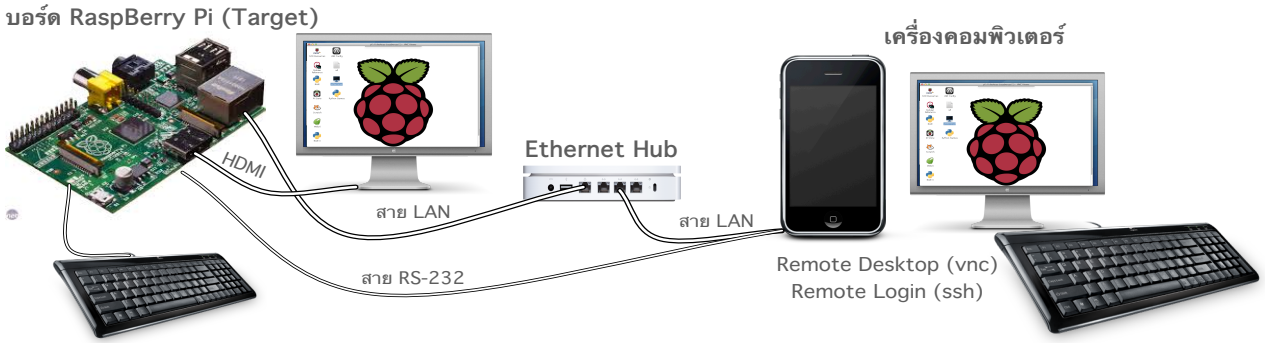
แนวทางการพัฒนาบนบอร์ด Raspberry Pi

ตั้งแต่ในอดีตจนถึงปัจจุบันมีบอร์ดสมองกลฝังตัวสำหรับการเรียนรู้และเป็นบอร์ดประยุกต์ (Evaluation Kit) ออกมาจำนวนมากมาย ซึ่งส่วนใหญ่จะต้องมีการเตรียมเครื่องคอมพิวเตอร์ (Host) ไว้อีกเครื่องสำหรับติดตั้ง เครื่องมือพัฒนา cross-compiling toolchains และชุดโปรแกรมสำหรับใช้ในการพัฒนาภาษาโปรแกรมต่างๆ นอกจากนั้นจะต้องเตรียมสายเคเบิลแบบสายอนุกรม (Serial cable) เพื่อเชื่อมต่อกับบอร์ดสมองกลฝังตัวผ่านพอร์ทคอนโซล (console) เพื่อให้สามารถเข้าถึงระบบปฏิบัติการภายในบอร์ดผ่านหน้าจอเทอร์มินัลและสามารถพิมพ์ชุดคำสั่งการตั้งค่าต่างๆ หรือเพื่อดีบักโค้ดโปรแกรมได้

แต่ด้วยวิธีการดังที่กล่าวมาข้างต้นก็ยังสร้างความยุ่งยากไม่น้อยให้กับนักพัฒนามือใหม่ทั้งหลาย แต่เมื่อมาถึงยุคการมาของบอร์ด Raspberry Pi ทำให้นักพัฒนาสามารถพัฒนาโปรแกรมและทำการคอมไพล์โปรแกรมอยู่บนบอร์ดสมองกลได้ทันที โดยเพียงนำบอร์ด Raspberry Pi มาต่อกับจอมอนิเตอร์ คีย์บอร์ด และเมาส์ เท่านั้น นอกจากนั้นนักพัฒนาก็ยังสามารถทำการ remote หน้าจอ (Remote Desktop) ผ่านเครื่องคอมพิวเตอร์ (Host) โดยใช้โปรแกรม VNC (หรือ ssh ในกรณี text mode) เข้าไปยังบอร์ด Raspberry Pi ได้เช่นกัน ดังแสดงในรูปข้างล่าง



รูปที่ 7.4 การเชื่อมต่ออุปกรณ์สำหรับการพัฒนาระบบสมองกลฝังตัวแบบทั่วไป

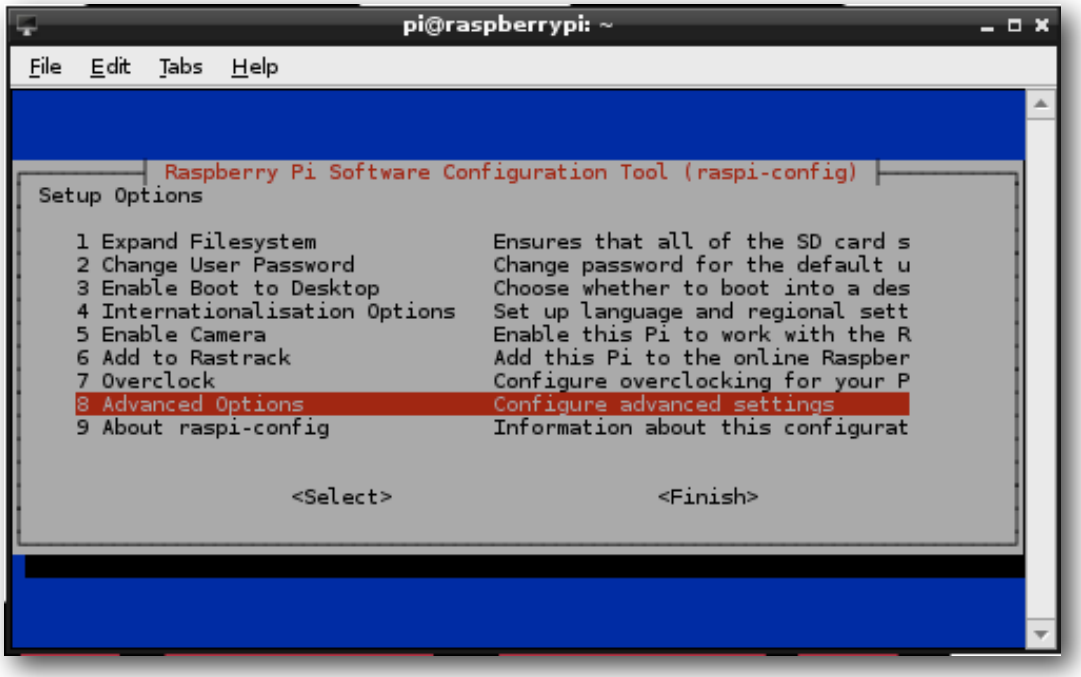


รูปที่ 7.5 การเชื่อมต่ออุปกรณ์สำหรับการพัฒนาระบบสมองกลฝังตัวของ Raspberry Pi

การเข้าสู่ระบบปฏิบัติการภายใน Raspberry Pi ผ่านโปรแกรม Secure Shell (ssh)

การติดตั้งและเปิดการใช้งาน ssh (secure shell) เป็นการสร้างการเชื่อมต่อ command line ระยะไกล ผ่านทางโปรโตคอล ssh ระหว่างเครื่องคอมพิวเตอร์ และบอร์ด Raspberry Pi วิธีการติดตั้งมีดังนี้

```
$ sudo raspi-config
```

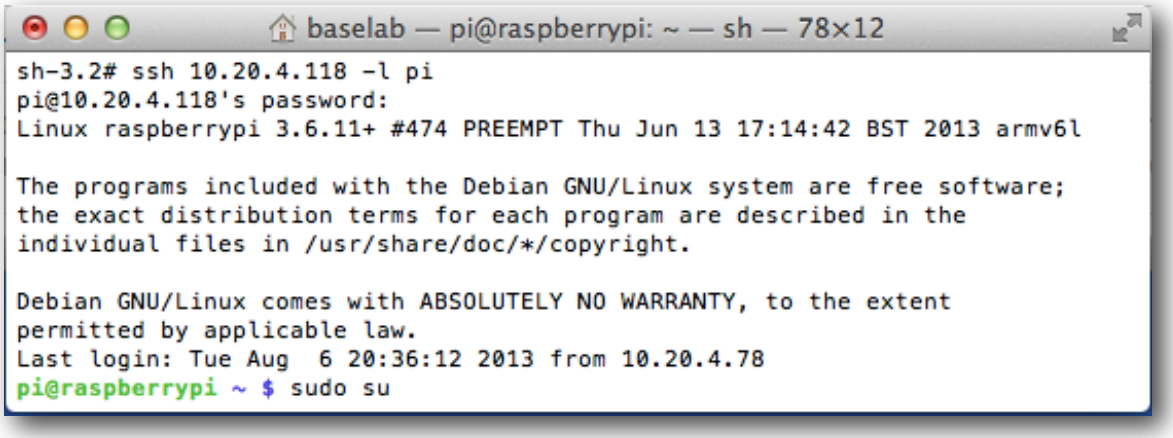


รูปที่ 7.6 แสดงการตั้งค่าของ Raspberry Pi ผ่านคำสั่ง raspi-config

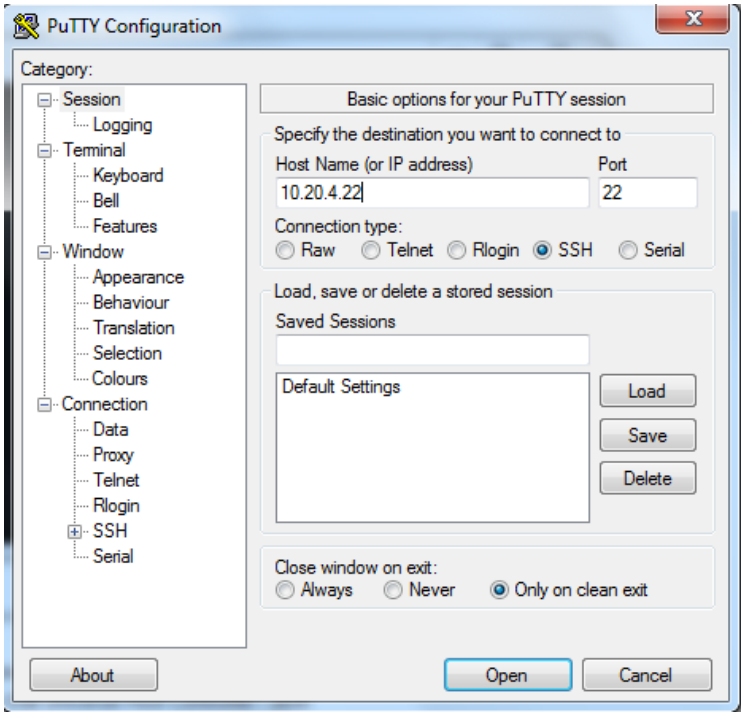
แล้วเลือกเมนู SSH แล้วตั้งค่าให้เป็น ENABLE ถัดไป

ทดสอบเปิดการเชื่อมต่อระยะไกลจากเครื่องอื่นผ่าน ssh client ไปยังบอร์ด Raspberry Pi (IP Address: 10.20.4.222 ด้วยชื่อผู้ใช้ pi)

(จากเครื่อง Mac)$ **ssh 10.20.4.222 -l pi**

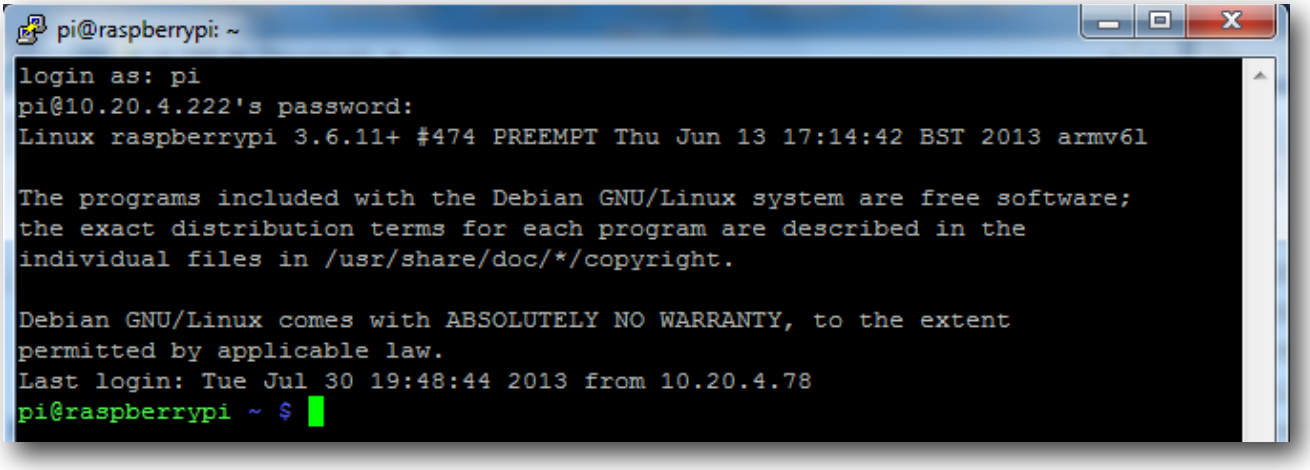


เปิดการเชื่อมต่อระยะไกลจากเครื่องอื่นผ่าน ssh client ไปยังบอร์ด Raspberry Pi ด้วยโปรแกรม Putty บนระบบปฏิบัติการ Windows



รูปที่ 7.7 หน้าตาโปรแกรม PuTTY ในส่วนหน้าแสดงการตั้งค่าต่างๆ

ทำการใส่ไอพีแอดเดลสของบอร์ด Raspberry PI แล้วเลือก Open จากนั้นจะเข้าไปที่หน้า terminal ให้ทำการ login : pi และ password : raspberry ต่อจากนั้นก็จะเข้าสู่หน้า terminal ของ Raspbian โดยสมบูรณ์



รูปที่ 7.8 หน้าต่างโปรแกรม Terminal ในระหว่างการล็อกอินเข้าสู่ระบบ

การเข้าสู่ระบบปฏิบัติการภายใน Raspberry Pi ผ่านโปรแกรม VNC (Remote Desktop)

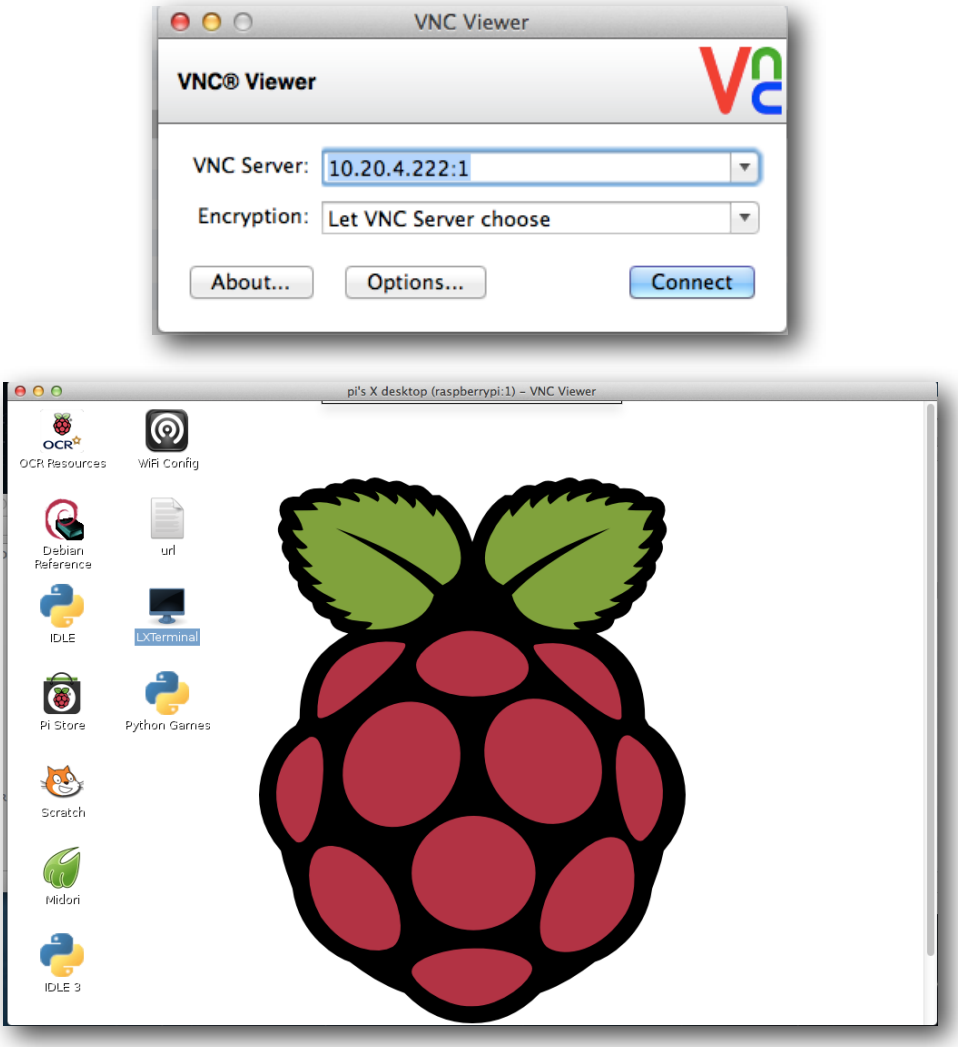
เริ่มต้นด้วยการติดตั้งโปรแกรม VNC server ชื่อว่า tightvncserver ลงเข้าไปในบอร์ด Raspberry Pi ด้วยคำสั่งดังนี้

```
$ sudo apt-get install tightvncserver
```

ทำการเปิดบริการโปรแกรม VNC server พร้อมระบุขนาดของหน้าจอ Raspberry Pi

```
$ vncserver :1 -geometry 1024x786 -depth 16 -pixelformat rgb565
```

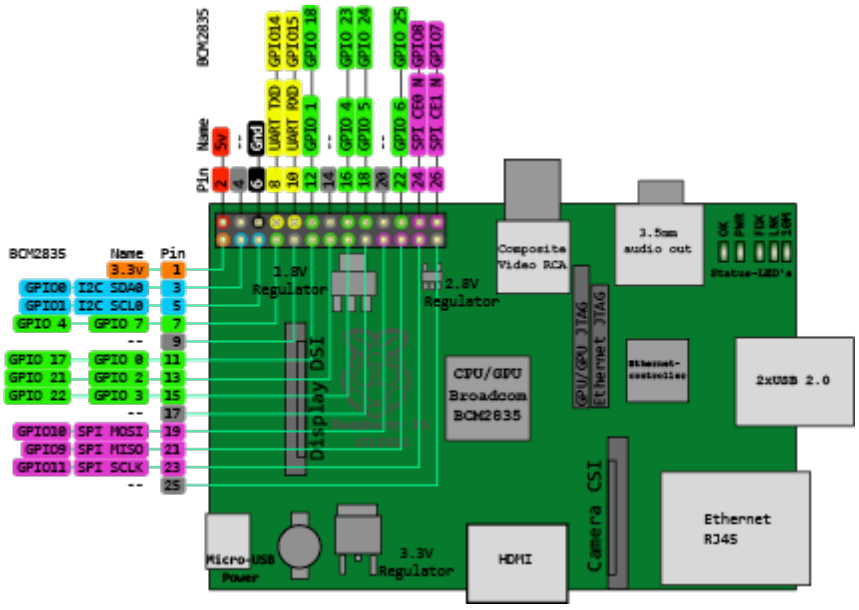
สำหรับเครื่องคอมพิวเตอร์ ให้ทำการติดตั้งโปรแกรม VNC viewer (รองรับหลายระบบปฏิบัติการ) แล้วระบุ IP ของเครื่อง Raspberry Pi ดังแสดงในรูปข้างล่างนี้



รูปที่ 7.9 แสดงการ Remote Desktop เข้าไปยังบอร์ด Raspberry Pi

GPIOs ภายในบอร์ด Raspberry PI

GPIO เป็นหนึ่งในคุณสมบัติพิเศษที่บอร์ด Raspberry Pi เปิดให้นักพัฒนาสามารถใช้งานได้ แม้ว่าคุณสมบัติบางอย่างมีไม่ครบเหมือนบอร์ดไมโครคอนโทรลเลอร์ทั่วไปแต่ก็เพียงพอที่จะสร้างโปรแกรมประยุกต์ได้มากมาย เพราะตัวบอร์ด Raspberry Pi เองนั้นด้วยคุณสมบัติที่สามารถเขียนโปรแกรมได้บนบอร์ดเหมือนเครื่องคอมพิวเตอร์ Host เมื่อบวกกับการให้สามารถเข้าถึง GPIOs ต่างๆได้ง่าย ก็ยิ่งทำให้นักพัฒนามือใหม่จนถึงระดับกลางสะดวกและง่ายต่อการติดตั้งโปรแกรมหรือเขียนโปรแกรมเป็นอย่างดี จึงทำให้บอร์ด Raspberry Pi เป็นอีกหนึ่งบอร์ดที่ได้รับความนิยมเป็นอย่างสูง



รูปที่ 7.10 ตำแหน่ง GPIO บนบอร์ด Raspberry Pi

โดยทั่วไป GPIO (General purpose I/O) ของบอร์ด Raspberry Pi จะอยู่ตรงส่วนขา (ขา) P1 ซึ่งมีจำนวนของขา 2x13 หรือ 26 ขา โดยขาทั้งหมดไม่ได้มีแค่ GPIO เพียงอย่างเดียว แต่ประกอบไปด้วย SPI, I2C, serial UART, ไฟเลี้ยง 3.3 V, ไฟเลี้ยง 5 V และ GND

<u>รายละเอียด GPIO ของบอร์ด Raspberry Pi</u>

-P1 มีจำนวน 26 ขา มีระยะห่างระหว่างขา 2.54 มม. (100 mil)

-เอาพุตท์ของแต่ละขา จะอยู่ในช่วง 0-3.3V และอินพุตไม่ควรมากกว่า 3.3 V ถ้ามีอินพุตที่เป็น 5 V ควรผ่านวงจรปรับแรงดันให้เป็น 3.3V ก่อนที่เข้ามาที่ขาอินพุต

-ไฟเลี้ยง 3.3 V สามารถจ่ายกระแสได้ไม่เกิน 50 mA

-ไฟเลี้ยง 5V สามารถจ่ายกระแสได้ไม่เกิน 100 mA

-มีดิจิตอล อินพุต/เอาพุต 8 ขา

-มี I2C 2 ขา

-มี SPI 5 ขา

-มี UART 2 ขา
-มีขาที่ยังไม่ใช้อีกอย่างน้อย 6 ขา

พื้นฐานการพัฒนาโปรแกรมภาษาPython (Python)

การใช้งานและพัฒนา GPIO บนบอร์ด Raspberry Pi ส่วนใหญ่เป็นการสั่งค่าให้ GPIO ที่อยู่บนบอร์ดสามารถใช้งานได้ ไม่ว่าจะเป็นการสั่งงานจากเทอร์มินัล หรือจะเป็นการเขียนโค้ดสั่งงาน โดยที่ส่วนใหญ่ระบบปฏิบัติการที่อยู่บนบอร์ด Raspberry Pi อย่างเช่น ระบบปฏิบัติการ Raspbian ได้เตรียมเครื่องมือ Python IDE พร้อมทั้ง Python Cross complier และ C/C++ Cross complier ไว้ให้แล้ว ซึ่งนักพัฒนาสามารถใช้งานได้ทันทีโดยไม่ต้องติดตั้งโปรแกรม toolchains เพิ่มเติม

ตัวอย่างการสั่งเอาพุตขา GPIO ผ่าน command line

เข้าสู่ระบบปฏิบัติการภายใน Raspberry Pi ในสิทธิ์ super user เพื่อสามารถเข้าถึงขา GPIO ได้

```
$ sudo su
```

export GPIO ขา จากไฟล์ `/sys/class/gpio/export`

```
# echo “4” > /sys/class/gpio/export
```

กำหนดชนิดของ GPIO4 ให้เป็นแบบเอาพุท

```
# echo “out” > /sys/class/gpio/gpio4/direction
```

สามารถสั่งให้ GPIO4 ติดหรือดับได้ ด้วยคำสั่ง

```
# echo “1” > /sys/class/gpio/gpio4/value    <--- (สั่งให้ LED ติด)
```

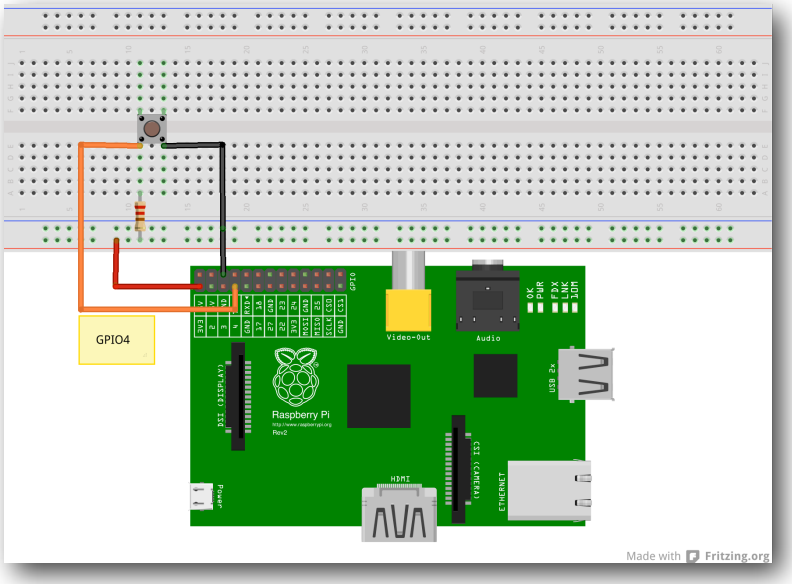
หรือ

```
# echo “0” > /sys/class/gpio/gpio4/value    <--- (สั่งให้ LED ดับ)
```

เมื่อต้องการยกเลิกใช้งาน GPIO ให้ใช้คำสั่ง unexport ไฟล์ `/sys/class/gpio/unexport`

```
# echo “4” > /sys/class/gpio/unexport
```

ตัวอย่างการรับค่าจากขา GPIO ที่เป็น Push button ผ่าน command line



รูปที่ 7.11 การต่อวงจรเพื่อรับการกดสวิทซ์กับบอร์ด Raspberry Pi

เข้าสู่ระบบปฏิบัติการภายใน Raspberry Pi ในสิทธิ์ super user เพื่อสามารถเข้าถึงขา GPIO ได้

```
$ sudo su
```

export GPIO ขา จากไฟล์ `/sys/class/gpio/export`

```
# echo "4" > /sys/class/gpio/export
```

กำหนดชนิดของ GPIO4 ให้เป็นแบบเอาพุท

```
# echo "in" > /sys/class/gpio/gpio4/direction
```

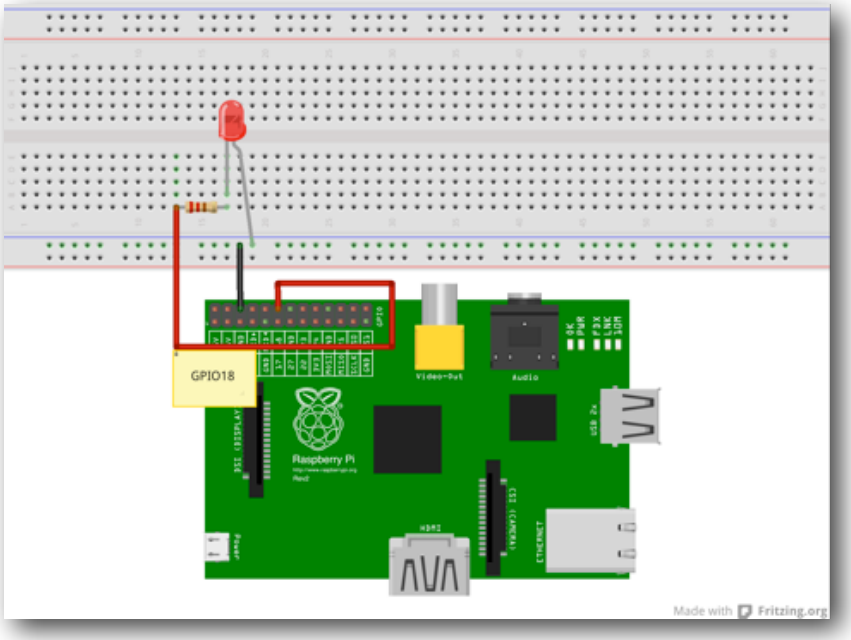
อ่านค่าของอินพุท ด้วยคำสั่ง cat

```
# cat /sys/class/gpio/gpio4/value
```

เมื่อต้องการยกเลิกใช้งาน GPIO ให้ใช้คำสั่ง unexport ไฟล์ `/sys/class/gpio/unexport`

```
# echo "4" > /sys/class/gpio/unexport
```

ตัวอย่างการสั่งเอาพุตขา GPIO เพื่อขับ LED



รูปที่ 7.12 การต่อวงจรเพื่อควบคุม LED ด้วยบอร์ด Raspberry Pi

สร้างไฟล์ **gpio_out_LED.py** โดยมีรายละเอียดโค้ดดังนี้

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BOARD)
GPIO.setup(12,GPIO.OUT)
while (1):
     GPIO.output(12,GPIO.HIGH)
     time.sleep(1)
     GPIO.output(12,GPIO.LOW)
     time.sleep(1)
```

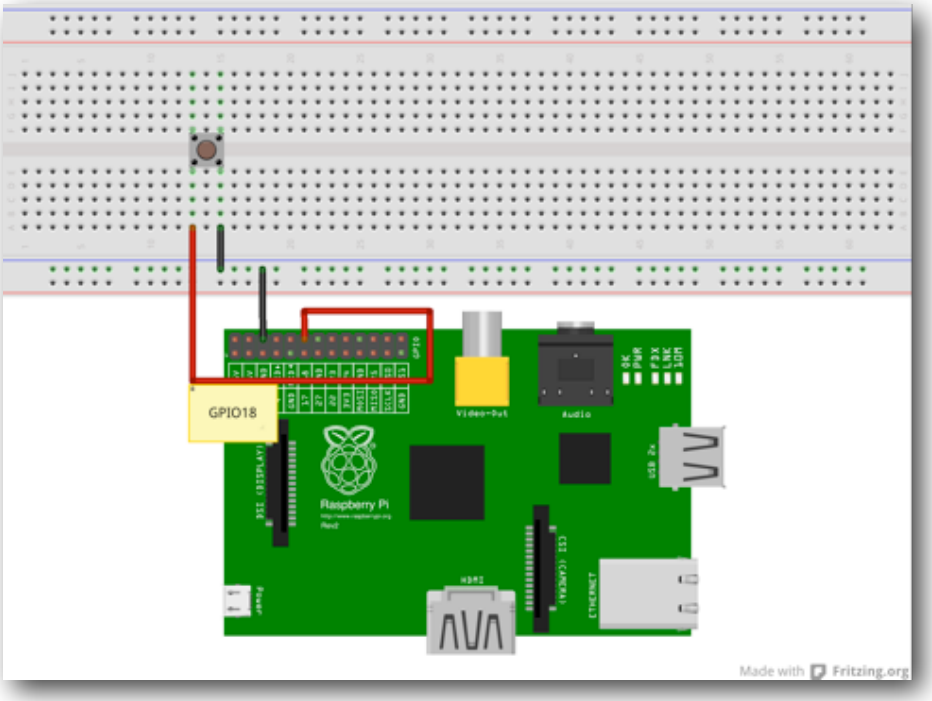
ในการตั้งค่ารูปแบบการนับหมายเลขขานั้น มีด้วยกัน 2 แบบคือ

แบบที่ 1: GPIO.BOARD เป็นการอ้างอิงหมายเลขขา GPIO ที่อยู่บนบอร์ด Raspberry Pi

แบบที่ 2: GPIO.BCM เป็นการอ้างอิงหมายเลขขา GPIO หน่วยประมวลผล Broadcom BCM2835 CPU

ตัวอย่างเช่น GPIO.BOARD เท่ากับตำแหน่งขาหมายเลข 12 แต่ถ้าเป็น GPIO.BCM จะเป็นตำแหน่งขาหมายเลข 18

ตัวอย่างการสั่งเอาพุตขา GPIO โดยการรับจากการกดปุ่ม push button



รูปที่ 7.13 การต่อวงจรเพื่อรับการกดสวิทช์ผ่านขา 18

สร้างไฟล์ **gpio_button.py** โดยมีรายละเอียดโค้ดดังนี้

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.IN, pull_up_down=GPIO.PUD_UP)
channel = 12
def check_button():
     if(GPIO.input(channel) == GPIO.LOW):
          print ('Button Pressed.')
while (1):
     check_button()
     time.sleep(1)
```

จากรูปด้านบน ถ้าผู้ใช้งานไม่ต้องการที่จะต่อตัวต้านทานเพื่อทำการ pull-up หรือ pull-down สามารถเพิ่มพารามิเตอร์ pull_up_down ใน GPIO.setup ได้ เช่น

```
GPIO.setup(12,GPIO.IN, pull_up_down=GPIO.PUD_UP) (ต่อตัวต้านทาน pull-up )
```

หรือ

```
GPIO.setup(12,GPIO.IN, pull_up_down=GPIO.PUD_DOWN) (ต่อตัวต้านทาน pull-down )
```

การตั้งค่าการตรวจจับการกดปุ่ม มีอยู่ด้วยกัน 2 สถานะ คือ วงจรอินพุตแบบศักย์ต่ำ (Active Low) และวงจรอินพุตแบบศักย์สูง (Active High) ซึ่งสามารถเขียนโปรแกรม python เพื่อตรวจจับได้ ดังนี้

```
if(GPIO.input(channel) == GPIO.LOW):  (ต่อวงจรอินพุตแบบศักย์ต่ำ - Active low)
   print ('Button Pressed.')
```

หรือ

```
if(GPIO.input(channel) == GPIO.HIGH):  (ต่อวงจรอินพุตแบบศักย์สูง - Active high)
          print ('Button Pressed.')
```

ตัวอย่างโค้ดโปรแกรม **gpio_button.py** ข้างต้นจะเป็นการอ่านค่าสถานะการกดปุ่มโดยการวนรอบเข้ามาอ่านค่าสถานะทุกๆ 1 วินาทีด้วยเทคนิคโพลลิ่ง (polling technique) แต่อย่างไรก็ตามอาจจะไม่เหมาะกับการนำไปใช้งานจริง เนื่องจากอาจมีงานอย่างอื่นที่โปรแกรมต้องทำด้วย ไม่เพียงแต่วนรอบเพื่ออ่านสถานะของการกดปุ่มเพียงอย่างเดียว ดังนั้นการนำเทคนิคอินเตอร์รัพท์ (Interrupt technique) ที่นิยมเขียนกันบนไมโครคอลโทรเลอร์ทั่วไป จึงเหมาะสมมากกว่าแบบเทคนิคโพลลิ่ง

ซึ่งภาษา Python สามารถสร้างฟังก์ชันในลักษณะ threaded callback เพื่อตรวจจับสัญญาณของอินพุต โดยการเขียนโค้ดโปรแกรมดังตัวอย่างข้างล่าง

```
def my_callback(channel):
 print('Button on press...')
 GPIO.add_event_detect(12, GPIO.FALLING, callback=my_callback, bounce-
time=200)
```

ตัวอย่างโค้ดโปรแกรมตัวใหม่

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.IN, pull_up_down=GPIO.PUD_UP)
channel = 12
def my_callback(channel):
   print('Button on press...')
   GPIO.add_event_detect(12, GPIO.FALLING, callback=my_callback, bounceti-
me=200)

while (1):
```

ตัวอย่างการประยุกต์บอร์ด Raspberry Pi ให้เป็น Web Server

นอกจากเขียนโค้ดโปรแกรมหรือใช้คำสั่งเพื่อเข้าถึง GPIO ของบอร์ด Raspberry Pi แล้ว นักพัฒนายังสามารถสั่งงาน GPIO ผ่านหน้าบราวเซอร์ (Web Browser) ซึ่งจะสะดวกต่อผู้ใช้งานเนื่องจากสามารถสั่งงานหรือเฝ้าติดตามดูสถานะของอุปกรณ์ที่ต่อกับ GPIO ของบอร์ด Raspberry Pi จากระยะไกลได้ ตัวอย่างการนำไปประยุกต์เช่น การนำบอร์ด Raspberry Pi เชื่อมต่อกับอุปกรณ์ไฟฟ้าภายในบ้าน แล้วสามารถสั่งงานเครื่องใช้ไฟฟ้าหรือดูสถานะเครื่องใช้ไฟฟ้าผ่านบราวเซอร์จากที่ทำงาน เป็นต้น

ตัวอย่างการติดตั้ง Web server ลงบอร์ด Raspberry Pi

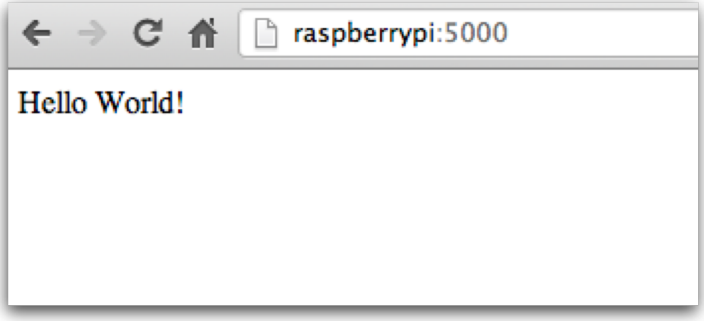
ติดตั้งโปรแกรมจัดการแพ็กเก็ต (pip) และ โปรแกรม Flask ซึ่งเป็น web application framework ขนาดเล็กที่ถูกพัฒนาด้วยภาษา Python รวมกับ Werkzeug WSGI toolkit และ Jinja2 engine

```
$ sudo apt-get install python-pip
$ sudo pip install Flask
```

จากนั้นทดลองเขียนโปรแกรม Hello world

```
$ mkdir hello
$ cd hello/
$ nano hello.py
   from flask import Flask
        app = Flask(__name__)
     @app.route("/")
     def  hello():
          return "Hello world!"
     if  __name__ == "__main__":
     app.run('0.0.0.0')
$ python hello.py
   * Running on http://0.0.0.0:5000/
```

จากนั้นเปิดหน้าบราวเซอร์บนเครื่องคอมพิวเตอร์ แล้วพิมพ์ raspberrypi:5000 ช่อง URL



รูปที่ 7.14 ผลลัพธ์การเรียกไฟล์ hello.py ผ่านหน้าบราวเซอร์ ที่พอร์ตหมายเลข 5000

ตัวอย่างการเพิ่มไฟล์ HTML และ CSS เข้าไปในโปรแกรม hello.py

สร้างไดเรกทอรีชื่อ “templates/” อยู่ภายในไดเรกทอรี hello/ เพื่อเก็บไฟล์ HTML ชื่อว่า home.html เนื่องจากฟังก์ชัน home() นั้นจะค้นหาไฟล์ home.html ที่อยู่ภายในไดเรกทอรี templates/ นอกจากนั้นถ้าต้องการใช้ CSS ก็ต้องสร้างไดเรกทอรีเพิ่มเติมชื่อว่า static/css/ ไว้ภายในไดเรกทอรี hello/ เพื่อเก็บไฟล์ CSS

```
$ cd hello/
$ mkdir templates
$ mkdir static
$ mkdir static/css
$ cd static/css
```

```
$ nano main.css
body {
  margin: 0;
  padding: 0;
  font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
  color: #444;
}
/*
 * Create dark grey header with a white logo
 */
header {
  background-color: #2B2B2B;
  height: 35px;
  width: 100%;
  opacity: .9;
  margin-bottom: 10px;
}
header h1.logo {
  margin: 0;
  font-size: 1.7em;
  color: #fff;
  text-transform: uppercase;
  float: left;
}
header h1.logo:hover {
  color: #fff;
  text-decoration: none;
}
/*
 * Center the body content
 */
.container {
  width: 940px;
  margin: 0 auto;
}
```

```
div.jumbo {
  padding: 10px 0 30px 0;
  background-color: #eeeeee;
  -webkit-border-radius: 6px;
     -moz-border-radius: 6px;
          border-radius: 6px;
}
h2 {
  font-size: 3em;
  margin-top: 40px;
  text-align: center;
  letter-spacing: -2px;
}
h3 {
  font-size: 1.7em;
  font-weight: 100;
  margin-top: 30px;
  text-align: center;
  letter-spacing: -1px;
  color: #999;
}
```

สร้างไฟล์ layout.html เพื่อกำหนดรูปแบบการแสดงของหน้าเว็ป

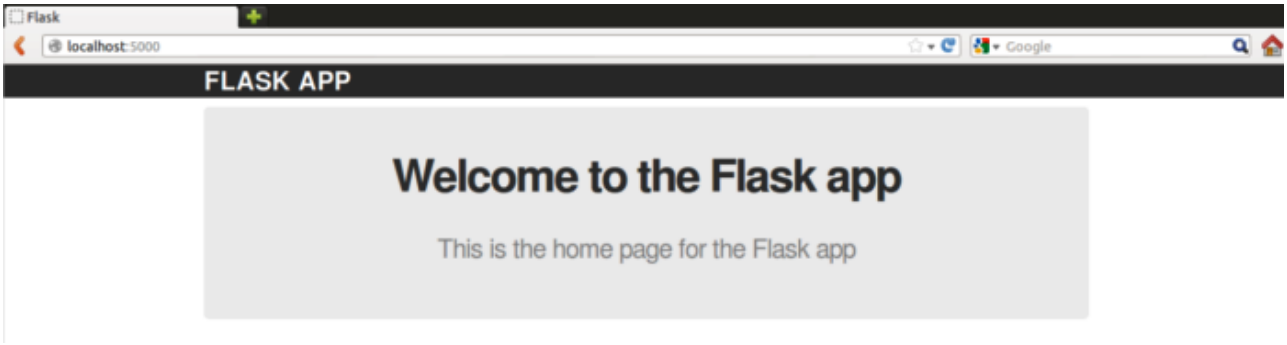
```
$ nano layout.html
<!DOCTYPE html>
<html>
  <head>
    <title>Flask</title>
    <strong><link rel="stylesheet" href="{{ url_for('static',
filename='css/main.css') }}"></strong>
  </head>
  <body>
    <header>
      <div class="container">
        <h1 class="logo">Flask App</h1>
      </div>
     </header>

    <div class="container">
      {% block content %}
      {% endblock %}
    </div>
  </body>
</html>
```

แก้ไขไฟล์ home.html เพื่อเรียกใช้รูปแบบในไฟล์ layout.html

```
$ nano home.html
{% extends "layout.html" %}
{% block content %}
  <div class="jumbo">
    <h2>Welcome to the Embedded System<h2>
    <h3>This is the home page for the Home Automation<h3>
  </div>
{% endblock %}
```

ขั้นตอนสุดท้าย ทำการทดสอบเปิดหน้าบราวเซอร์ที่ตำแหน่ง localhost:5000 อีกครั้ง

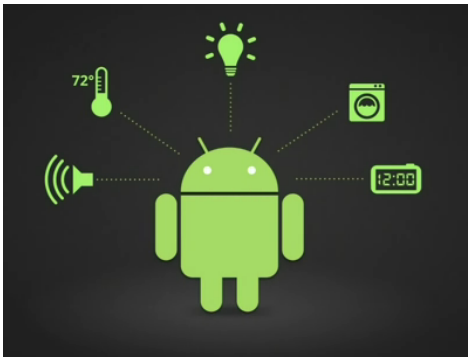


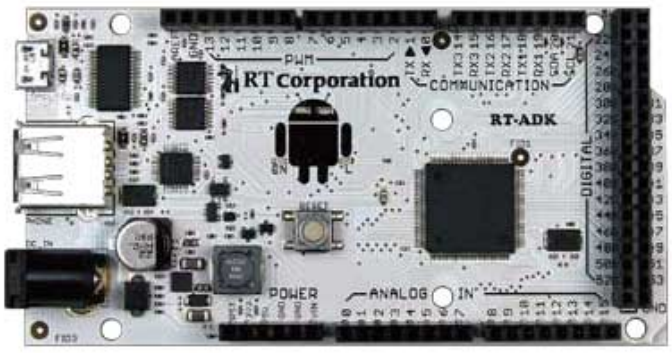
รูปที่ 7.15 ผลลัพธ์การเข้าหน้าบราวเซอร์ ที่มีการกำหนด CSS Style ไว้

เครื่องมือพัฒนาพื้นฐานสำหรับ Android และ Arduino

Accessory Development Kit (ADK) คือการพัฒนาต่อยอดของฮาร์แวร์สำหรับนักพัฒนามือสมัครเล่น ที่ใช้เป็นจุดเริ่มต้นในการเรียนรู้ในการพัฒนาโปรแกรมสร้างสรรค์ต่างๆ ให้สามารถเชื่อมต่อกับอุปกรณ์เสริมสำหรับเครื่องโทรศัพท์และแท็ปเล็ตที่มีระบบปฏิบัติการแอนดรอยด์ได้ นับเป็นจุดเริ่มต้นสำคัญในวงการอุปกรณ์ชนิดพกพาที่สามารถจะเกิดการสร้างสรรค์สิ่งใหม่มากยิ่งขึ้น

ซึ่งภายในงาน Google I/O 2011 ทางบริษัทกูเกิลก็ได้เปิดตัว ADK รุ่นแรก และต่อมาบริษัทในญี่ปุ่นชื่อ RT Corporation ได้พัฒนาบอร์ด RT-ADK ซึ่งถือว่าเป็นบอร์ด Arduino (http://www.arduino.cc) ที่ถูกผลิตออกมา โดยภายในใช้ไมโครคอนโทรลเลอร์ ATMega 2560 พร้อมกับชุดเซ็นเซอร์ต่างๆ เพื่อรองรับเทคโนโลยีนี้โดยเฉพาะ นอกจากนั้นจุดเด่นของ Arduino คือได้ถูกออกแบบให้สามารถขยายการเชื่อมต่อกับบอร์ดโมดูลอื่นๆ ที่เรียกว่าบอร์ด Shield เช่นบอร์ด Ethernet Shield บอร์ด WIFI Shield บอร์ด Bluetooth Shield เป็นต้น





รูปที่ 7.16 สามารถในการเชื่อมต่ออุปกรณ์ภายนอกผ่าน ADK และบอร์ด RT-ADK รุ่นแรก

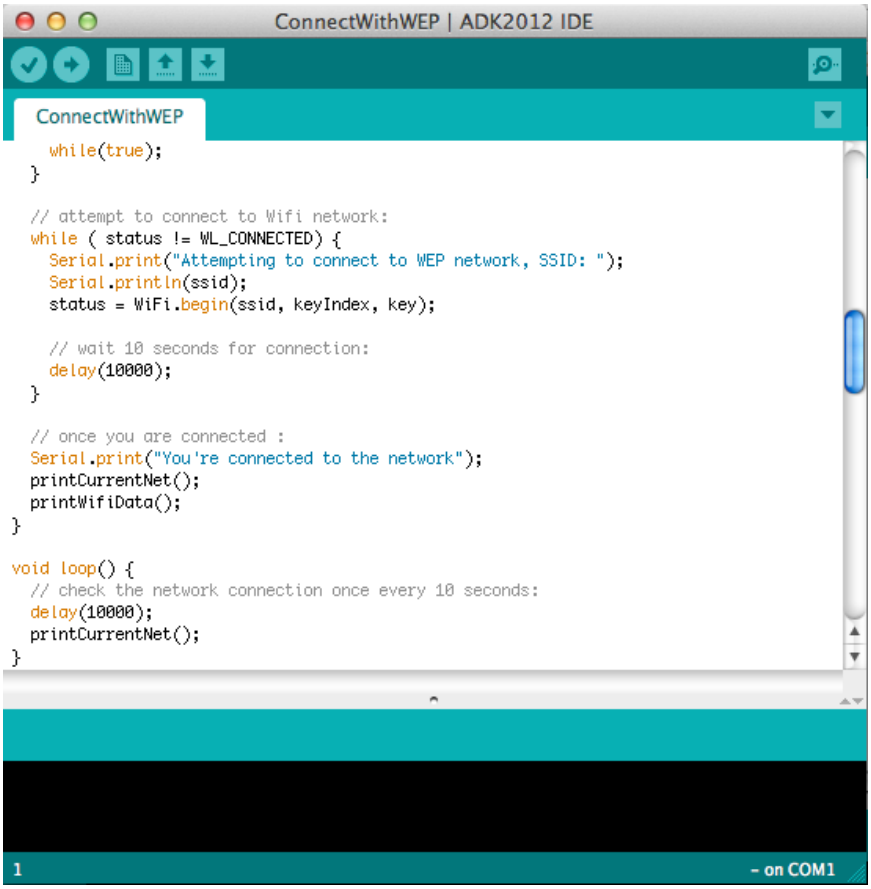
รูปแบบในการติดต่อสื่อสารจะเป็นแบบ Android Open Accessory (AOA Protocol) เพื่อทำการติดต่อสื่อสารกันได้กับอุปกรณ์แอนดรอยด์ (Android Devices) ผ่านสายการเชื่อมต่อ USB

Android Open Accessory

Android Open Accessory ได้ถูกพัฒนาขึ้นมาเพื่อให้อุปกรณ์แอนดรอยด์สามารถเชื่อมต่อกับอุปกรณ์ภายนอกผ่านทาง USB ไปยัง USB Host ของอุปกรณ์ภายนอก แต่อุปกรณ์ภายนอกจะไม่สามารถใช้พลังงานไฟฟ้าจากอุปกรณ์แอนดรอยด์ได้ ดังนั้นอุปกรณ์ภายนอกจะต้องมีแหล่งจ่ายพลังงานไฟฟ้าของตัวเองและต้องมีระดับกระแสไม่น้อยกว่า 500mA ที่ความต่างศักย์ไฟฟ้า 5V เพื่อให้สามารถสื่อสารกับอุปกรณ์แอนดรอยด์ได้ สำหรับบอร์ด Arduino นั้นนักพัฒนาสามารถเขียนโปรแกรมโดยใช้โปรแกรม Arduino IDE (www.arduino.cc) หรือโปรแกรม ADK2012 IDE (บริษัทกูเกิล)

โปรแกรม ADK2012 IDE

หลักจากดาวน์โหลดโปรแกรม ADK2012 IDE จาก Android ADK แล้วทำการติดตั้งลงเครื่องคอมพิวเตอร์เรียบร้อย หน้าตาของโปรแกรมจะคล้าย Arduino IDE แต่จะมีส่วนของไลบารีทางด้าน UsbAccessory ที่กูเกิลได้พัฒนาเพิ่มเติมเข้ามาดังแสดงในรูปข้างล่าง ในกรณีที่นักพัฒนาต้องการลงไลบารีเพิ่มเติมก็ทำได้โดยการแตกไฟล์ไลบารีแล้วคัดลอกไปไว้ในไดเรกทอรีชื่อว่า libraries



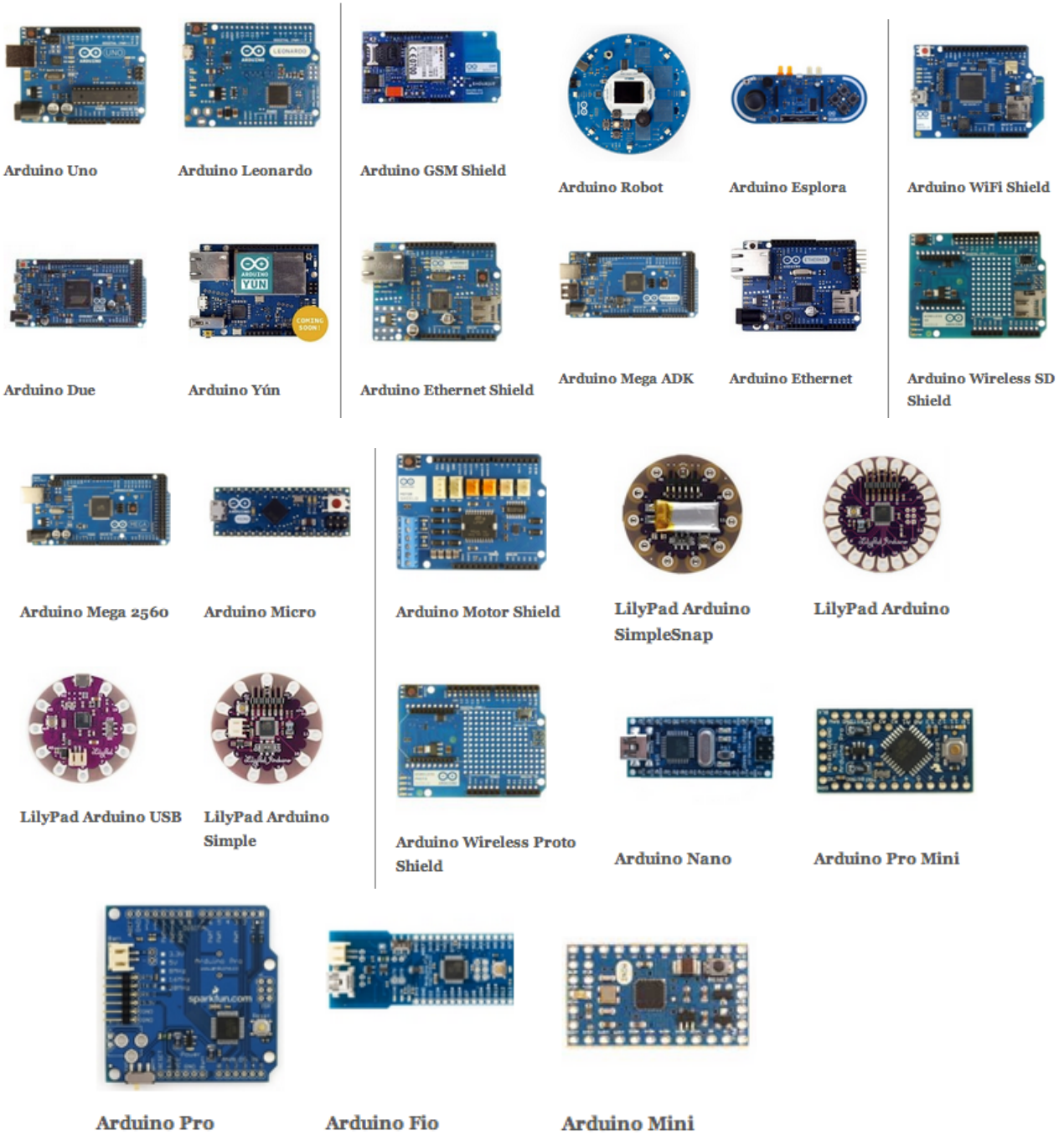
รูปที่ 7.17 หน้าตาโปรแกรม ADK2012 IDE

ปัจจุบันได้มีการพัฒนาบอร์ด Arduino ออกมาด้วยกันหลายรุ่นดังแสดงในตารางข้างล่าง ดังนั้นนักพัฒนาควรทำความเข้าใจรายละเอียดความสามารถของบอร์ดแต่ละรุ่นและพัฒนาโปรแกรมเพื่อดึงเอาความสามารถมาใช้งานได้อย่างเต็มที่เพื่อให้เหมาะสมกับงานประยุกต์

ตาราง 7.1 แสดงรุ่นต่างๆของ Arduino

รุ่นบอร์ด Arduino	
Arduino Uno	Arduino BT w/ ATmega328
Arduino Duemilanove w/ ATmega328	Arduino BT w/ ATmega168
Arduino Diecimila or Duemilanove w/ ATmega 168	LilyPad Arduino USB
Arduino Nano w/ ATmega328	LilyPad Arduino w/ ATmega328
Arduino Nano w/ ATmega168	LilyPad Arduino w/ ATmega168
Arduino Mega 2560 or Mega ADK	Arduino Pro or Pro Mini (5V, 16MHz) w/ ATmega328
Arduino Mega (ATmega1280)	Arduino Pro or Pro Mini (5V, 16MHz) w/ ATmega168
Arduino Leonardo	Arduino Pro or Pro Mini (3.3V, 8MHz) w/ ATmega328
Arduino Esplora	Arduino Pro or Pro Mini (3.3V, 8MHz) w/ ATmega168
Arduino Micro	Arduino NG or older w/ ATmega168
Arduino Mini w/ ATmega328	Arduino NG or older w/ ATmega8

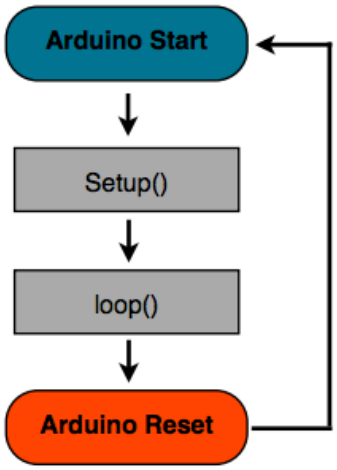
Arduino Mini w/ ATmega168	Arduino Robot Control
Arduino Ethernet	Arduino Robot Motor
Arduino Fio	



รูปที่ 7.18 ตัวอย่างบอร์ด Arduino แต่ละรุ่น

หลักการทำงานของโปรแกรมภายในบอร์ด Arduino ดังแสดงในรูปข้างล่าง โดยเมื่อบอร์ด Arduino เริ่มทำงาน ขึ้น โค้ดโปรแกรมภายในฟังก์ชันตั้งค่า `Setup()` ก็จะถูกเรียกในครั้งแรก ซึ่งส่วนจะเป็นการตั้งค่าการทำงานให้กับบอร์ด และการกำหนดรายละเอียดการทำงานให้กับพอร์ตต่างๆ หลังจากนั้น

โปรแกรมจะเข้าสู่ฟังก์ชันลูป `loop()` ในขั้นตอนนี้โปรแกรมจะวนทำงานอยู่ภายในฟังก์ชัน `loop()` นี้ไปเรื่อยๆจนกว่าจะมีคำสั่งให้รีเซตการทำงานของบอร์ดใหม่อีกครั้ง (Arduino Reset)



รูปที่ 7.19 แสดงขั้นตอนการทำงานภายในโปรแกรม Arduino

ตัวอย่างการเชื่อมต่อระหว่าง Android กับ Arduino ผ่าน ADK

คลาสสำคัญภายใน ADK2012 IDE สำหรับใช้ในการสื่อสารผ่าน ADK ระหว่างแอนดรอยด์และ Arduino นั้นได้แก่คลาส AndroidAccessory ซึ่งมีรายละเอียดภายในดังนี้

```
class AndroidAccessory {
  private:
      const char *manufacturer;
      const char *model;
      const char *description;
      const char *version;
      const char *uri;
      const char *serial;
      ...
      int getProtocol(byte addr);
      void sendString(byte addr, int index, const char *str);
      bool switchDevice(byte addr);
      bool findEndpoints(byte addr, EP_RECORD *inEp, EP_RECORD *outEp);
      bool configureAndroid(void);

  public:
      AndroidAccessory(const char *manufacturer,
                       const char *model,
                       const char *description,
                       const char *version,
                       const char *uri,
                       const char *serial);

      void powerOn(void);
```

```
    bool isConnected(void);
    int read(void *buff, int len, unsigned int nakLimit = USB_NAK_LIMIT);
    int write(void *buff, int len);
};
```

ฟังก์ชันหลักที่จะเรียกใช้ภายในฟังก์ชัน setup() เพื่อใช้ในการเปิด Android Accessory คือ powerOn(), read() และ write() โดยที่ตัวแปร buff นั้นจะใช้ในการเก็บข้อมูลที่ต้องการจะส่งออกหรือรับเข้ามาจากฝั่งโปรแกรมแอนดรอยด์ผ่านทาง USB ตามขนาดความยาวของข้อมูล (len) ในกรณีที่เป็นการรับส่งข้อมูลแบบเต็มความเร็ว (Full Speed USB) ในฟังก์ชัน read() นั้นส่วนของค่า NAK จะเป็นการบอกค่าสถานะว่าอุปกรณ์ที่จะส่งข้อมูลให้นั้นยังไม่พร้อมที่จะส่งกลับหรือยังไม่มีข้อมูลที่จะส่งให้ในเวลานี้ นอกจากนั้นก็ยังถูกใช้ในการแจ้งเตือนระหว่างถ่ายโอนข้อมูลไปยังเครื่องรับว่าไม่มีข้อมูลที่จะส่งให้แล้ว

ตัวอย่างข้างล่างเป็นการเขียนโปรแกรมเพื่อทำการเชื่อมต่อ ADK แล้วรับข้อมูลที่ถูกส่งมาจากแอนดรอยด์

```
#include <AndroidAccessory.h>

    AndroidAccessory acc("Google, Inc.",
              "DemoKit",
              "DemoKit Arduino Board",
              "1.0",
              "http://www.android.com",
              "0000000012345678");

    void Setup(){
        acc.powerOn(); // สั่งเปิดใช้งาน Android Accessory
    }
    void loop(){
        if (acc.isConnected()) {
             // อ่านค่าที่ถูกส่งออกมาจาก Android
            int len = acc.read(msg, sizeof(msg), 1);
        }
    }
```

การเขียนโปรแกรมส่วนแอนดรอยด์

ในฝั่งของแอนดรอยด์จะใช้คลาสหลักได้แก่คลาส UsbManager และคลาส UsbAccessory ที่กูเกิลได้จัดเตรียมไว้ให้เบื้องต้นแล้ว โดยขั้นตอนแรกนักพัฒนาจะต้องสร้างโปรเจคของแอนดรอยด์ขึ้นมาแล้วทำการระบุการเรียกใช้ UsbAccessory เข้าไปในไฟล์ AndroidManifest.xml ดังแสดงด้านล่าง

```
    <application
       android:icon="@drawable/icon"
       android:label="@string/app_name" >
    <uses-library android:name="com.android.future.usb.accessory" />
        ...
```

```
            ...
            ...
    </application>
```

โค้ดส่วนหนึ่งภายในโปรแกรมแอนดรอยด์เพื่อขอเปิดและเข้าถึงการรับส่งข้อมูลผ่าน USB

```
// เป็นการกำหนด Intent เบื้องต้นสำหรับการดักอีเว้นท์กรณีที่มีการเชื่อมต่อ ADK
private PendingIntent mPermissionIntent;
private static final String ACTION_USB_PERMISSION =
"com.android.example.USB_PERMISSION";

private UsbManager mUsbManager;
private UsbAccessory mAccessory;

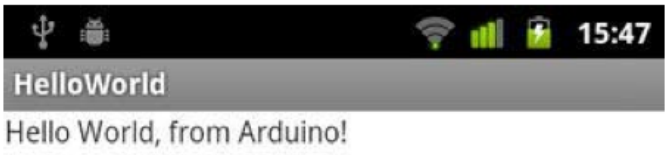
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   mUsbManager = UsbManager.getInstance(this);
   mPermissionIntent = PendingIntent.getBroadcast(this, 0, new Intent(
                  ACTION_USB_PERMISSION), 0);
// กำหนด filter
  IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
  filter.addAction(UsbManager.ACTION_USB_ACCESSORY_DETACHED);
// register ให้สามารถรับ intent ได้
  registerReceiver(mUsbReceiver, filter);
  setContentView(R.layout.main);
  textView = (TextView) findViewById(R.id.textView);
}

/** Called when the activity is resumed from its paused state and
immediately after onCreate(). */
@Override
public void onResume() {
  super.onResume();
  if (mInputStream != null && mOutputStream != null) {
   return; }
  UsbAccessory[] accessories = mUsbManager.getAccessoryList();
  UsbAccessory accessory = (accessories == null ? null : accessories[0]);
  if (accessory != null) {
//  ตรวจสอบสิทธิ์ในการเรียกใช้งาน Accessory
  if (mUsbManager.hasPermission(accessory)) {
     openAccessory(accessory); // สั่งเปิดการใช้งาน accessory
  } else {
     synchronized (mUsbReceiver) {
  if (!mPermissionRequestPending) {
     mUsbManager.requestPermission(accessory, mPermissionIntent);
    mPermissionRequestPending = true;
  }
 }
}
} else {
```

```
    Log.d(TAG, "mAccessory is null");
  }
}
/** เมื่อ Application ถูกปิดโปรแกรมจะทำการปิด Accessory */
@Override
public void onPause() {
  super.onPause();
  closeAccessory();
}

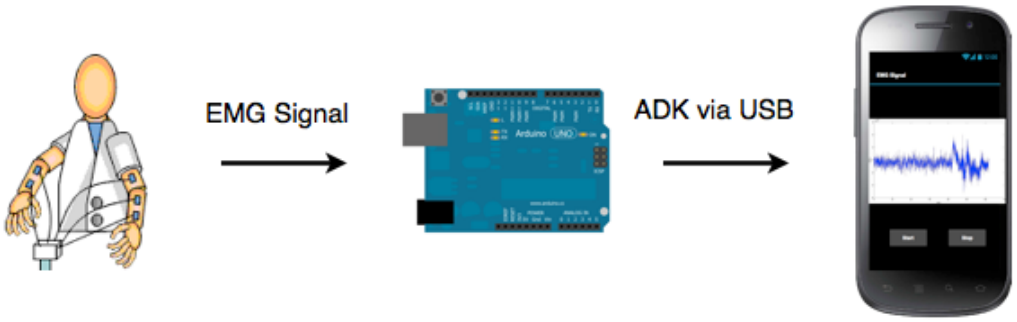
/** เมื่อปิด Application จะทำการ unregister Intent  */
@Override
public void onDestroy() {
  super.onDestroy();
  unregisterReceiver(mUsbReceiver);
}
```

จากโค้ดตัวอย่างข้างต้นเป็นการตรวจสอบและสั่งเปิดโหมดการเชื่อมต่อผ่าน USB ซึ่งหากมีการเชื่อมต่อเข้ามาของบอร์ด Arduino ผ่านสาย USB ก็จะทำในฟังก์ชัน openAccessory(accessory) เมื่อสั่งเปิดการทำงานแล้วภายในจะมีการสร้าง Thread ขึ้นมา เพื่อทำการอ่านข้อมูลที่ถูกส่งออกมาจากบอร์ด Arduino ดังรูปข้างล่าง



รูปที่ 7.20 ผลลัพธ์การรับข้อมูลจากบอร์ด Arduino ผ่านพอร์ต USB

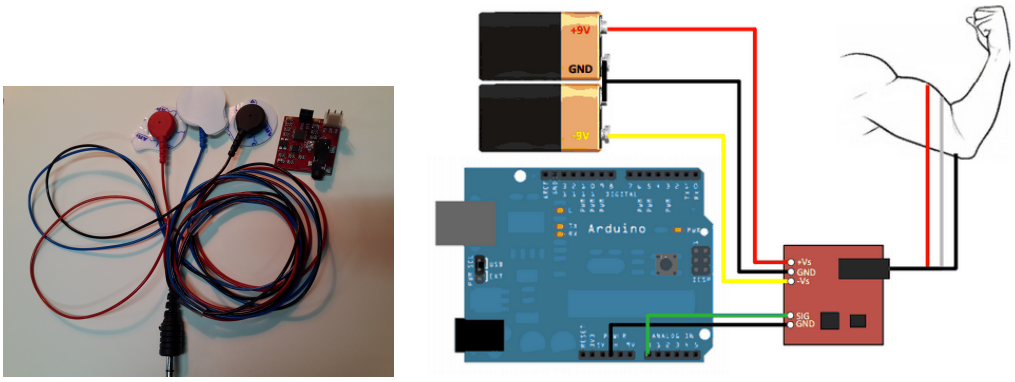
ตัวอย่างการอ่านและแสดงสัญญาณไฟฟ้ากล้ามเนื้อ (EMG)



รูปที่ 7.21 แสดงภาพรวมของระบบ

สำหรับตัวอย่างนี้เป็นการตรวจวัดสัญญาณกล้ามเนื้อที่ได้รับจากเซ็นเซอร์ EMG ที่เชื่อมต่อกับบอร์ด Arduino แล้วถูกส่งข้อมูลต่อไปยังอุปกรณ์แอนดรอยด์ผ่านสาย USB ด้วยมาตราฐาน AOA Protocol หลังจากนั้นจึงนำมาแสดงผลในลักษณะกราฟเส้น ซึ่งมีแนวทางการพัฒนาด้วยกัน 2 ส่วนดังนี้

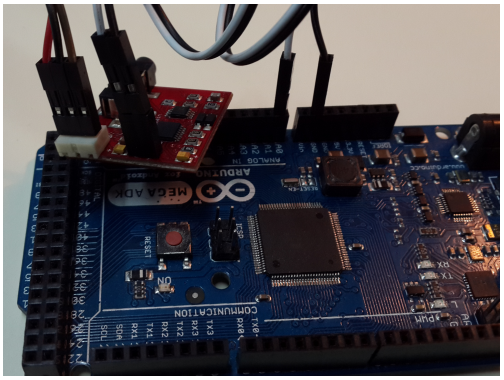
พัฒนาส่วนโปรแกรมภาษาซีบนบอร์ด Arduino และการต่อกับเซ็นเซอร์ EMG



รูปที่ 7.22 แสดงการเชื่อมต่อเซ็นเซอร์ EMG กับบอร์ด Arduino

หลักการทำงานของเซ็นเซอร์ EMG คือแผ่น EMG จะทำหน้าที่เป็นสื่อนำไฟฟ้าเมื่อผู้ใช้ขยับร่างกาย จะเกิดสัญญาณทางไฟฟ้าเคลื่อนที่ไปตามกล้ามเนื้อของร่างกาย ดังนั้นแผ่น EMG จะทำการขยายสัญญาณทางไฟฟ้านั้น ให้อยู่ในช่วงที่บอร์ด Arduino สามารถตรวจวัดได้โดยค่าสัญญาณที่ได้ออกมาจะเป็นค่าระดับแรงดันกระแสตรงขนาดช่วง 0-5 โวลท์

ในส่วนของบอร์ด Arduino จะประกอบไปด้วย มีสองส่วนที่ทำงานร่วมกันคือ ส่วนแรกเป็นส่วนของการอ่านค่า Analog Read Voltage (จากตัวอย่าง) ส่วนที่สองเป็นการเชื่อมต่อ ADK กับอุปกรณ์ Android จากนั้นจะทำการอ่านค่า Analog เพื่อส่งข้อมูลที่อ่านได้ผ่าน ADK ไปยังอุปกรณ์ Android เพื่อนำข้อมูลไปวิเคราะห์ต่อไป



รูปที่ 7.23 แสดงการเชื่อมต่อเซ็นเซอร์ EMG กับบอร์ด Arduino

ในส่วนนี้จะเป็นการเขียนโปรแกรมบนบอร์ด Arduino เพื่อทำการเปิด accessory mode แล้วอ่านค่าแรงดันที่ได้จากเซนเซอร์ EMG เมื่อได้ข้อมูลก็จะทำการจัดข้อมูลในรูปแบบ “E*<ค่าที่อ่านได้>*” เพื่อส่งผ่าน USB ไปยังแอนดรอยด์ต่อไป ดังตัวอย่างโปรแกรมข้างล่างนี้

```
#include <Wire.h>
#include <Servo.h>
#include <Max3421e.h>
#include <Usb.h>
#include <AndroidAccessory.h>
#define  LED3_RED       2
```

```
#define  LED3_GREEN     4
#define  LED3_BLUE      3
#define  LED2_RED       5
#define  LED2_GREEN     7
#define  LED2_BLUE      6
#define  LED1_RED       8
#define  LED1_GREEN     10
#define  LED1_BLUE      9

AndroidAccessory acc("Google, Inc.",
                     "DemoKit",
                     "DemoKit Arduino Board",
                     "1.0",
                     "http://www.android.com",
                     "0000000012345678");

boolean sendData=false;
void setup();
void loop();

void init_leds()
{
      digitalWrite(LED1_RED, 1);
      digitalWrite(LED1_GREEN, 1);
      digitalWrite(LED1_BLUE, 1);

      pinMode(LED1_RED, OUTPUT);
      pinMode(LED1_GREEN, OUTPUT);
      pinMode(LED1_BLUE, OUTPUT);

      digitalWrite(LED2_RED, 1);
      digitalWrite(LED2_GREEN, 1);
      digitalWrite(LED2_BLUE, 1);

      pinMode(LED2_RED, OUTPUT);
      pinMode(LED2_GREEN, OUTPUT);
      pinMode(LED2_BLUE, OUTPUT);

      digitalWrite(LED3_RED, 1);
      digitalWrite(LED3_GREEN, 1);
      digitalWrite(LED3_BLUE, 1);

      pinMode(LED3_RED, OUTPUT);
      pinMode(LED3_GREEN, OUTPUT);
      pinMode(LED3_BLUE, OUTPUT);
}

byte b1, b2, b3, b4, c;
int value=0;
void setup()
{
      pinMode(0, INPUT);
```

```
	Serial.begin(115200);
	Serial.print("\r\nStart");
	init_leds();
	c = 0;
	acc.powerOn();
}


void loop()
{
	byte err;
	byte idle;
	static byte count = 0;
	byte msg[5];
	long touchcount;
	if (acc.isConnected()) {
		value = analogRead(0);
		msg[0] = 'E';
		msg[1] = value;
		acc.write(msg, sizeof(msg));
		Serial.println("\f");
		Serial.println(msg[0]);
		Serial.println(msg[1]);
	}
}
```

พัฒนาส่วนโปรแกรมแอนดรอยด์

ส่วนนี้เป็นการเขียนโปรแกรมประยุกต์แอนดรอยด์ เพื่อทำการเชื่อมต่อ ADK กับบอร์ด Arduino จากนั้นก็นำข้อมูลมาแสดงเป็นกราฟแบบเวลาจริง ในที่นี้จะอธิบายเกี่ยวกับการสร้างกราฟเบื้องต้นเพื่อนำมาแสดงผลทันทีที่รับได้ข้อมูลจากบอร์ด Arduino โดยการประยุกต์ใช้ไลบรารีชื่อว่า GraphView ซึ่งสามารถสร้างกราฟได้ทั้งแบบชนิดกราฟเส้น และกราฟแท่ง

```
int num = 150;  // จำนวนของข้อมูล
GraphViewData[] data = new GraphViewData[num];
double v = 0;
for (int i = 0; i < num; i++) {
     v += 0.2;
     // คำนวณค่าในแต่ละจุดที่จะทำการ plot กราฟ
     data[i] = new GraphViewData(i, Math.sin(v));
}
GraphView graphView = new LineGraphView(this, "AndroidGraph");
graphView.addSeries(new GraphViewSeries(data));
// ตั้งค่าของกราฟให้เริ่มต้นที่จุดที่ 2 ให้มีขนาดเท่ากับ 40 ช่อง
graphView.setViewPort(2, 40);
// ตั้งค่าให้สามารถ Scroll ได้
graphView.setScrollable(true);
```

```
graphView.setScalable(true);
LinearLayout layout = (LinearLayout) findViewById(R.id.lay1);
layout.addView(graphView);

// ตั้งค่าให้กับแกน X
graphView.setHorizontalLabels(new String[] {"2 days ago", "yesterday", "today", "to-
morrow"});
// ตั้งค่าให้กับแกน Y
graphView.setVerticalLabels(new String[] {"high", "middle", "low"});
```

ในกรณีที่ต้องการเพิ่มเส้นข้อมูลใหม่ให้ทำการสร้าง GrapViewSeries ใหม่จากนั้นให้ทำการเพิ่มเข้าไปในการ Graph-View ที่ได้สร้างไว้แล้ก่อนหน้าเช่น

```
// draw sin curve
int num = 150;
GraphViewData[] data = new GraphViewData[num];
double v = 0;
for (int i = 0; i < num; i++) {
    v += 0.2;
    data[i] = new GraphViewData(i, Math.sin(v));
}
GraphViewSeries seriesSin = new GraphViewSeries(data);
// draw cos curve
data = new GraphViewData[num];
v=0;
for (int i=0; i<num; i++) {
  v += 0.2;
  data[i] = new GraphViewData(i, Math.cos(v));
}
GraphViewSeries seriesCos = new GraphViewSeries( data);
```

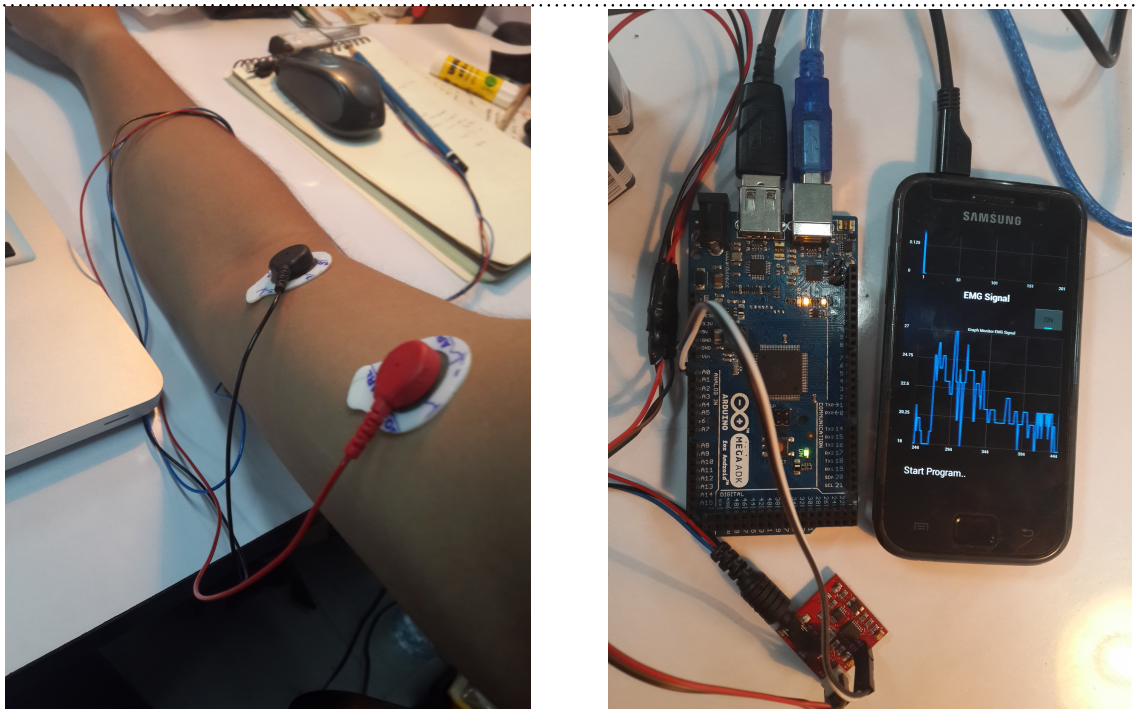
จากข้างต้นเป็นตัวอย่างการสร้างกราฟจากข้อมูลที่มี แต่เนื่องจากการนำค่า EMG มาแสดงนั้นเป็นการแสดงแบบเวลาจริง (Real-time) จึงจำเป็นจะต้องใช้เทคนิคของแอนดรอยด์ เกี่ยวกับการวนรอบเพื่อทำการแสดงผลกราฟทันทีเมื่อได้รับข้อมูลใหม่เพิ่มเข้ามา โดยใช้ Handler และ Runnable ดังตัวอย่างโค้ดข้างล่าง

```
// โค้ดตัวอย่างด้านล่างจะเป็นการสร้าง Runnable ขึ้นมาเพื่อทำการสร้างตัวส่งค่าขึ้นไปเพื่อทำการ plot กราฟ
จากนั้นจะใช้ Handler เพื่อสร้างลูปในการวนทุก 1 มิลลิวินาที
Handler mHandler = new Handler();
Runnable mTimer1;
mTimer1 = new Runnable() {
  @Override
  public void run() {
     graph2LastXValue += 1d;
     exampleSeries2.appendData(new GraphViewData(graph2LastXValue,
                                getRandom()), true);
                    mHandler.postDelayed(this, 1);
     }
};
```

เมื่อถึงจุดนี้จะสามารถสร้างการทำงานของกราฟที่มีการ plot แบบ Real-time ได้ต่อมาจะเป็นการสร้างฟังก์ชันที่ทำงานแบบ Async Task เพื่อทำการอ่านค่าที่ถูกส่งผ่านมาทาง ADK ดังตัวอย่าง

```
// ฟังก์ชันเปิดใช้งาน ADK และสร้าง InputStream เพื่อเตรียมการอ่านข้อมูล
private void openAccessory(UsbAccessory accessory) {
          mFileDescriptor = mUsbManager.openAccessory(accessory);
          if (mFileDescriptor != null) {
               mAccessory = accessory;
               FileDescriptor fd = mFileDescriptor.getFileDescriptor();
               mInputStream = new FileInputStream(fd);
               new receiveData().execute();
               enableControls(true);
          }
}

public class receiveData extends AsyncTask<Object, Void, Void> {
  @Override
  protected Void doInBackground(Object... params) {
     while (true) {
      // ทำการอ่านข้อมูลและเก็บไว้ใน array Buffer
        ret = mInputStream.read(buffer);
      /* ทำการอ่านข้อมูลและเก็บไว้ จากนั้นให้ดึงค่าตัวแปรดังกล่าวนำไป plot กราฟ */
          }
     }
}
```



รูปที่ 7.24 แสดงการเชื่อมต่อเซ็นเซอร์ EMG กับบอร์ด Arduino

บทสรุปและก้าวต่อไป...

เนื้อหาทั้งหมดของหนังสือเล่มนี้ล้วนแล้วได้มาจากประสบการณ์ของผู้เขียนที่พยายามเก็บรวบรวมองค์ความรู้พื้นฐานที่สำคัญต่างๆในตลอดระยะเวลาหลายสิบปี รวมทั้งยังได้แรงบันดาลใจมาจากหนังสือ Textbook เหล่านี้ ซึ่งได้แก่ Embedded Linux Primer A Practical Real World Approach, Pro Linux Embedded Systems, Building Embedded Linux Systems, Embedded Android, Android Native Development Kit Cookbook ที่ทำให้ผู้เขียนมีความตั้งใจอยากจะทำการถ่ายทอดออกมาให้เป็นหนังสือภาษาไทยเพื่อนักพัฒนาคนไทย ซึ่งอย่างน้อยน่าจะพอที่จะสามารถช่วยจุดประกายความฝันให้กับนักพัฒนาไทยให้หันมาให้ความสำคัญในเทคโนโลยีระบบสมองกลฝังตัวได้ไม่มากก็น้อย

โดยผู้เขียนได้รวบรวมองค์ความรู้พื้นฐานตั้งแต่ระบบปฏิบัติการลีนุกซ์ ชุดเครื่องมือสำหรับนักพัฒนาการพัฒนาโปรแกรมภาษาต่างๆที่เกี่ยวข้อง (C, C++, JAVA, Python) มุมมองแต่ละส่วนของเทคโนโลยีระบบสมองกลฝังตัวและรายละเอียดทางด้านเทคนิคที่นักพัฒนาควรรู้ ซึ่งอาจจะยังไม่ได้ลงในรายละเอียดเชิงลึกมากสักเท่าใด เนื่องจากผู้เขียนต้องการเชื่อมโยงองค์ความรู้จากศาสตร์ด้านต่างๆแล้วนำมาเรียบเรียงอยู่ไว้ในบทแต่ละบทให้ครบถ้วนมากที่สุดเพื่อให้ผู้อ่านสามารถทำความเข้าใจในแต่ละส่วนได้ว่ามีส่วนใดบ้างที่เกี่ยวข้องกับระบบสมองกลฝังตัวและสำหรับผู้อ่านที่ต้องการจะก้าวเป็นนักพัฒนาทางด้านระบบสมองกลฝังตัวอย่างจริงจังได้มีโอกาสทดลองใช้ชุดคำสั่งต่างๆทั้งบนระบบปฏิบัติการลีนุกซ์และระบบปฏิบัติการแอนดรอยด์ รวมทั้งทำความเข้าใจในการออกแบบพัฒนาภาษาโปรแกรมให้คุ้นเคยมากยิ่งขึ้น

แต่ก็ยังคงมีเนื้อหาและรายละเอียดทางด้านเทคนิคสำหรับการออกแบบพัฒนาระบบสมองกลฝังตัวที่น่าสนใจอีกมากโดยเฉพาะการพัฒนาภายใต้ระบบปฏิบัติการลีนุกซ์แบบฝังตัวและระบบปฏิบัติการแอนดรอยด์ รวมทั้งแนวทางการออกแบบบอร์ดสมองกลฝังตัวสำหรับงานประยุกต์ ที่ผู้เขียนตั้งใจไว้ว่าจะพยายามเรียบเรียงเพื่อเขียนลงในเล่มถัดๆไป ซึ่งได้แก่

- การพัฒนาโปรแกรมเชลล์สคริปท์ขั้นสูง
- การปรับแต่งลีนุกซ์เคอร์เนลและติดตั้งโปรแกรม Open Sources
- การติดตั้งโปรแกรมให้บริการสำหรับระบบเครือข่าย เช่น Web Server, File Server เป็นต้น
- การพัฒนาโปรแกรมระดับล่างเพื่อติดต่อกับระบบฮาร์ดแวร์
- การพัฒนาโปรแกรม Linux Kernel Module สำหรับงานประยุกต์
- การพัฒนาโปรแกรมเพื่อเรียกใช้ไลบรารี Native C/C++
- การพัฒนาโปรแกรม Qt 5.2 สำหรับอนาคตของอุปกรณ์ชนิดพกพา
- การใช้งานและปรับแต่ง Android Framework
- กลไกการทำงานในระดับล่างภายในระบบปฏิบัติการแอนดรอยด์เช่น Android IPC/Binder Framework, Android Hardware Abstract Layer (HAL) เป็นต้น
- การพัฒนา Android NDK Application
- การพัฒนาโปรแกรมประยุกต์ด้วยบอร์ดสมองกลฝังตัว เช่น BeagleBone, BeagleBoard xM, iMX53 QSB, PandaBoard ES, AM335x Starter Kit, ODROID-X2, RaspBerry Pi, Intel Galileo เป็นต้น

ประวัติผู้เขียน



ผู้ช่วยศาสตราจารย์วิรุฬห์ ศรีบริรักษ์
อาจารย์ประจำภาควิชาวิศวกรรมไฟฟ้า
คณะวิศวกรรมศาสตร์ มหาวิทยาลัยบูรพา

ประวัติการทำงาน

2553 - ปัจจุบัน	รองคณบดีฝ่ายพัฒนานิสิต คณะวิศวกรรมศาสตร์ มหาวิทยาลัยบรูพา
2553 - ปัจจุบัน	กรรมการบริหารสมาคมสมองกลฝังตัวแห่งประเทศไทย (TESA)
2550 - ปัจจุบัน	หัวหน้าห้องปฏิบัติการวิจัยและพัฒนาระบบสมองกลฝังตัวและทฤษฎีข้อมูลชั้นสูง (ESTIA) ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ มหาวิทยาลัยบรูพา
2548 - ปัจจุบัน	รองประธานหน่วยวิจัยโลจิสติกส์และวิศวกรรมสมองกลฝังตัว (หัวหน้าทีมวิศวกรรมสมองกลฝังตัว) BAL-LABS มหาวิทยาลัยบรูพา
2550 - 2551	หัวหน้าทีมวิจัยและผู้คิดค้นออกแบบระบบสื่อสารข้อมูลแบบไร้สายระหว่างรถและสถานี RFID Sensor Network และโปรโตคอลสื่อสารระหว่างสถานี RFID Sensor Network และ ศูนย์ควบคุมกลาง
2549 - 2550	หัวหน้าทีมออกแบบและพัฒนาระบบฮาร์ดแวร์และซอฟต์แวร์ให้กับบริษัท Endemol ผู้เป็นเจ้าของรายการ Fear Factor, Survivor และ 1 versus 100
2549 - 2551	หัวหน้าภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ มหาวิทยาลัยบรูพา
2549 - 2552	คณะกรรมการผู้เชี่ยวชาญสมาคมสมองกลฝังตัวแห่งประเทศไทย
2547 - 2548	รองคณบดีฝ่ายกิจการพิเศษ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยบรูพา
2546 - 2547	รักษาราชการแทนผู้ช่วยคณบดีฝ่ายเทคโนโลยีสารสนเทศ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยบรูพา
2544 - 2546	หัวหน้าทีม Wireless Sensor Network Team โครงการ Korea Brain Project, Korea University, Seoul, Republic of South Korea

รางวัลที่ได้รับ

2555	รางวัลรัตนบูรพา สาขาการสร้างสรรค์และประดิษฐ์คิดค้น มหาวิทยาลัยบูรพา
2553	รางวัลสิ่งประดิษฐ์ทางวิทยาศาสตร์ที่มีศักยภาพสูงในการลงทุน โดยสำนักงานพัฒนาวิทยาศาสตร์และเทคโนโลยีแห่งชาติ (NSTDA) กระทรวงวิทยาศาสตร์
2552	รางวัลนวัตกรรมแห่งชาติทางด้านเศรษฐกิจ ในชื่อ “ระบบเครือข่ายเซ็นเซอร์ไร้สาย” เพื่อใช้ในการติดตามยานพาหนะแบบเรียลไทม์ สำนักงานนวัตกรรมแห่งชาติ (NIA)
2552	รางวัลชนะเลิศ การออกแบบและพัฒนาผลิตภัณฑ์สมองกลฝังตัวแห่งประเทศไทย ประเภท Intelligent Transport System (ITS) and Automotive “Vehicle Identification ITS Solution” สมาคมสมองกลฝังตัวแห่งประเทศไทย
2549	รางวัลยอดเยี่ยมในการนำเสนอแนวทางการนำสื่อ E-Learning มาใช้ทางด้านการศึกษา ที่ TIT University, ญี่ปุ่น
2545	Best Paper Award in Wireless Sensor Network, Korea University, Seoul, South Korea
2543	Republic of South Korea Government Scholarship Grantee

สถานที่ทำงาน

ห้องปฏิบัติการวิจัยและพัฒนาระบบสมองกลฝังตัวและทฤษฎีข้อมูลชั้นสูง (ESITA)
ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ มหาวิทยาลัยบูรพา
ถนนลงหาดบางแสน ตำบลแสนสุข อำเภอเมืองชลบุรี จังหวัดชลบุรี

โทรศัพท์ 038-10-2222 ต่อ 3380 โทรสาร 038-74-5806
E-mail wiroon@eng.buu.ac.th

EMBEDDED ANDROID DEVELOPMENT

หนึ่งในหลายๆกำลังใจถึงนักเขียน จากผู้อ่าน 1,500 ท่านแรก

ขอแสดงความยินดีในความสำเร็จด้านการเขียนหนังสือ แล้วผมจะนำความรู้ในหนังสือไปใช้ให้เกิดประโยชน์ต่อศิษย์และตนเอง...

ขอบคุณครับที่ได้เขียนหนังสือเล่มนี้ออกมา เพราะตอนนี้ผมก็ทำทางด้านนี้อยู่ แต่ก็ยังไม่เข้าใจถ่องแท้ และในเมืองไทยก็ยังไม่ค่อยมี ถ้าได้หนังสือเล่มนี้มานำทางและผมก็อยากมาประสบการณ์ที่สามารถไปถ่ายทอดให้น้องๆๆได้อีกด้วย ขอบคุณครับ...

ขอบคุณมากครับ ผมเป็นคนนอกสายงานด้านนี้ (เด็กช่างกล) อาศัยศึกษาจากผู้รู้มั่ง หนังสือมั่ง จาก google มั่ง หนังสือของอาจารย์เล่มนี้ คงช่วยให้คนนอกวงการอย่างผมมีโอกาศเข้าใจและนำไปใช้ได้มากขึ้น แน่ๆ ขอบคุณมากๆครับ อาจารย์ อยากให้ประเทศนี้มีคนแบบนี้เยอะๆครับ ขอบคุณครับ...

อยากให้อาจารย์ทำหนังสือดี ๆ แบบนี้ออกมาเรื่อย ๆ ครับ จากความคิดหนึ่งสู่อีกความคิดหนึ่ง มันจะแตกกิ่งออกไปเหมือนกับต้นไม้...สุดท้ายแล้วต้นไม้ต้นนั้นก็จะเติบโตและให้ประโยชน์ต่อ ๆ ไป แล้วผมจะนำความรู้ในหนังสือไปใช้ให้เกิดประโยชน์ต่อศิษย์และตนเอง...

อยากให้หลายๆ คนเป็นแบบนี้นะค่ะ แบ่งบันประสบการณ์การณ์และความรู้แบบทอดๆ มันทำให้รู้สึกว่า .. นอกจากจะน่าชื่นชมในความเก่งและความสำเร็จของเค้าแล้ว ยังรู้สึกซาบซึ้งใจที่ทำให้คนอีกหลายๆ คนได้รับความรู้นั้น ขอบคุณมากนะค่ะ ;) ....

ผมเคยเป็นเด็กใน HyperCamp และ TESA TopGun ขอชื่อชมผลงานของ อาจารย์ และ จะเดินสาย Embedded ต่อครับ ...

ผมคลั่งไคล้อาจารย์ตั้งแต่แข่ง TopGun แล้วครับ (โอเวอร์มากๆๆ (>.<)) ขอบคุณอาจารย์มากๆครับที่เขียนหนังสือดีๆอย่างนี้มาแบ่งปันให้ได้อ่านครับผม -(^_^)- ...

ผมมีความสนใจในด้านนี้แต่ยังขาดความรู้ประสบการและอ่อนแอในด้านภาษา นับเป็นการเริ่มต้นที่ดีมากเลยครับสำหรับหนังสือเล่มนี้ หนังสือเล่มเป็นเหมือนแรงบันดาลใจเลยครับ ขอบคุณครับ :) ...



タイ語の Embedded Android があった、よかった。＾__＾

อาจารย์จอมครับ ผมขอขอบคุณสำหรับการเผยแพร่ความรู้ที่เป็นประโยชน์อย่างมากมายนะครับ และผมจะนำความรู้ที่ได้มาใช้ให้เกิดประโยชน์ให้มากที่สุดนะครับ ^__^

ขอเป็นกำลังใจให้เผยแพร่บทความหรือหนังสือดีๆต่อไปครับ เพราะคนเราไม่มีใครรู้ทุกเรื่อง สิ่งทีคุณรู้ไม่ว่าจะมาจากประสบการณ์ตรงจากการทำงานหรือการขวนขวานเรียนรู้นั้นผมคิดว่ามันต้องมีประโยชน์กับผู้อื่นอย่างแน่นอนไม่มากก็น้อย สุดท้ายอยากจะบอกว่าถ้าเรายิ่งทำก็ยิ่งได้ยิ่งให้ก็ยิ่งมี ขอบคุณครับ...

ขอขอบคุณสำหรับหนังสือคุณภาพของผู้เขียน ผมมีความสนใจในระบบปฏิบัติการณ์แอนดรอยด์เป็นอย่างมาก อีกทั้งเพื่อเป็นการพัฒนาความรู้ด้านความเข้าใจในระบบสมองกลฝังตัว เพื่อพัฒนาศักยภาพของตัวผมเองให้เป็นนักพัฒนาที่ดีในอนาคต ขอขอบคุณผู้เขียนเป็นอย่างมากครับ...

ผมพยายามจะศึกษามานาน ด้วยอายุเยอะ และภาษาอังกฤษไม่ค่อยดี มีหนังสือภาษาไทยมาให้อ่านนับว่าเป็นความกรุณาต่อผมมากๆ ขอให้ท่านผู้จัดทำได้รับการตอบแทนสิ่งดีๆ จากพระผู้เป็นเจ้า...