# วิทยานิพนธ์

## การพอร์ต μClinux และการพัฒนาโปรโตคอล RTP/RTCP บนระบบแบบฝังตัว

## A PORTING OF μCLINUX AND A DEVELOPMENT OF RTP/RTCP PROTOCOL ON AN EMBEDDED SYSTEM

นายศศิน จันทร์พวงทอง

บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์

พ.ศ. 2550

# ใบรับรองวิทยานิพนธ์

**บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์**

วิศวกรรมศาสตรมหาบัณฑิต (วิศวกรรมคอมพิวเตอร์)

**ปริญญา**

| วิศวกรรมคอมพิวเตอร์ | วิศวกรรมคอมพิวเตอร์ |
|---|---|
| **สาขา** | **ภาควิชา** |

**เรื่อง** การพอร์ต μClinux และการพัฒนาโปรโตคอล RTP/RTCP บนระบบแบบฝังตัว

A Porting of μClinux and a Development of RTP/RTCP Protocol on an Embedded System

**นามผู้วิจัย** นายศศิน จันทร์พวงทอง

**ได้พิจารณาเห็นชอบโดย**

**ประธานกรรมการ** .....

( ผู้ช่วยศาสตราจารย์เขมะฑัต วิภาตะวนิช, Ph.D. )

**กรรมการ** .....

( ผู้ช่วยศาสตราจารย์ภุชงค์ อุทโยภาศ, Ph.D. )

**กรรมการ** .....

( อาจารย์พีรวัฒน์ วัฒนพงศ์, Ph.D. )

**หัวหน้าภาควิชา** .....

( ผู้ช่วยศาสตราจารย์เขมะฑัต วิภาตะวนิช, Ph.D. )

**บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์รับรองแล้ว**

.....

( รองศาสตราจารย์วินัย อาจคงหาญ, M.A. )

**คณบดีบัณฑิตวิทยาลัย**

**วันที่** ..... **เดือน** ..... **พ.ศ.** .....

วิทยานิพนธ์

เรื่อง

# การพอร์ต μClinux และการพัฒนาโปรโตคอล RTP/RTCP บนระบบแบบฝังตัว

# A Porting of μClinux and a Development of RTP/RTCP Protocol on an Embedded System


โดย

นายศศิน จันทร์พวงทอง


เสนอ

บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์
เพื่อความสมบูรณ์แห่งปริญญาวิศวกรรมศาสตรมหาบัณฑิต (วิศวกรรมคอมพิวเตอร์)
พ.ศ. 2550

ศศิน จันทร์พวงทอง 2550: การพอร์ต μClinux และการพัฒนาโปรโตคอล RTP/RTCP บนระบบแบบฝังตัว ปริญญาวิศวกรรมศาสตรมหาบัณฑิต (วิศวกรรมคอมพิวเตอร์) สาขาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ ประธานกรรมการที่ปรึกษา: ผู้ช่วยศาสตราจารย์เขมะฑัต วิภาตะวนิช, Ph.D. 121 หน้า

งานวิจัยชิ้นนี้มีวัตถุประสงค์ที่จะพัฒนาโปรโตคอล RTP และ RTCP ขึ้นมาใช้งานบนระบบคอมพิวเตอร์แบบฝังตัว เพื่อให้ระบบคอมพิวเตอร์แบบฝังตัวสามารถรับส่งข้อมูลประเภท real-time บนเครือข่าย IP ได้ ซึ่งความสามารถดังกล่าวถือเป็นพื้นฐานในการสร้างอุปกรณ์สื่อสารสำหรับรับส่งข้อมูลประเภทเสียงและภาพบนเครือข่าย IP โดยจะทำการพัฒนาโปรโตคอล RTP และ RTCP ให้อยู่ในรูปแบบของ system call บนระบบปฏิบัติการ μClinux ซึ่งเป็นระบบปฏิบัติการที่นิยมนำมาใช้กับระบบคอมพิวเตอร์แบบฝังตัวที่ใช้หน่วยประมวลผลแบบไม่มี MMUอยู่ภายใน โดยการพัฒนาโปรโตคอลดังกล่าวในลักษณะของ system call ที่ทำงานอยู่บน kernel-mode ของระบบปฏิบัติการจะช่วยให้โปรโตคอล RTP และ RTCP สามารถทำงานอยู่ภายในระบบคอมพิวเตอร์แบบฝังตัวได้อย่างรวดเร็วและมีประสิทธิภาพมากกว่า API ที่อยู่ภายใน library ของโปรโตคอล RTP และ RTCP ซึ่งทำงานอยู่บน user-mode ของระบบปฏิบัติการ รวมถึงจะช่วยอำนวยความสะดวกให้กับผู้อื่นในการนำ system call เหล่านี้ไปพัฒนาเป็นแอพพลิเคชั่นหรือโปรโตคอลระดับสูงที่ใช้ในการรับส่งข้อมูลประเภท real-time บนเครือข่าย IP สำหรับทำงานกับระบบคอมพิวเตอร์แบบฝังตัวต่อไปในอนาคต

_____ _____ ___ / ___ / ___

ลายมือชื่อนิสิต ลายมือชื่อประธานกรรมการ

Sasin Janpuangtong 2007: A Porting of μClinux and a Development of RTP/RTCP Protocol on an Embedded System. Master of Engineering (Computer Engineering), Major Field: Computer Engineering, Department of Computer Engineering.
Thesis Advisor: Assistant Professor Kemathat Vibhatavanij, Ph.D. 121 pages.

This thesis has proposed the developing of RTP and RTCP protocol for embedded system. These protocols lets the embedded system be able to send / receive real-time data on IP network. That capability is a main function for all communication devices that used for sending / receiving voices and images data on the IP network. In this research, The RTP and RTCP protocols are developed under system call format, on μClinux operating system. This operating system is popular with the embedded system, which uses processing unit that dose not have MMU inside. Developing the system call format that works on kernel-mode of the operating system, helps the RTP and RTCP protocols are able to work in the embedded system rapidly and efficiently. The performance is better than an API in a library of the RTP and RTCP protocols, which works on user-mode of the operating system. Moreover, this encourages developers to bring the system call to develop applications or higher protocols, which are used for sending / receiving real-time data on the IP network for working with the embedded system in the future.

_____ _____ ___ / ___ / ___

Student's signature Thesis Advisor's signature

# กิตติกรรมประกาศ

ข้าพเจ้าขอกราบขอบพระคุณอาจารย์เขมะฑัต วิภาตะวนิช ประธานกรรมการที่ช่วยให้ความรู้และข้อเสนอแนะต่างๆ ทั้งในเรื่องของงานวิจัยและความรู้รอบตัวต่างๆ ซึ่งทำให้งานวิจัยชิ้นนี้สำเร็จเรียบร้อยลงด้วยดีและขอบคุณความห่วงใยและเอาใจใส่ที่อาจารย์ให้ข้าพเจ้าตลอดมา

ขอกราบขอบพระคุณบิดา มารดาและครอบครัวของข้าพเจ้าที่ให้กำลังใจและเป็นแรงกระตุ้นให้ข้าพเจ้าต่อสู้กับปัญหาและอุปสรรคต่างๆ จนสำเร็จและมีวันนี้ได้

ขอขอบคุณพี่ๆ ที่ NECTEC ที่คอยให้ความช่วยเหลือในทุกๆ ด้าน โดยเฉพาะคุณทวีศักดิ์ วชิราวรรณ ที่คอยให้คำปรึกษาและช่วยเหลือข้าพเจ้าสำหรับการติดตั้งระบบปฏิบัติการ μClinux ลงบน NBC board สุดท้ายขอบพระคุณทุกท่านที่มีส่วนในงานวิจัยนี้ทั้งที่ได้กล่าวถึงและไม่ได้กล่าวถึงไว้ ณ ที่นี้ด้วย

ศศิน จันทร์พวงทอง

กุมภาพันธ์ 2550

# สารบัญ

# สารบัญตาราง

# สารบัญภาพ

# สารบัญภาพ (ต่อ)

**ภาพที่** **หน้า**



**ภาพผนวกที่**

# สารบัญภาพ (ต่อ)

# สารบัญภาพ (ต่อ)

# คำอธิบายสัญลักษณ์และคำย่อ

| | | |
|---|---|---|
| RTP | = | Real-time Transport Protocol |
| RTCP | = | Real-time Transport Control Protocol |
| MMU | = | Memory Management Unit |
| IP | = | Internet Protocol |
| NBC | = | Net Base Camp |
| API | = | Application Programming Interface |
| TCP | = | Transport Control Protocol |
| UDP | = | User Datagram Protocol |
| RTCP SR | = | RTCP Sender Report |
| RTCP RR | = | RTCP Receiver Report |
| RTCP SDES | = | RTCP Source Description |
| CNAME | = | Canonical Name |
| PC | = | Personal Computer |
| ROM | = | Read-Only Memory |
| GUI | = | Graphic User Interface |
| NFS | = | Network File System |
| ms | = | millisecond |
| VoIP | = | Voice over Internet Protocol |
| IPv4 | = | Internet Protocol version 4 |
| IPv6 | = | Internet Protocol version 6 |

# การพอร์ต μClinux และการพัฒนาโปรโตคอล RTP/RTCP บนระบบแบบฝังตัว

# A Porting of μClinux and a Development of RTP/RTCP Protocol on an Embedded System

## คำนำ

### ความเป็นมาและความสำคัญของปัญหา

อุปกรณ์สื่อสารไม่ว่าจะเป็นรูปแบบใดก็ตาม จำเป็นต้องดำเนินการตามโปรโตคอลบางอย่าง เพื่อให้สามารถสื่อสารกันได้เข้าใจทั้งฝ่ายรับและฝ่ายส่ง ซึ่งการจะดำเนินการให้ข้อมูลที่รับหรือส่งผ่านอุปกรณ์สื่อสารเป็นไปตามลักษณะของโปรโตคอลที่ต้องการ อุปกรณ์สื่อสารต้องอาศัยทั้งหน่วยประมวลผล, หน่วยความจำ, อินเตอร์เฟสของเน็ตเวิร์ค, และคอนโทรลเลอร์ต่างๆ เข้ามาทำงานร่วมกัน ส่งผลให้อุปกรณ์สื่อสารมีลักษณะคล้ายกับระบบคอมพิวเตอร์ เพียงแต่ภายในจะมีเฉพาะทรัพยากรที่จำเป็นสำหรับการรับส่งและประมวลผลข้อมูลให้เป็นไปตามรูปแบบของโปรโตคอลที่ต้องการเท่านั้น ดังนั้นอุปกรณ์สื่อสารโดยทั่วไปจึงมีลักษณะเป็นระบบคอมพิวเตอร์ซึ่งทำงานเฉพาะอย่างที่เรียกว่า ระบบคอมพิวเตอร์แบบฝังตัว

ระบบคอมพิวเตอร์แบบฝังตัวส่วนใหญ่ประกอบด้วยทรัพยากรหลายๆ ประเภททำงานอยู่ร่วมกัน จึงต้องอาศัยซอฟต์แวร์ในลักษณะของระบบปฏิบัติการทำหน้าที่ควบคุมการทำงานของทรัพยากรทั้งหมดบนระบบคอมพิวเตอร์แบบฝังตัวให้เป็นไปอย่างมีประสิทธิภาพ และเป็นเสมือนตัวกลางให้ผู้ใช้สามารถติดต่อใช้งานทรัพยากรที่มีอยู่บนระบบคอมพิวเตอร์แบบฝังตัวได้สะดวกยิ่งขึ้น นอกจากนี้เมื่อผู้ใช้พัฒนาแอพพลิเคชั่นเข้าไปทำงานกับระบบคอมพิวเตอร์แบบฝังตัว ระบบปฏิบัติการจะช่วยควบคุมให้แอพพลิเคชั่นนั้นๆ ทำงานร่วมกับระบบและแอพพลิเคชั่นอื่นๆ ได้อย่างราบรื่น ยกตัวอย่างเช่น ช่วยควบคุมการจองพื้นที่หน่วยความจำของแอพพลิเคชั่นแต่ละตัวเพื่อให้ใช้งานหน่วยความจำที่มีอยู่ภายในระบบได้อย่างคุ้มค่ามากที่สุด, ช่วยควบคุมการติดต่อกันระหว่างโปรเซสแต่ละโปรเซสของแอพพลิเคชั่น, รวมไปถึงควบคุมการเข้าใช้งานหน่วยประมวลผลของโปรเซสของแอพพลิเคชั่นแต่ละตัว เพื่อให้แอพพลิเคชั่นทั้งหมดสามารถทำงานไปพร้อมๆ กันได้อย่างมีประสิทธิภาพและใช้หน่วยประมวลผลได้อย่างคุ้มค่า เป็นต้น ซึ่งการควบคุมเหล่านี้เป็นเรื่องที่ค่อนข้างซับซ้อน ถ้าผู้ใช้ต้องควบคุมการทำงานของแอพพลิเคชั่นภายในระบบ

คอมพิวเตอร์แบบฝังตัวด้วยตนเองทั้งหมดจะมีความยุ่งยากมาก ซึ่งระบบปฏิบัติการหนึ่งที่นิยมนำมาใช้งานบนระบบคอมพิวเตอร์แบบฝังตัวมากในปัจจุบันก็คือ ระบบปฏิบัติการ μClinux เนื่องจากระบบปฏิบัติการดังกล่าวสามารถทำงานอยู่บนระบบคอมพิวเตอร์แบบฝังตัวซึ่งใช้หน่วยประมวลผลแบบที่ไม่มี MMU อยู่ภายในได้เป็นอย่างดี

ในปัจจุบันมีความนิยมสื่อสารข้อมูลประเภท real-time บนเครือข่าย IP มากยิ่งขึ้นทั้งข้อมูลประเภทภาพและเสียง เนื่องจากข้อมูลประเภทดังกล่าวทำให้ผู้ใช้งานสามารถโต้ตอบกับผู้อื่นภายในระบบเครือข่ายได้ทันที แต่การที่อุปกรณ์สื่อสารจะสามารถขนส่งข้อมูลประเภท real-time บนเครือข่าย IP ได้นั้นจะต้องมีโปรโตคอลสำหรับขนส่งข้อมูลประเภทดังกล่าวทำงานอยู่ภายใน ซึ่งโปรโตคอลที่ทำหน้าที่ขนส่งข้อมูลประเภท real-time บนเครือข่าย IP ก็คือโปรโตคอล RTP และ RTCP โดยโปรโตคอลทั้งสองจะทำงานร่วมกันเพื่อให้การขนส่งข้อมูลประเภท real-time บนเครือข่าย IP เป็นไปอย่างมีประสิทธิภาพ (Mitra, 2001) (Goode, 2002)

เนื่องจากโปรโตคอล RTP และ RTCP ไม่ได้เป็นโปรโตคอลหลักในการขนส่งข้อมูลบนเครือข่าย IP เหมือนกับโปรโตคอล TCP, UDP, หรือ IP แต่เป็นโปรโตคอลที่ได้รับการออกแบบมาเพื่อเสริมการทำงานของโปรโตคอลหลักสำหรับการขนส่งข้อมูลบนเครือข่าย IP ให้สามารถขนส่งข้อมูลประเภท real-time ได้อย่างมีประสิทธิภาพ ดังนั้นระบบปฏิบัติการส่วนใหญ่จึงไม่ได้จัดเตรียมโปรโตคอล RTP และ RTCP ให้ใช้งานโดยอัตโนมัติ ซึ่งในปัจจุบันการพัฒนาโปรโตคอล RTP และ RTCP จะอยู่ในรูปแบบของแอพพลิเคชั่นสำเร็จรูปหรือเป็น library ที่รวบรวม API สำหรับการทำงานของโปรโตคอลดังกล่าวเอาไว้ภายใน เพื่อให้ผู้ใช้งานสามารถนำเอา API เหล่านี้ไปพัฒนาเป็นแอพพลิเคชั่นที่ตรงตามความต้องการได้อย่างสะดวก

library ของโปรโตคอล RTP และ RTCP ที่มีให้ใช้งานส่วนใหญ่ได้รับการออกแบบมาให้ทำงานอยู่บน user-mode ระบบปฏิบัติการ Linux เป็นหลัก ดังนั้นอาจจะไม่สามารถนำ library ดังกล่าวมาใช้งานบนuser-mode ของระบบปฏิบัติการ μClinux สำหรับระบบคอมพิวเตอร์แบบฝังตัวได้โดยตรง เนื่องจากลักษณะการทำงานและองค์ประกอบต่างๆ ภายในระบบปฏิบัติการทั้งสองมีความแตกต่างกันมากพอสมควร (โดยเฉพาะการทำงานที่เกี่ยวกับ MMU) จึงทำให้ API บางตัวภายใน library ของโปรโตคอล RTP และ RTCP ที่สามารถทำงานได้อย่างถูกต้องบนระบบปฏิบัติการ Linux อาจจะไม่สามารถทำงานได้หรือได้ผลลัพธ์ไม่ถูกต้องเมื่อนำ API ดังกล่าวมาใช้งานบนระบบปฏิบัติการ μClinux ซึ่งเป็นเรื่องยุ่งยากพอสมควรที่ผู้ใช้งานจะต้องตรวจสอบ,

ทดลอง, เลือก, และแก้ไข API ต่างๆ ภายใน library ของโปรโตคอล RTP และ RTCP ที่ได้รับการพัฒนาขึ้นมาสำหรับใช้งานบนระบบปฏิบัติการ Linux ให้สามารถนำไปทำงานบนระบบปฏิบัติการ μClinux ได้อย่างถูกต้อง

ดังนั้นงานวิจัยชิ้นนี้จึงได้ทำการพัฒนา system call ของโปรโตคอล RTP และ RTCP ที่เหมาะสมสำหรับการทำงานอยู่บน kernel-mode ของระบบปฏิบัติการ μClinux โดยเฉพาะ ซึ่งจะช่วยให้ผู้ที่ต้องการใช้งานโปรโตคอล RTP และ RTCP สำหรับพัฒนาแอพพลิเคชั่นหรือโปรโตคอลระดับสูงที่เกี่ยวข้องกับการขนส่งข้อมูลประเภท real-time บนเครือข่าย IP เพื่อนำไปใช้งานบนระบบปฏิบัติการ μClinux ภายในระบบคอมพิวเตอร์แบบฝังตัวได้รับความสะดวกมากยิ่งขึ้น เนื่องจาก system call จะได้รับการติดตั้งลงไปบน kernel ของระบบปฏิบัติการโดยอัตโนมัติตั้งแต่ขั้นตอนการคอมไพล์และ build kernel ของระบบปฏิบัติการ ดังนั้นเมื่อผู้ใช้งานมี system call ของโปรโตคอล RTP และ RTCP พร้อมให้เรียกใช้งานอยู่ภายใน kernel ของระบบปฏิบัติการ μClinux แล้วก็ไม่จำเป็นต้องติดตั้ง library ของโปรโตคอล RTP และ RTCP ลงไปบน user-mode ของระบบปฏิบัติการ μClinux อีก นอกจากนี้การพัฒนาการทำงานของโปรโตคอล RTP และ RTCP ในลักษณะของ system call ที่ทำงานอยู่บน kernel-mode ของระบบปฏิบัติการยังสามารถทำงานได้อย่างรวดเร็วและมีประสิทธิภาพมากกว่า API ของโปรโตคอลดังกล่าวที่ทำงานอยู่บน user-mode ของระบบปฏิบัติการ

# วัตถุประสงค์

1. เพื่อทดสอบว่าระบบคอมพิวเตอร์แบบฝังตัวซึ่งใช้หน่วยประมวลผลแบบที่ไม่มี MMU อยู่ภายในสามารถรองรับการทำงานของโปรโตคอล RTP และ RTCP ได้หรือไม่

2. เพื่อพัฒนา system call ของโปรโตคอล RTP และ RTCP ขึ้นมาทำงานอยู่บน kernel-mode ของระบบปฏิบัติการ μClinux โดยเฉพาะ

3. เพื่อให้ system call ของโปรโตคอล RTP และ RTCP ที่พัฒนาขึ้นช่วยอำนวยความสะดวกให้กับผู้ที่ต้องการพัฒนาแอพพลิเคชั่นหรือโปรโตคอลระดับสูงสำหรับขนส่งข้อมูลแบบ real-time บนเครือข่าย IP ขึ้นมาใช้งานบนระบบปฏิบัติการ μClinux ภายในระบบคอมพิวเตอร์แบบฝังตัว

4. เพื่อหา codec ที่เหมาะสมซึ่งจะทำให้โปรโตคอล RTP และ RTCP สามารถทำงานบนระบบคอมพิวเตอร์แบบฝังตัวได้อย่างถูกต้อง

5. เพื่อตรวจสอบประสิทธิภาพการทำงานของ system call ที่พัฒนาขึ้นเทียบกับ API ที่นำมาใช้เป็นต้นแบบในการพัฒนา system call ดังกล่าว

6. เพื่อเป็นพื้นฐานในการพัฒนาระบบคอมพิวเตอร์แบบฝังตัวซึ่งใช้หน่วยประมวลผลแบบที่ไม่มี MMU อยู่ภายในให้กลายเป็นโทรศัพท์ IP ราคาถูกต่อไป

# การตรวจเอกสาร

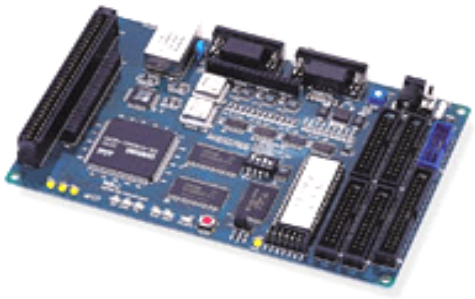
## ระบบคอมพิวเตอร์แบบฝังตัว

ระบบคอมพิวเตอร์แบบฝังตัวเป็นระบบคอมพิวเตอร์ที่ได้รับการออกแบบมาให้ทำงานเฉพาะอย่าง ดังนั้นภายในระบบคอมพิวเตอร์แบบฝังตัวจะประกอบด้วยทรัพยากรเท่าที่จำเป็นสำหรับการทำงานตามที่ได้กำหนดไว้เท่านั้น ทำให้ระบบคอมพิวเตอร์แบบฝังตัวแต่ละแบบจะมีลักษณะแตกต่างกันออกไปตามการนำไปใช้งาน ตัวอย่างของระบบคอมพิวเตอร์แบบฝังตัวในปัจจุบัน ได้แก่ Router, เครื่องเล่นเกมประเภทต่างๆ, อุปกรณ์ควบคุมภายในรถยนต์, รวมไปถึงอุปกรณ์สื่อสารประเภทต่างๆ เป็นต้น

ระบบคอมพิวเตอร์แบบฝังตัวในปัจจุบันจะใช้ microcontroller สำหรับควบคุมการทำงาน ซึ่งภายใน microcontroller จะมีทั้งหน่วยประมวลผล, หน่วยความจำ, และอินเตอร์เฟสต่างๆ ประกอบรวมเข้าด้วยกันภายใน 1 chipset เพื่อใช้ควบคุมการทำงานเฉพาะอย่าง ส่งผลให้ระบบคอมพิวเตอร์แบบฝังตัวมีขนาดเล็กและจะทำงานได้ตามลักษณะของ microcontroller ที่เลือกใช้เท่านั้น สำหรับ microcontroller นั้นสามารถแบ่งออกได้เป็น 2 รูปแบบตามลักษณะของหน่วยประมวลผลภายในที่เลือกใช้ ดังนี้ แบบที่ 1 คือ แบบที่เลือกใช้หน่วยประมวลผลซึ่งมี MMU อยู่ภายใน ทำให้ microcontroller ที่ใช้หน่วยประมวลผลในลักษณะนี้เหมาะกับงานที่มีขนาดใหญ่และมีความซับซ้อนในการประมวลผลค่อนข้างมาก แต่จะส่งผลให้ microcontroller มีราคาสูง แบบที่ 2 คือ แบบที่เลือกใช้หน่วยประมวลผลซึ่งไม่มี MMU อยู่ภายใน microcontroller ที่ใช้หน่วยประมวลผลลักษณะนี้จะมีราคาถูกและสามารถประมวลได้อย่างรวดเร็ว แต่จะเหมาะกับงานที่มีความซับซ้อนในการประมวลผลไม่มากนัก

ในปัจจุบันมีระบบคอมพิวเตอร์แบบฝังตัวให้ใช้งานหลายรูปแบบ มีทั้งแบบที่ผลิตสำเร็จรูปมาจากบริษัทและแบบที่นำอุปกรณ์ต่างๆ มาประกอบขึ้นเองตามความต้องการ ซึ่งภายในระบบคอมพิวเตอร์แบบฝังตัวจะมีอุปกรณ์อะไรอยู่บ้างนั้นขึ้นอยู่กับวัตถุประสงค์การนำไปใช้งานเป็นหลัก ในงานวิจัยนี้เป็นการพัฒนาโปรโตคอล RTP/RTCP ขึ้นมาทำงานบนระบบคอมพิวเตอร์แบบฝังตัว เพื่อเป็นพื้นฐานในการพัฒนาระบบคอมพิวเตอร์แบบฝังตัวให้กลายเป็นโทรศัพท์ IP ราคาประหยัดต่อไปในอนาคต เมื่อเน้นให้มีราคาประหยัดระบบคอมพิวเตอร์แบบฝังตัวที่ใช้จึงควรเป็นแบบที่ใช้หน่วยประมวลผลซึ่งไม่มี MMU อยู่ภายใน อีกทั้งเมื่อมีการทำงานทางด้านเน็ตเวิร์ค

ร่วมกับโปรโตคอลต่างๆ เพื่อการรับส่งข้อมูลบนเครือข่าย IP บนระบบคอมพิวเตอร์แบบฝังตัวจึงจำเป็นต้องมีอินเตอร์เฟสของ Ethernet อยู่ด้วย งานวิจัยชิ้นนี้จึงเลือกใช้ระบบคอมพิวเตอร์แบบฝังตัวที่มีชื่อว่า NBC board (ได้รับการสนับสนุนจาก NECTEC) ที่ภายในใช้ microcontroller Samsung S3C4530A ซึ่งเป็น microcontroller ที่ได้รับการออกแบบมาสำหรับควบคุมงานทางด้านเน็ตเวิร์คบนเครือข่าย Ethernet โดยเฉพาะ ทำให้ microcontroller นี้มีอินเตอร์เฟสของ Ethernet ความเร็ว 10/100 mbps อยู่ภายใน และใช้หน่วยประมวลผลเป็น ARM7TDMI ซึ่งไม่มี MMU อยู่ภายในทำให้ microcontroller ดังกล่าวสามารถทำงานได้อย่างรวดเร็วและมีราคาถูก



**ภาพที่ 1** NBC board ระบบคอมพิวเตอร์แบบฝังตัวที่ใช้ในงานวิจัยนี้

## ระบบปฏิบัติการ µClinux

ปัจจุบันมีผู้พัฒนาระบบปฏิบัติการขึ้นมาใช้งานกับระบบคอมพิวเตอร์แบบฝังตัวมากมายหลายรูปแบบ แต่จะสามารถแบ่งออกเป็น 2 ประเภทใหญ่ๆ ตามวัตถุประสงค์การใช้งาน (Lingxia Wang, Yao, Yang and Zhu, -) คือระบบปฏิบัติการที่ใช้กับงานเฉพาะทาง เช่น IOS ที่ใช้งานบน Router ของ Cisco เป็นต้น และระบบปฏิบัติการที่ใช้กับงานทั่วไป เช่น Windows CE, Linux เป็นต้น โดยระบบปฏิบัติการแบบที่ใช้กับงานทั่วไปที่ได้รับความนิยมในการนำมาใช้กับระบบคอมพิวเตอร์แบบฝังตัวมากในปัจจุบันคือ ระบบปฏิบัติการ Linux เนื่องจากระบบปฏิบัติการ Linux เป็นซอฟต์แวร์แบบ open source ทำให้ผู้ใช้สามารถปรับเปลี่ยน, เพิ่มเติม, รวมไปถึงแก้ไขระบบปฏิบัติการให้ตรงตามความต้องการได้ด้วยตนเอง อีกทั้งภายในระบบปฏิบัติการ Linux ยังมีเครื่องมือต่างๆ ที่ใช้ในการพัฒนาแอพพลิเคชั่นและดูแลระบบอย่างครบถ้วน โดยที่ผู้ใช้ไม่จำเป็นต้องเสียเวลาพัฒนาสิ่งเหล่านี้ขึ้นมาใช้งานเอง ซึ่งจะช่วยให้ผู้ใช้สามารถพัฒนาแอพพลิเคชั่น

ต่างๆ ขึ้นมาทำงานกับระบบคอมพิวเตอร์แบบฝังตัวได้อย่างสะดวก และทำให้ผู้ใช้สามารถดูแลและควบคุมการทำงานต่างๆ ของระบบคอมพิวเตอร์แบบฝังตัวได้ดียิ่งขึ้น

เนื่องจาก Kernel ของระบบปฏิบัติการ Linux ต้องทำงานกับหน่วยประมวลผลที่มี MMU อยู่ภายในเท่านั้น จึงสามารถนำเอาระบบปฏิบัติการ Linux มาใช้บนระบบคอมพิวเตอร์แบบฝังตัวที่ใช้หน่วยประมวลผลที่มี MMU อยู่ภายในได้ แต่ไม่สามารถนำมาใช้งานกับระบบคอมพิวเตอร์แบบฝังตัวที่ใช้หน่วยประมวลผลที่ไม่มี MMU อยู่ภายใน จึงมีการพัฒนาระบบปฏิบัติการที่มีลักษณะคล้ายคลึงกับระบบปฏิบัติการ Linux ขึ้นมา แต่ให้สามารถทำงานอยู่บนระบบคอมพิวเตอร์แบบฝังตัวที่ใช้หน่วยประมวลผลซึ่งไม่มี MMU อยู่ภายในได้อย่างมีประสิทธิภาพ โดยระบบปฏิบัติการที่พัฒนาขึ้นมาใหม่นี้มีชื่อว่า microcontroller Linux (μClinux) (Yaghmour, 2003)

ระบบปฏิบัติการ μClinux เป็น distribute ที่ตัดเอาส่วนการทำงานกับ MMU ออกจาก Kernel ของระบบปฏิบัติการ Linux เพื่อให้เหมาะสมกับการทำงานบนระบบคอมพิวเตอร์แบบฝังตัวที่ใช้หน่วยประมวลผลซึ่งไม่มี MMU อยู่ภายใน ส่งผลให้ขนาด kernel ของระบบปฏิบัติการ μClinux หลังจากคอมไพล์แล้วมีขนาดเล็ก ทำให้ช่วยประหยัดพื้นที่หน่วยความจำภายในระบบคอมพิวเตอร์แบบฝังตัวสำหรับการจัดเก็บและสำหรับใช้ในการทำงานของระบบปฏิบัติการ μClinux ถึงแม้ว่า kernel ของระบบปฏิบัติการ μClinux จะไม่มีส่วนที่ทำงานกับ MMU แต่ระบบปฏิบัติการ μClinux ยังคงคุณสมบัติเด่นๆ ของระบบปฏิบัติการ Linux ได้อย่างครบถ้วน เช่น รองรับ file system หลายประเภท, รองรับการทำงานกับโปรโตคอลในกลุ่ม TCP/IP รวมไปถึงโปรโตคอลใหม่ๆ ที่มีใช้งานกันอย่างแพร่หลายในปัจจุบัน เป็นต้น ด้วยคุณสมบัติเหล่านี้ทำให้มีผู้นิยมนำระบบปฏิบัติการ μClinux มาใช้งานกับระบบคอมพิวเตอร์แบบฝังตัวที่ใช้หน่วยประมวลผลซึ่งไม่มี MMU อยู่ภายในอย่างกว้างขวาง ทำให้ในปัจจุบันระบบปฏิบัติการ μClinux สามารถทำงานรองรับกับระบบคอมพิวเตอร์แบบฝังตัวหลายๆ platform เช่น ARM, ColdFire, Axis ETRAX, Motorola 6800 เป็นต้น (Nikkanen, 2003)

นอกจากการปรับเปลี่ยนภายใน Kernel แล้ว ยังได้มีการพัฒนา library ของภาษา C ที่มีชื่อว่า uClibc สำหรับนำมาใช้งานบนระบบปฏิบัติการ μClinux โดยเฉพาะ ภายใน uClibc ได้รวบรวม API หลักๆ มาจาก glibc ซึ่งเป็น library มาตรฐานที่มีใช้งานอยู่บนระบบปฏิบัติการ Linux โดยได้ตัดทอน API ที่ทำงานกับ MMU ออกไปให้เหลือเฉพาะ API ที่จำเป็นสำหรับการทำงานกับระบบคอมพิวเตอร์แบบฝังตัวที่ใช้หน่วยประมวลผลซึ่งไม่มี MMU อยู่ภายในเท่านั้น ทำให้ uClibc มี

ขนาดเล็กและสามารถทำงานร่วมกับระบบปฏิบัติการ μClinux ได้อย่างมีประสิทธิภาพ อีกทั้งการที่ uClibc ใช้ API จาก glibc ทำให้สามารถใช้เอกสารที่เขียนเป็นคู่มือในการใช้งาน API ต่างๆ ภายใน glibc มาเป็นคู่มือในการใช้งาน API ที่มีอยู่ภายใน uClibc ได้ และถึงแม้ว่าในขณะนี้ uClibc จะมี API ต่างๆ ไม่เท่ากับที่มีอยู่ใน glibc ซึ่งใช้กันอยู่ในระบบปฏิบัติการ Linux แต่ภายใน μClinux ได้จัดเตรียม library เสริมบางส่วนเอาไว้ให้สามารถใช้ร่วมกับ uClibc ในการคอมไพล์และ build แอพพลิเคชั่นส่วนใหญ่ที่มีใช้งานอยู่บนระบบปฏิบัติการ Linux ให้มาทำงานอยู่บนระบบปฏิบัติการ μClinux ได้เป็นอย่างดี (Nikkanen, 2003)

นอกจากมี uClibc ที่เป็น library ซึ่งรวบรวม API ต่างๆ ของภาษา C สำหรับใช้งานกับระบบปฏิบัติการ μClinux แล้ว ยังจำเป็นต้องมีเครื่องมือที่ทำงานร่วมกับ library uClibc เพื่อใช้ในการ build และคอมไพล์แอพพลิเคชั่นต่างๆ ที่จะนำไปใช้งานบนระบบปฏิบัติการ μClinux รวมไปถึงใช้ในการ build และคอมไพล์ image ของระบบปฏิบัติการ μClinux ขึ้นมาก่อนจะนำไปติดตั้งใช้งานบนระบบคอมพิวเตอร์แบบฝังตัวต่อไป ซึ่งได้มีผู้พัฒนาเครื่องมือเหล่านี้ในลักษณะ cross-tool เพื่อความสะดวกในการใช้งาน โดยสามารถนำ cross-tool ไปติดตั้งบนระบบปฏิบัติการ Linux และใช้เครื่องมือที่อยู่ภายใน cross-tool ในการ build หรือคอมไพล์แอพพลิเคชั่นรวมไปถึง image ของระบบปฏิบัติการ μClinux ตาม platform ที่เราต้องการบนระบบปฏิบัติการ Linux ได้ทันที โดยในปัจจุบันมี cross-tool สำหรับระบบปฏิบัติการ μClinux หลากหลาย platform ให้ใช้งาน เช่น arm-elf-toolchain สำหรับ ARM platform, m68k-elf-toolchain สำหรับ Motorola 6800 platform เป็นต้น ซึ่งเครื่องมือต่างๆ ที่อยู่ภายใน cross-tool ส่วนใหญ่ดัดแปลงมาจาก GNU tool ที่เป็นกลุ่มของเครื่องมือสำหรับใช้ในการพัฒนาแอพพลิเคชั่นบนระบบปฏิบัติการ Linux เช่น GNU gcc (C/C++ คอมไพเลอร์), GNU binutils (binary tool ต่างๆ เช่น Linker, Assembler) เป็นต้น (Nikkanen, 2003) เพื่อให้เหมาะสมในการนำไปใช้งานกับแต่ละ platform โดยยังคงให้วิธีการใช้งานเครื่องมือต่างๆ ภายใน cross-tool มีลักษณะใกล้เคียงกับวิธีการใช้งานเครื่องมือภายใน GNU tool ซึ่งช่วยให้ใช้งานเครื่องมือเหล่านี้ได้อย่างสะดวก และสามารถนำเอกสารวิธีการใช้งาน GNU tool ที่มีอยู่มากมายมาเป็นคู่มือในการใช้งานเครื่องมือภายใน cross-tool ได้ (Nikkanen, 2003)

เพื่อควบคุมและดูแลการทำงานของทรัพยากรและแอพพลิเคชั่นต่างๆ ภายใน NBC board ให้ทำงานร่วมกันได้อย่างถูกต้องและมีประสิทธิภาพ จึงได้ทำการติดตั้งระบบปฏิบัติการ μClinux ลงบน NBC board นอกจากนี้ระบบปฏิบัติการ μClinux ยังช่วยจัดเตรียมโปรโตคอลพื้นฐานในกลุ่ม TCP/IP ซึ่งใช้ในการขนส่งข้อมูลบนเครือข่าย IP เอาไว้ ทำให้สามารถพัฒนาโปรโตคอลระดับสูง

และแอพพลิเคชั่นต่างๆ สำหรับขนส่งข้อมูลบนเครือข่าย IP ได้สะดวกยิ่งขึ้น โดยขั้นตอนในการติดตั้งระบบปฏิบัติการ μClinux ลงบน NBC board สามารถศึกษาได้จากภาคผนวก ก

## โปรโตคอล RTP และ RTCP

ทั้งโปรโตคอล RTP และ RTCP จะมีการทำงานร่วมกัน เพื่อให้การรับส่งข้อมูลประเภท real-time บนเครือข่าย IP เป็นไปอย่างมีประสิทธิภาพมากที่สุด ซึ่งโปรโตคอลดังกล่าวมีหน้าที่หลักๆ ดังนี้

### 1. โปรโตคอล RTP

หน้าที่หลักของโปรโตคอล RTP คือจัดการในเรื่องการขนส่งข้อมูลประเภท real-time บนเครือข่าย IP ในทุกๆ ด้าน โดยเป็นโปรโตคอลที่เข้ามาช่วยเสริมการทำงานของโปรโตคอล UDP และโปรโตคอล IP ในการขนส่งข้อมูลประเภท real-time บนเครือข่าย IP ให้มีประสิทธิภาพมากที่สุด เช่น มีการสร้าง sequence number ให้กับ packet ของข้อมูลเพื่อใช้ตรวจสอบการสูญหายของ packet ในระหว่างการขนส่ง, ใส่ค่า timestamp ให้กับแต่ละ packet สำหรับใช้ในการทำ synchronize ระหว่างฝ่ายรับและฝ่ายส่ง รวมไปถึงใช้ในการทำ timing recovery ที่ฝ่ายรับข้อมูลเพื่อให้แอพพลิเคชั่นที่นำข้อมูลเหล่านี้ไปใช้งาน สามารถแสดงผลได้อย่างถูกต้องมากยิ่งขึ้น เป็นต้น (Perkins, 2003)( Schulzrinne, Casner, Frederick and Jacobson, 2003)

### 2. โปรโตคอล RTCP

เป็นโปรโตคอลที่ทำงานไปพร้อมๆ กับโปรโตคอล RTP แต่จะได้รับการกำหนดให้ทำงานอยู่คนละ port กับโปรโตคอล RTP ซึ่งปกติแล้ว port ที่กำหนดให้กับโปรโตคอลทั้ง 2 จะต่อเนื่องกันโดย port ที่เป็นเลขคู่จะได้รับการกำหนดให้กับ RTP packet ในขณะที่ port ถัดไปที่มีเลขสูงกว่า (เป็นเลขคี่) จะได้รับการกำหนดให้กับ RTCP packet หน้าที่หลักของโปรโตคอล RTCP คือสร้างรายงานในด้านต่างๆ เพื่อช่วยให้การทำงานของโปรโตคอล RTP เป็นไปอย่างมีประสิทธิภาพ ซึ่งรายงานที่สร้างขึ้นโดยโปรโตคอล RTCP มี 3 รูปแบบใหญ่ๆ ดังนี้ (Perkins, 2003) (Schulzrinne, Casner, Frederick and Jacobson, 2003)

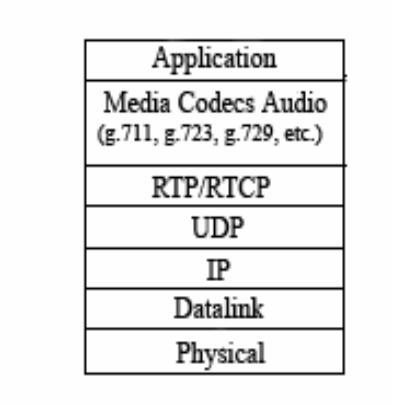
1) รายงานที่ใช้ในการปรับปรุงคุณภาพการขนส่งข้อมูลให้มีประสิทธิภาพมากยิ่งขึ้น โดยฝ่ายรับและฝ่ายส่งที่อยู่ภายใน session เดียวกันจะอาศัยโปรโตคอล RTCP ในการสร้างรายงานของค่าต่างๆ ที่จำเป็นสำหรับการนำไปปรับปรุงคุณภาพการขนส่งข้อมูลภายใน session นั้นๆ โดยฝ่ายส่งข้อมูลจะสร้าง RTCP SR ที่ภายในจะบอกถึงจำนวนของ packet และขนาดของข้อมูลทั้งหมดที่ฝ่ายส่งได้ส่งออกไปตั้งแต่เริ่มต้นการทำงาน แล้วจึงส่ง RTCP SR ที่สร้างขึ้นออกไปให้กับผู้รับข้อมูลทั้งหมดที่รับข้อมูลจากฝ่ายส่งนั้นๆ โดยตรง ในขณะที่ฝ่ายรับจะทำการสร้าง RTCP RR ที่ภายในจะบอกถึงค่า interarrival jitter, จำนวนของ packet ที่สูญหาย, ค่า loss fraction ที่แสดงจำนวนของ packet ที่สูญหายเทียบกับจำนวนของ packet ที่คาดว่าจะได้รับจริง เป็นต้น แล้วส่งออกไปให้กับผู้ส่งทั้งหมดที่ส่งข้อมูลเข้ามา ซึ่งทั้งฝ่ายรับและฝ่ายส่งจะนำข้อมูลที่ตนเองได้รับมาจากอีกฝ่ายหนึ่งไปประมวลผลเพื่อปรับปรุงคุณภาพการรับส่งข้อมูลถึงกันให้มีประสิทธิภาพมากขึ้น

2) รายงานที่ใช้ในการแสดงรายละเอียดของสมาชิกที่อยู่ภายใน session สมาชิกแต่ละคนภายใน session จะต้องสร้าง RTCP SDES ขึ้นมา ซึ่งภายในจะมีรายละเอียดของสมาชิกนั้นๆ เช่น ชื่อ, EMAIL address, หมายเลขโทรศัพท์ เป็นต้น แต่รายละเอียดที่สำคัญที่สุดที่จำเป็นต้องมีอยู่ภายใน RTCP SDES เสมอ คือ CNAME ที่เป็นเสมือน ID ใช้แสดงตัวตนของสมาชิกนั้นๆ ภายใน session หลังจากนั้นสมาชิกแต่ละคนจะต้องส่ง RTCP SDES ของตนเองออกไปให้กับสมาชิกอื่นๆ ภายใน session นั้นๆ ทั้งหมด เพื่อเป็นการยืนยันการมีตัวตนของสมาชิกนั้นๆ ภายใน session ซึ่งจะทำให้สมาชิกอื่นๆ ยอมรับ RTP และ RTCP packet ที่ส่งมาจากสมาชิกนี้ อีกทั้งยังช่วยให้สมาชิกแต่ละคนทราบถึงรายละเอียดและจำนวนสมาชิกทั้งหมดที่มีอยู่ภายใน session ณ ขณะเวลานั้น ซึ่งจะช่วยในการคำนวณค่าบางอย่างในการทำงานของโปรโตคอล RTP และ RTCP สำหรับสมาชิกแต่ละคนให้ถูกต้องตรงกันทั้ง session เช่น ช่วงเวลาถัดไปที่จะใช้ส่ง RTCP compound packet เป็นต้น

3) รายงานที่ใช้สำหรับควบคุมจำนวนสมาชิกภายใน session ในกรณีที่มีสมาชิกออกจาก session สมาชิกนั้นๆ จะต้องส่ง RTCP BYE ไปให้กับสมาชิกอื่นๆ ภายใน session นั้นๆ ทั้งหมด เพื่อเป็นการยืนยันและแจ้งให้สมาชิกอื่นๆ ได้ทราบว่าสมาชิกนี้กำลังจะออกจาก session แล้ว ซึ่งจะทำให้สมาชิกอื่นๆ ไม่ยอมรับ RTP และ RTCP packet ใดๆ ที่ส่งมาโดยสมาชิกนี้ อีกทั้งยังช่วยให้สมาชิกอื่นๆ ปรับเปลี่ยนค่าบางอย่างในการทำงานของโปรโตคอล RTP และ RTCP ให้ถูกต้องตรงกับจำนวนสมาชิกที่ลดลงจาก session ต่อไป

โดยปกติแล้วรายงานของโปรโตคอล RTCP จะถูกส่งออกไปเป็นช่วงๆ ขึ้นอยู่กับจำนวนของสมาชิกและอัตราส่วนของสมาชิกที่ทำหน้าที่เป็นผู้ส่งข้อมูลภายใน session ณ ขณะเวลานั้น รวมไปถึงขนาดของ bandwidth ที่กำหนดให้กับโปรโตคอล RTCP ภายใน session นั้น ซึ่งถ้าค่าเหล่านี้มีการเปลี่ยนแปลงจะต้องทำการคำนวณช่วงในการส่งรายงานของโปรโตคอล RTCP ใหม่เสมอ สำหรับค่าของช่วงเวลาที่ต่ำที่สุดที่ใช้ในการส่งรายงานของโปรโตคอล RTCP คือ 5 วินาที นอกจากนี้รายงานของโปรโตคอล RTCP แต่ละอันไม่ได้ส่งแยกเป็น packet เดี่ยวๆ แต่จะนำรายงานทั้งหมดที่ต้องการส่งออกไป ณ ขณะเวลานั้น รวมเข้ามาเป็น packet เพียง packet เดียวในลักษณะของ RTCP compound packet เพื่อประหยัด bandwidth ภายใน session (Perkins, 2003)

โปรโตคอล RTP และ RTCP จะทำงานอยู่บนโปรโตคอล UDP/IP ซึ่งเป็นโปรโตคอลหลักที่ใช้ในการขนส่งข้อมูลบนเครือข่าย IP โดยโปรโตคอล RTP และ RTCP นอกจากจะมีหน้าที่หลักในการขนส่งข้อมูลประเภท real-time ไปบนเครือข่าย IP แล้วยังทำหน้าที่สร้างความน่าเชื่อถือในการขนส่งข้อมูลประเภท real-time บนเครือข่ายที่ไม่มีความน่าเชื่อถืออีกด้วย เนื่องจากโปรโตคอล UDP/IP เป็นโปรโตคอลที่ทำงานในลักษณะ connectionless ซึ่งไม่มีกลไกในการสร้างการเชื่อมต่อระหว่างฝ่ายส่งและฝ่ายรับข้อมูลก่อนเริ่มการรับและส่งข้อมูลจริงๆ อีกทั้งในระหว่างการขนส่งข้อมูลระหว่างกันจะไม่มีการยืนยันความถูกต้องว่าฝ่ายรับได้รับข้อมูลครบหรือไม่ และถ้าฝ่ายรับรับข้อมูลไม่ครบก็จะไม่มีการส่งข้อมูลชุดนั้นซ้ำ แต่การที่โปรโตคอล UDP/IP ไม่มีกลไกเหล่านี้ทำให้สามารถขนส่งข้อมูลได้อย่างรวดเร็ว ซึ่งเหมาะกับข้อมูลประเภท real-time ดังนั้นเมื่อโปรโตคอลที่ใช้ในการขนส่งข้อมูลแบบ real-time บนเครือข่าย IP ไม่มีความน่าเชื่อถือเพียงพอ โปรโตคอล RTP และ RTCP จึงได้รับการพัฒนาขึ้นมาเพื่อเพิ่มความน่าเชื่อถือที่ End-point แทน ซึ่ง End-point ในที่นี้ก็คือผู้รับและผู้ส่งข้อมูลนั่นเอง โดยโปรโตคอล RTP และ RTCP จะช่วยเพิ่มความฉลาดให้กับ End-point ในการตรวจสอบการสูญหายของข้อมูล รวมไปถึงปรับเปลี่ยนการแสดงผลของข้อมูลให้ถูกต้องตามข้อมูลที่ได้รับเข้ามา ซึ่งจะทำให้การขนส่งข้อมูลประเภท real-time บนเครือข่าย IP มีความน่าเชื่อถือมากยิ่งขึ้น (Perkins, 2003) ความสัมพันธ์ของโปรโตคอลในแต่ละระดับชั้นสำหรับการขนส่งข้อมูลประเภท real-time บนเครือข่าย IP เป็นดังรูปที่ 2 (Kurose and Ross, 2002)




**ภาพที่ 2** แสดงความสัมพันธ์ของโปรโตคอลในแต่ละระดับชั้น ที่ใช้ในการขนส่งข้อมูลแบบ real-time บนเครือข่าย IP

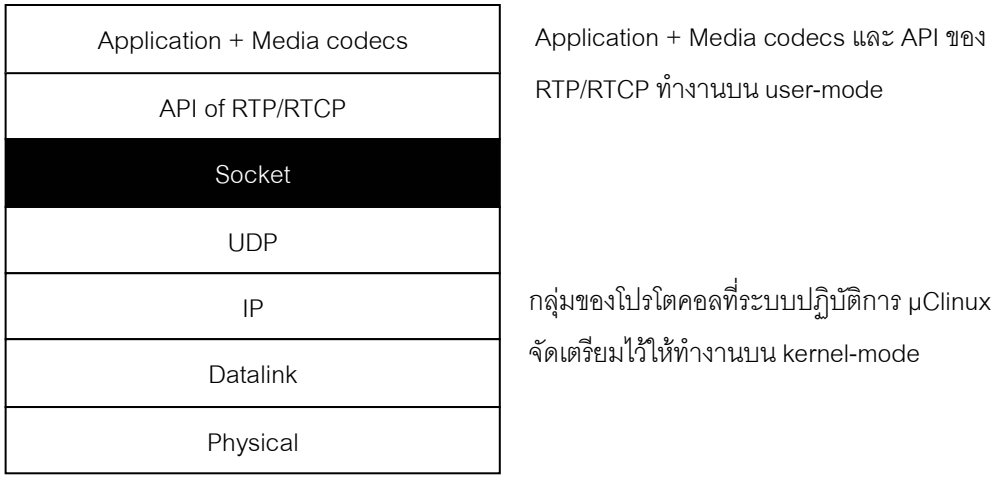
หลักการทำงานเบื้องต้นของโปรโตคอล RTP และ RTCP ในการขนส่งข้อมูลแบบ real-time บนเครือข่าย IP นั้น เริ่มต้นเมื่อฝ่ายรับและฝ่ายส่งต้องการขนส่งข้อมูลถึงกัน จะต้องทำการสร้างช่องทางขนส่งข้อมูลที่เรียกว่า session (IP address และ port) ขึ้นมาเชื่อมต่อระหว่างทั้ง 2 ฝ่าย โดยทั้งคู่จะต้องทำการสุ่มค่าขึ้นมาเพื่อใช้เป็นเสมือน ID ประจำตัวชั่วคราวซึ่งเรียกว่า Synchronization Source (SSRC) ในการใช้แสดงความมีตัวตนภายใน session นั้นๆ หลังจากนั้นฝ่ายส่งจะนำข้อมูลที่ได้รับมาจาก media codec มาสร้างเป็น RTP packet แล้วเริ่มส่งออกไปให้ฝ่ายรับผ่านทาง session ที่ได้สร้างขึ้นทันที ในช่วงนี้ถึงแม้ว่าฝ่ายรับจะได้รับ RTP packet ที่ฝ่ายส่งส่งมาให้ แต่จะละทิ้ง packet เหล่านั้นทั้งหมด เนื่องจากฝ่ายรับยังไม่ได้รับ RTCP SDES ที่ใช้แสดงความมีตัวตนของฝ่ายส่ง ทำให้ฝ่ายรับไม่สามารถยอมรับ RTP packet ที่ส่งมาโดยฝ่ายส่งนี้ได้ จนกระทั่งถึงรอบเวลาที่ฝ่ายส่งต้องทำการส่ง RTCP compound packet ไปให้กับฝ่ายรับ เมื่อฝ่ายรับได้รับ RTCP compound packet ที่ส่งมาโดยฝ่ายส่งและมี RTCP SDES ที่ใช้ระบุตัวตนของฝ่ายส่งอยู่ภายใน ฝ่ายรับจะทำการเก็บค่า SSRC และ CNAME ที่อยู่ภายใน RTCP SDES เอาไว้ และจะยอมรับ RTP packet ที่ส่งมาโดยฝ่ายส่งนี้ทันที กระบวนการดังกล่าวเป็นเสมือนการยืนยันการเชื่อมต่อระหว่างฝ่ายรับและฝ่ายส่งก่อนเริ่มทำการขนส่งข้อมูลกันจริงๆ นั่นเอง ในช่วงนี้ทั้งฝ่ายรับและฝ่ายส่งสามารถขนส่งข้อมูล RTP packet ถึงกันได้ รวมทั้งจะมีการส่ง RTCP compound packet ออกไปเป็นช่วงๆ (ปกติประมาณ 5 วินาที ต่อ 1 packet) ซึ่งทั้งฝ่ายรับและฝ่ายส่งจะใช้ค่าที่อยู่ภายใน RTCP compound packet ที่ตนเองได้รับไปใช้ในการปรับปรุงคุณภาพการขนส่งข้อมูลระหว่างกัน และเมื่อฝ่ายส่งทำการส่งข้อมูลให้กับฝ่ายรับเรียบร้อยแล้ว และต้องการออกจาก session ฝ่ายรับจะทำการสร้าง RTCP BYE แล้วส่งออกไปให้กับฝ่ายรับ เมื่อฝ่ายรับได้รับ RTCP BYE จากฝ่ายส่ง ฝ่ายรับจะ

ยังคงรับข้อมูลที่ส่งมาโดยฝ่ายส่งนั้นๆ อีกครู่หนึ่ง ก่อนจะทำการลบ SSRC และ CNAME ออกไป และจะไม่ยอมรับ RTP packet ที่ส่งมาโดยฝ่ายส่งนั้นๆ อีกเลย กระบวนดังกล่าวเป็นเสมือนการยืนยันให้ยกเลิกการเชื่อมต่อระหว่างฝ่ายรับและฝ่ายส่งนั่นเอง (Perkins, 2003)

**ลักษณะของโปรโตคอล RTP และ RTCP ที่มีใช้งานในปัจจุบัน**

เนื่องจากโปรโตคอล RTP และ RTCP ไม่ได้เป็นโปรโตคอลหลักที่ใช้ในการขนส่งข้อมูลบนเครือข่าย IP เหมือนกับโปรโตคอล TCP, UDP และ IP แต่เป็นโปรโตคอลที่เข้ามาเสริมให้การขนส่งข้อมูลแบบ real-time บนเครือข่าย IP เป็นไปอย่างมีประสิทธิภาพมากยิ่งขึ้น ดังนั้นระบบปฏิบัติการส่วนใหญ่จึงไม่ได้จัดเตรียมโปรโตคอลทั้งสองเอาไว้ให้ใช้งาน ในกรณีที่ต้องการใช้งานโปรโตคอล RTP และ RTCP ผู้ใช้งานจำเป็นต้องอาศัยแอพพลิเคชั่นสำเร็จรูปหรือ library ของโปรโตคอล RTP และ RTCP ที่มีผู้พัฒนาเอาไว้ มาติดตั้งบนระบบปฏิบัติการอีกทีหนึ่ง โดยภายใน library จะมี API ที่เกี่ยวข้องกับการทำงานของโปรโตคอล RTP และ RTCP ให้ผู้ใช้งานสามารถนำไปใช้ในการพัฒนาเป็นแอพพลิเคชั่นสำหรับขนส่งข้อมูลแบบ real-time บนเครือข่าย IP ได้ต่อไป

API ของโปรโตคอล RTP และ RTCP ที่มีใช้งานอยู่ในปัจจุบันส่วนใหญ่ทำงานอยู่บน user-mode ของระบบปฏิบัติการ ส่งผลให้ API ที่ทำหน้าที่เกี่ยวกับการขนส่ง RTP และ RTCP packet ออกไปยังเครือข่าย IP จำเป็นต้องทำการสร้างอินเทอร์เฟสของ socket เพื่อใช้เสมือนเป็นทางผ่านของข้อมูล (RTP และ RTCP packet) ต่อเชื่อมเข้ากับโปรโตคอล UDP และ IP ซึ่งเป็นกลุ่มโปรโตคอลมาตรฐานที่ระบบปฏิบัติการเป็นผู้จัดเตรียมเอาไว้ให้สำหรับการขนส่งข้อมูลบนเครือข่าย IP ลักษณะการทำงานของ API บน user-mode ของระบบปฏิบัติการ แสดงดังรูปที่ 3 (Stevens, Fenner and Rudoff, 2004)




**ภาพที่ 3** แสดงการทำงานของ API ของโปรโตคอล RTP และ RTCP บน user-mode ของระบบปฏิบัติการ

Library ของโปรโตคอล RTP และ RTCP ที่นิยมนำมาใช้งานได้แก่ RTPLib, Common Library ของ UCL, ccRTP, jrtplib, ortp, และ RTP library ของ Vovida ซึ่ง API ที่อยู่ภายใน library เหล่านี้ทำงานอยู่บน user-mode ของระบบปฏิบัติการทั้งหมดและได้รับการพัฒนาขึ้นมาจากภาษา C หรือ C++ เพื่อการทำงานที่รวดเร็ว โดยการจะนำเอา API ภายใน library เหล่านี้ไปใช้งานได้นั้น ผู้ใช้งานจำเป็นต้องติดตั้ง library ที่ต้องการลงบนระบบปฏิบัติการก่อน จึงสามารถใช้ API ภายใน library ดังกล่าวไปพัฒนาเป็นแอพพลิเคชั่นที่ตรงตามความต้องการต่อไป

Library ของโปรโตคอล RTP และ RTCP ที่มีให้ใช้งานอยู่ในปัจจุบันส่วนใหญ่ได้รับการพัฒนาขึ้นมาเพื่อทำงานกับระบบปฏิบัติการ Linux บนเครื่อง PC เป็นหลัก ทำให้การนำ library เหล่านี้มาใช้งานบนระบบคอมพิวเตอร์แบบฝังตัวค่อนข้างยุ่งยากพอสมควร เนื่องจากผู้ใช้จำเป็นต้องเขียน Makefile ขึ้นมาใหม่เพื่อให้เรียกใช้เครื่องมือ (cross-tool) รวมถึง library (uClibc, library เสริมอื่นๆ) ให้ถูกต้องสำหรับการคอมไพล์และติดตั้ง library ของโปรโตคอล RTP และ RTCP ให้สามารถใช้งานบนระบบปฏิบัติการ μClinux (Mecklenburg, 2004) นอกจากนี้ library ของโปรโตคอล RTP และ RTCP บางตัวอาจจะไม่สามารถนำมาใช้งานร่วมกับระบบปฏิบัติการ μClinux ได้ เนื่องจากภายใน library ดังกล่าวอาจจะมี API บางตัวที่ต้องทำงานกับ MMU หรืออาจจะมีการเรียกใช้ API บางอย่างซึ่ง uClibc รวมทั้งระบบปฏิบัติการ μClinux ไม่ได้จัดเตรียมเอาไว้ให้ ซึ่งส่งผลให้แอพพลิเคชั่นที่พัฒนาขึ้นมาจาก API เหล่านี้ไม่สามารถทำงานบนระบบปฏิบัติการ μClinux ได้ สิ่งต่างๆ เหล่านี้เป็นภาระต่อผู้ที่ต้องการใช้งาน library ของ

โปรโตคอล RTP และ RTCP บนระบบปฏิบัติการ μClinux พอสมควร โดยผู้ใช้จำเป็นต้องทำการตรวจสอบ, ทดลอง, และแก้ไขบางส่วนภายใน library ก่อนที่จะนำ library ดังกล่าวไปใช้งานบนระบบปฏิบัติการ μClinux ได้ต่อไป

# อุปกรณ์และวิธีการ

## อุปกรณ์

### ฮาร์ดแวร์

1. เครื่องคอมพิวเตอร์ 2 ชุดสำหรับต่อกับ NBC board
   - Pentium 4 1.2 GHz
   - หน่วยความจำขนาด 256 MB
   - Serial port (RS-232)
   - อินเทอร์เฟสเน็ตเวิร์คสำหรับต่อ LAN
2. เครื่องคอมพิวเตอร์ที่ใช้ในการพัฒนาแอพพลิเคชั่นสำหรับการทดลองทั้งหมด รวมถึงใช้ในการคอมไพล์และ build image ของระบบปฏิบัติการ μClinux สำหรับ NBC board
   - Pentium 4 1.7 GHz
   - หน่วยความจำขนาด 768 MB
   - อินเทอร์เฟสเน็ตเวิร์คสำหรับต่อ LAN
3. NBC Board 2 ชุด
   - microcontroller Samsung S3C4530A หน่วยประมวลผลความเร็ว 25 MHz
   - อินเตอร์เฟส Ethernet 10/100 mbps
   - Serial port (RS-232)
4. สาย RS-232 2 สาย
5. หม้อแปลงสำหรับ NBC board 2 ชุด

### ซอฟต์แวร์

1. ระบบปฏิบัติการ Windows XP สำหรับฮาร์ดแวร์ในข้อ 1 และ 2
2. ระบบปฏิบัติการ Linux Redhat 9 สำหรับฮาร์ดแวร์ในข้อ 2
3. ระบบปฏิบัติการ μClinux สำหรับ NBC board

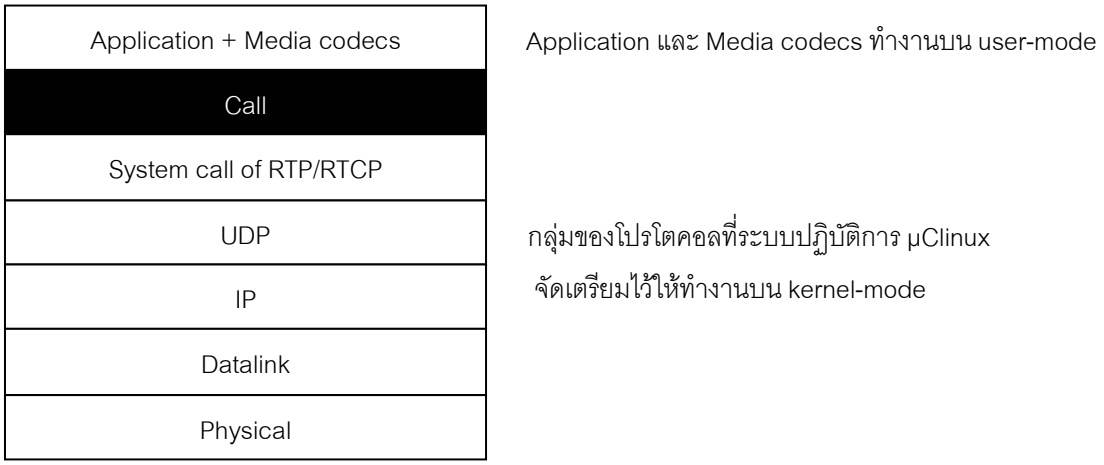
4. โปรแกรม Hyperterminal สำหรับให้ผู้ใช้งานสามารถสื่อสารกับ NBC board ผ่านทางเครื่องคอมพิวเตอร์ได้

5. library ของโปรโตคอล RTP และ RTCP ที่ API ภายในทำงานอยู่บน user-mode ของระบบปฏิบัติการ

## วิธีการ

งานวิจัยชิ้นนี้เป็นการพัฒนา system call หลักๆ ของโปรโตคอล RTP และ RTCP ขึ้นมาทำงานบน kernel-mode ของระบบปฏิบัติการ μClinux เพื่อให้ผู้ใช้งานสามารถนำเอา system call เหล่านี้ไปพัฒนาเป็นแอพพลิเคชั่นหรือโปรโตคอลระดับสูงต่างๆ ที่เกี่ยวข้องกับการขนส่งข้อมูลประเภท real-time บนเครือข่าย IP สำหรับใช้งานบนระบบปฏิบัติการ μClinux โดยเฉพาะ ซึ่งการพัฒนา system call ดังกล่าวจะอาศัยอัลกอริทึมและหลักการทำงานจาก API ต่างๆ ภายใน library ของโปรโตคอล RTP และ RTCP ที่ทำงานอยู่บน user-mode ของระบบปฏิบัติการมาเป็นต้นแบบ โดย library ของโปรโตคอล RTP และ RTCP ที่จะนำมาใช้เป็นต้นแบบได้นั้น API ทั้งหมดภายใน library ดังกล่าวต้องสามารถนำไปใช้พัฒนาแอพพลิเคชั่นสำหรับทำงานอยู่บนระบบปฏิบัติการ μClinux ได้อย่างถูกต้องเท่านั้น ที่ต้องเลือก library ในลักษณะนี้มาใช้เป็นต้นแบบ เนื่องจาก API ทั้งหมดภายใน library ดังกล่าวจะมีการเรียกใช้เฉพาะ API อื่นๆ ที่ uClibc และระบบปฏิบัติการ μClinux ได้จัดเตรียมเอาไว้ให้เท่านั้น ซึ่ง API เหล่านี้ (API ที่ uClibc และระบบปฏิบัติการ μClinux ได้จัดเตรียมเอาไว้) มีพื้นฐานการทำงานมาจากการเรียกใช้ system call ต่างๆ ภายใน kernel ของระบบปฏิบัติการ μClinux อีกต่อหนึ่งนั่นเอง ด้วยเหตุนี้ถ้า library ใดที่ API ภายในทั้งหมดสามารถนำไปพัฒนาเป็นแอพพลิเคชั่นที่ใช้งานอยู่บนระบบปฏิบัติการ μClinux ได้อย่างถูกต้อง ก็แสดงว่าภายใน kernel ของระบบปฏิบัติการ μClinux ได้จัดเตรียม system call ที่จำเป็นสำหรับการทำงานของ API ทั้งหมดภายใน library นั้นๆ อย่างครบถ้วน ดังนั้นจึงสามารถปรับเปลี่ยน API ทั้งหมดภายใน library ดังกล่าวให้กลายเป็น system call ที่ทำงานอยู่บน kernel-mode ของระบบปฏิบัติการได้ โดยเปลี่ยนจากการเรียกใช้ API ต่างๆ ไปเป็นเรียกใช้ system call ที่ API เหล่านั้นเรียกใช้งานแทน ซึ่งจากวิธีนี้จะทำให้ได้ system call ของโปรโตคอล RTP และ RTCP ที่มีการทำงานเหมือนกับ API ต้นแบบเกือบทุกประการ และสามารถนำไปใช้ในการพัฒนาแอพพลิเคชั่นสำหรับทำงานอยู่บนระบบปฏิบัติการ μClinux ได้ต่อไป

การพัฒนาการทำงานของโปรโตคอล RTP และ RTCP ให้อยู่ในรูปแบบของ system call นี้จะช่วยให้การทำงานของโปรโตคอล RTP และ RTCP สามารถทำงานบนระบบปฏิบัติการ μClinux ได้รวดเร็วและมีประสิทธิภาพมากยิ่งขึ้น เมื่อเปรียบเทียบกับ API ที่นำมาใช้เป็นต้นแบบซึ่งทำงานอยู่บน user-mode ของระบบปฏิบัติการ เนื่องจาก system call ทำงานอยู่บน kernel-mode ของระบบปฏิบัติการจึงสามารถเข้าถึงทรัพยากรต่างๆ ได้อย่างรวดเร็ว นอกจากนี้การพัฒนาการทำงานของโปรโตคอล RTP และ RTCP ให้อยู่ในรูปแบบของ system call ยังช่วยให้ผู้ที่ต้องการพัฒนาแอพพลิเคชั่นหรือโปรโตคอลระดับสูงต่างๆ ที่เกี่ยวข้องกับการรับส่งข้อมูลประเภท real-time บนเครือข่าย IP สำหรับทำงานบนระบบปฏิบัติการ μClinux ได้รับความสะดวกมากยิ่งขึ้น เนื่องจากสามารถกำหนดให้ติดตั้ง module ของ system call สำหรับโปรโตคอล RTP และ RTCP ลงไปบน kernel ของระบบปฏิบัติการ μClinux ได้ตั้งแต่ขั้นตอนการคอมไพล์และ build image ของระบบปฏิบัติการ μClinux ขึ้นมา ส่งผลให้ image ของระบบปฏิบัติการ μClinux ที่คอมไพล์และ build ออกมาเรียบร้อยแล้ว จะมี system call ของโปรโตคอล RTP และ RTCP อยู่ภายในให้สามารถเรียกใช้งานได้ทันที โดยที่ผู้ใช้งานไม่จำเป็นต้องติดตั้ง library อื่นใดลงไปบน user-mode ของระบบปฏิบัติการ μClinux เพิ่มเติม ลักษณะการทำงานของ system call ของโปรโตคอล RTP และ RTCP ที่จะพัฒนาขึ้นมาใช้งานเป็นดังรูปที่ 4




**ภาพที่ 4** แสดงลักษณะการทำงานของ system call ของโปรโตคอล RTP และ RTCP ที่พัฒนาขึ้น

จากรูปที่ 4 พบว่าระบบปฏิบัติการ μClinux จะเป็นผู้จัดเตรียมฟังก์ชั่นการทำงานต่างๆ ของโปรโตคอล RTP และ RTCP ในลักษณะของ system call ที่ทำงานอยู่บน kernel-mode ของระบบปฏิบัติการ ผู้ใช้งานที่ต้องการพัฒนาแอพพลิเคชั่นหรือโปรโตคอลระดับสูงต่างๆ ที่เกี่ยวข้องกับการรับส่งข้อมูลประเภท real-time บนเครือข่าย IP สำหรับทำงานบนระบบปฏิบัติการ μClinux สามารถพัฒนาโดยเรียกใช้ system call เหล่านี้ได้ทันที

**ขั้นตอนในการพัฒนา system call บน kernel-mode ของระบบปฏิบัติการ μClinux**

1) เลือก library ของโปรโตคอล RTP และ RTCP เพื่อใช้เป็นต้นแบบในการพัฒนาเป็น system call

จากที่กล่าวไว้ก่อนหน้านี้ว่าการพัฒนา system call ของโปรโตคอล RTP และ RTCP ที่ทำงานอยู่บน kernel-mode ของระบบปฏิบัติการ μClinux ในงานวิจัยชิ้นนี้ จะอาศัยอัลกอริทึมและหลักการทำงานจาก API ต่างๆ ภายใน library ของโปรโตคอล RTP และ RTCP ที่ทำงานอยู่บน user-mode ของระบบปฏิบัติการมาเป็นต้นแบบ โดยกำหนดว่า library ที่จะนำมาใช้เป็นต้นแบบได้นั้น API ทั้งหมดภายใน library ดังกล่าวต้องสามารถนำไปพัฒนาแอพพลิเคชั่นสำหรับทำงานบนระบบปฏิบัติการ μClinux ได้อย่างถูกต้องเท่านั้น ดังนั้นขั้นตอนแรกในการเริ่มพัฒนา system call ที่ต้องการก็คือหา library ของโปรโตคอล RTP และ RTCP ซึ่งมีลักษณะตรงตามเงื่อนไขที่กำหนดสำหรับนำมาใช้เป็นต้นแบบ

เนื่องจากในปัจจุบันยังไม่มีการพัฒนา library ของโปรโตคอล RTP และ RTCP ขึ้นมาทำงานกับระบบปฏิบัติการ μClinux โดยเฉพาะ จึงจำเป็นต้องอาศัย library ของโปรโตคอล RTP และ RTCP ที่มีใช้งานบนระบบปฏิบัติการ Linux สำหรับเครื่อง PC และน่าจะสามารถนำไปใช้งานบนระบบปฏิบัติการ μClinux ได้มาใช้เป็นต้นแบบในการพัฒนาเป็น system call ที่ต้องการแทน เมื่อตรวจสอบ library ของโปรโตคอล RTP และ RTCP ที่มีใช้งานอยู่บนระบบปฏิบัติการ Linux บนเครื่อง PC ในปัจจุบันพบว่ามีผู้พัฒนาเอาไว้มากมายหลายแบบ ซึ่ง library ดังกล่าวแต่ละแบบก็จะมีข้อดีข้อเสียแตกต่างกันออกไป หลังจากพิจารณาลักษณะและหน้าที่การทำงานของ API ทั้งหมดที่อยู่ภายในแต่ละ library แล้วจึงได้เลือกเอา Common library ของ UCL มาใช้เป็นต้นแบบ เนื่องจาก API ทั้งหมดที่อยู่ภายใน library นี้พัฒนาขึ้นมาจากภาษา C ทำให้เหมาะต่อการใช้งานบนระบบปฏิบัติการ μClinux มากกว่า library ที่ API ภายในพัฒนาขึ้นมาจากภาษา C++ เนื่องจาก

เครื่องมือและ library หลักต่างๆ สำหรับภาษา C (uClibc, และกลุ่มของ library เสริมที่ระบบปฏิบัติการ μClinux จัดเตรียมเอาไว้ให้) ที่ใช้ในการพัฒนาแอพพลิเคชั่นสำหรับทำงานบนระบบปฏิบัติการ μClinux มีความสมบูรณ์มากกว่าภาษา C++ ที่ยังคงอยู่ในขั้นตอนของการพัฒนาให้มีความสมบูรณ์มากขึ้น (ปีที่ทำการวิจัย : 2549) ทำให้ library ที่ API ทั้งหมดพัฒนาขึ้นมาจากภาษา C สามารถใช้เครื่องมือและ library ต่างๆ ภายในระบบปฏิบัติการ μClinux คอมไพล์และติดตั้งใช้งานบนระบบปฏิบัติการ μClinux ได้อย่างถูกต้องครบถ้วน อีกทั้งเมื่อพัฒนาแอพพลิเคชั่นขึ้นมาจาก API ภายใน library ดังกล่าวแล้วก็สามารถใช้เครื่องมือและ library ต่างๆ ภายในระบบปฏิบัติการ μClinux คอมไพล์และ build ขึ้นมาใช้งานบนระบบปฏิบัติการ μClinux ได้อย่างถูกต้องและสมบูรณ์กว่า library ที่ API ภายในทั้งหมดพัฒนาขึ้นมาจากภาษา C++ นอกจากนี้เมื่อตรวจสอบการทำงานของ API ทั้งหมดภายใน common library ของ UCL พบว่า ภายใน library ดังกล่าวมี API ที่ใช้ในการทำงานทั้งหมดทุกด้านของโปรโตคอล RTP และ RTCP อย่างครบถ้วนและ API ต่างๆ ดังกล่าวได้รับการพัฒนาขึ้นมาให้เป็นไปตามมาตรฐานการทำงานของโปรโตคอล RTP และ RTCP ทั้งหมด

2) ทดสอบการทำงานของ API ต่างๆ ภายใน Common library ของ UCL บนระบบปฏิบัติการ μClinux

หลังจากได้เลือก Common library ของ UCL สำหรับนำมาใช้เป็นต้นแบบในการพัฒนา system call ของโปรโตคอล RTP และ RTCP บนระบบปฏิบัติการ μClinux แล้ว ขั้นตอนต่อมาคือต้องทดสอบว่า API ต่างๆ ภายใน library ดังกล่าวสามารถนำมาใช้ในการพัฒนาแอพพลิเคชั่นสำหรับทำงานบนระบบปฏิบัติการ μClinux ได้หรือไม่ก่อนที่จะนำเอา API เหล่านั้นมาพัฒนาเป็น system call บนระบบปฏิบัติการ μClinux ต่อไป โดยการทดสอบจะเริ่มจากทำการคอมไพล์ API ทั้งหมดภายใน Common library ของ UCL ด้วยเครื่องมือและ library ต่างๆ ที่ระบบปฏิบัติการ μClinux จัดเตรียมไว้ให้ว่าสามารถคอมไพล์ได้อย่างถูกต้องหรือไม่ ซึ่งเมื่อทดลองคอมไพล์แล้วพบว่าสามารถคอมไพล์ API ทั้งหมดภายใน library ดังกล่าวได้สำเร็จ

ขั้นตอนต่อมาจึงนำเอา API ที่คอมไพล์ไว้แล้วเหล่านี้มาใช้ในการพัฒนาแอพพลิเคชั่นสำหรับรับ-ส่งและประมวลผล RTP และ RTCP packet ขึ้นมาทำงานบนระบบปฏิบัติการ μClinux เพื่อใช้ทดสอบการทำงานของ API ต่างๆ ดังกล่าวว่าสามารถทำงานอยู่บน user-mode ของระบบปฏิบัติการ μClinux ได้อย่างถูกต้องสมบูรณ์หรือไม่ โดยเมื่อพัฒนาแอพพลิเคชั่นดังกล่าว

ขึ้นมาเรียบร้อยแล้วจึงได้นำแอพพลิเคชั่นนั้นมาคอมไพล์และ build ด้วยเครื่องมือและ library หลักต่างๆ ภายในระบบปฏิบัติการ μClinux ซึ่งพบว่าสามารถคอมไพล์และ build แอพพลิเคชั่นดังกล่าวได้อย่างถูกต้อง โดยในขั้นตอนนี้จะมีการ link แอพพลิเคชั่นเข้ากับ API ต่างๆ ภายใน Common library ของ UCL ที่แอพพลิเคชั่นดังกล่าวเรียกใช้งาน และ API เหล่านั้นจะไป link รวมกับ API อื่นๆ ที่มีการเรียกใช้เพื่อทำงานขั้นพื้นฐานต่างๆ ภายใน API นั้นๆ เช่น การสร้าง socket, การจองพื้นที่หน่วยความจำ เป็นต้น ซึ่ง API สำหรับการทำงานขั้นพื้นฐานเหล่านี้จะอยู่ภายใน library หลักที่ระบบปฏิบัติการ μClinux จัดเตรียมไว้ให้ (uClibc และ library เสริมอื่นๆ) โดยถ้าสามารถคอมไพล์และ build แอพพลิเคชั่นที่พัฒนาขึ้นมาจาก API ต่างๆ ภายใน library ดังกล่าว (Common library ของ UCL) ด้วยเครื่องมือและ library หลักต่างๆ ที่ระบบปฏิบัติการ μClinux จัดเตรียมไว้ให้ได้สำเร็จโดยไม่มีข้อผิดพลาดใดๆ เกิดขึ้นก็แสดงว่า library ต่างๆ ที่ระบบปฏิบัติการ μClinux จัดเตรียมไว้ให้ทั้ง uClibc และ library เสริมอื่นๆ มี API ที่รองรับการเรียกใช้งานจาก API ทั้งหมดภายใน Common library ของ UCL อย่างครบถ้วน ซึ่งจะส่งผลให้สามารถนำเอา API ต่างๆ ทั้งหมดภายใน Common library ของ UCL มาใช้ในการพัฒนาแอพพลิเคชั่นสำหรับทำงานอยู่บน user-mode ของระบบปฏิบัติการ μClinux ได้ (Yaghmour, 2003)

หลังจากทำการคอมไพล์และ build แอพพลิเคชั่นสำหรับรับ-ส่งและประมวลผล RTP และ RTCP packet ที่พัฒนาขึ้นมาจาก API ภายใน Common library ของ UCL ด้วยเครื่องมือและ library หลักต่างๆ ซึ่งระบบปฏิบัติการ μClinux จัดเตรียมไว้ให้ ได้สำเร็จแล้ว ขั้นตอนต่อไปคือต้องนำเอาแอพพลิเคชั่นเหล่านั้นมาทดลองทำงานจริงบนระบบปฏิบัติการ μClinux เพื่อตรวจสอบการทำงานของแอพพลิเคชั่นดังกล่าวว่าสามารถทำงานบนระบบปฏิบัติการ μClinux ได้อย่างถูกต้องหรือไม่ เนื่องจาก API ทั้งหมดภายใน Common library ของ UCL ได้รับการออกแบบและพัฒนาขึ้นมาให้ทำงานกับระบบปฏิบัติการ Linux ซึ่ง API ภายใน library ดังกล่าวจะมีการเรียกใช้ API อื่นๆ ภายใน library หลักที่ระบบปฏิบัติการ Linux จัดเตรียมไว้ให้สำหรับการทำงานขั้นพื้นฐานภายใน API นั้นๆ ดังนั้นผลการทำงานของ API ทั้งหมดภายใน Common library ของ UCL จึงยึดตามผลการทำงานของ API สำหรับการทำงานขั้นพื้นฐานที่ระบบปฏิบัติการ Linux จัดเตรียมไว้ให้เป็นหลัก ด้วยเหตุนี้ถึงแม้ว่าระบบปฏิบัติการ μClinux จะจัดเตรียม API สำหรับการทำงานขั้นพื้นฐานเอาไว้ภายใน library ต่างๆ (uClibc และ library เสริมอื่นๆ) เพื่อให้ API ทั้งหมดใน Common library ของ UCL ได้เรียกใช้งานอย่างครบถ้วนแล้วก็ตาม แต่ถ้ามี API สำหรับการทำงานขั้นพื้นฐานบางตัวของระบบปฏิบัติการ μClinux มีลักษณะการทำงานและผลลัพธ์ที่ได้แตกต่างจาก API ตัวเดียวกันซึ่งทำงานอยู่บนระบบปฏิบัติการ Linux และ API ดังกล่าวได้รับการเรียกใช้งานโดย API

ตัวใดตัวหนึ่งภายใน Common library ของ UCL ก็จะส่งผลให้ API นั้นๆ ภายใน Common library ของ UCL ไม่สามารถทำงานได้อย่างถูกต้องบนระบบปฏิบัติการ μClinux และจะทำให้แอพพลิเคชั่นที่เรียกใช้งาน API ดังกล่าวมีการทำงานที่ผิดพลาดเกิดขึ้นเมื่อนำไปใช้งานบนระบบปฏิบัติการ μClinux อีกด้วย ดังนั้นจึงต้องทำการตรวจสอบผลการทำงานของแอพพลิเคชั่นที่พัฒนาขึ้นจาก API ต่างๆ ภายใน Common library ของ UCL บนระบบปฏิบัติการ μClinux ว่าสามารถทำงานได้อย่างถูกต้องและได้ผลลัพธ์ในการทำงานเหมือนกับผลการทำงานของแอพพลิเคชั่นแบบเดียวกันนี้ที่ทำงานอยู่บนระบบปฏิบัติการ Linux หรือไม่ โดยเมื่อนำแอพพลิเคชั่นดังกล่าวไปทดลองใช้งานจริงกับระบบปฏิบัติการ μClinux ภายใน NBC board จะพบว่าแอพพลิเคชั่นดังกล่าวสามารถทำงานได้อย่างถูกต้องโดยไม่มีข้อผิดพลาดใดๆ เกิดขึ้น และเมื่อนำผลลัพธ์การทำงานของแอพพลิเคชั่นดังกล่าวไปเปรียบเทียบกับผลการทำงานของแอพพลิเคชั่นเดียวกันนี้ที่ทำงานอยู่บนระบบปฏิบัติการ Linux ภายในเครื่อง PC จะพบว่าได้ผลลัพธ์ของการทำงานเหมือนกันทุกประการ จึงแสดงให้เห็นว่า API ทั้งหมดที่อยู่ภายใน Common library ของ UCL สามารถทำงานบน user-mode ของระบบปฏิบัติการ μClinux ได้อย่างถูกต้อง

3) พัฒนา system call ของโปรโตคอล RTP และ RTCP โดยอาศัยหลักการทำงานจาก API ทั้งหมดภายใน Common library ของ UCL

หลังจากตรวจสอบแล้วว่า API ทั้งหมดภายใน Common library ของ UCL สามารถทำงานอยู่บน user-mode ของระบบปฏิบัติการ μClinux ได้อย่างถูกต้อง จึงได้นำเอา API เหล่านั้นมาใช้เป็นต้นแบบในการพัฒนา system call ของโปรโตคอล RTP และ RTCP ที่ทำงานอยู่บน kernel-mode ของระบบปฏิบัติการ μClinux ต่อไป

วิธีการพัฒนา system call ของโปรโตคอล RTP และ RTCP ในงานวิจัยชิ้นนี้จะอาศัยหลักการทำงานจาก API ที่สามารถทำงานอยู่บน user-mode ของระบบปฏิบัติการ μClinux ได้อย่างถูกต้องมาเป็นต้นแบบ ซึ่งในที่นี้ได้เลือกใช้ API ต่างๆ จาก Common library ของ UCL ตามที่ได้กล่าวไปก่อนหน้านี้ โดยการที่ API ที่เลือกมาใช้เป็นต้นแบบสามารถทำงานอยู่บน user-mode ของระบบปฏิบัติการ μClinux ได้อย่างถูกต้องนั้น ก็แสดงว่า API สำหรับการทำงานขั้นพื้นฐานต่างๆ ที่ API ต้นแบบมีการเรียกใช้งานจาก library ซึ่งระบบปฏิบัติการ μClinux ได้จัดเตรียมไว้ให้สามารถทำงานได้อย่างถูกต้องและให้ผลลัพธ์ตามที่ API ต้นแบบเหล่านั้นต้องการ โดยการทำงานของ API สำหรับงานขั้นพื้นฐานดังกล่าวบน user-mode ของระบบปฏิบัติการก็จะมาจากการเรียกใช้ system

call ต่างๆ จาก kernel-mode ของระบบปฏิบัติการอีกต่อหนึ่งนั่นเอง ดังนั้นการปรับเปลี่ยนจาก API ต้นแบบที่ทำงานอยู่บน user-mode ของระบบปฏิบัติการให้กลายมาเป็น system call ที่ทำงานอยู่บน kernel-mode ของระบบปฏิบัติการจึงสามารถทำได้โดยเปลี่ยนจากการเรียกใช้ API สำหรับการทำงานขั้นพื้นฐานต่างๆ ภายใน API ต้นแบบให้กลายเป็น system call ที่ API สำหรับการทำงานขั้นพื้นฐานเหล่านั้นเรียกใช้งานโดยตรง และจะต้องเปลี่ยนการเรียกใช้ไฟล์ header ของ API ต้นแบบจากการเรียกใช้ไฟล์ header ที่อยู่บน user-mode ของระบบปฏิบัติการไปเป็นไฟล์ header ที่อยู่บน kernel-mode ของระบบปฏิบัติการแทน เพื่อให้สามารถคอมไพล์และติดตั้ง system call ที่พัฒนาขึ้นมาจาก API ต้นแบบดังกล่าวลงไปใช้งานบน kernel ของระบบปฏิบัติการ μClinux ได้อย่างถูกต้องต่อไป โดยเมื่อทำตามขั้นตอนเหล่านี้แล้วจะทำให้ได้ system call ที่มีลักษณะการทำงาน.ใกล้เคียงกับ API ที่นำมาใช้เป็นต้นแบบเกือบทุกประการ

หลังจากที่พัฒนา system call สำหรับโปรโตคอล RTP และ RTCP เรียบร้อยแล้ว ก่อนจะนำ system call ดังกล่าวไปติดตั้งและใช้งานบน kernel ของระบบปฏิบัติการ μClinux ได้นั้น จะต้องมีการแก้ไของค์ประกอบบางอย่างภายใน kernel ของระบบปฏิบัติการ μClinux เพิ่มเติม เพื่อให้ kernel ของระบบปฏิบัติการ μClinux สามารถทำงานรองรับกับ system call ใหม่นี้ได้อย่างถูกต้อง

4) ทดสอบการทำงานของ system call ต่างๆ บน kernel-mode ของระบบปฏิบัติการ μClinux

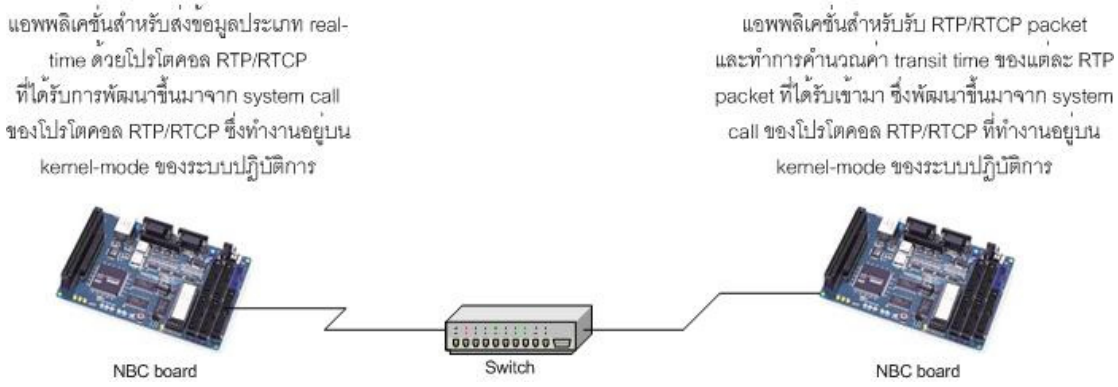
เมื่อทำการพัฒนาและติดตั้ง system call ของโปรโตคอล RTP และ RTCP ลงไปใช้งานบน kernel ของระบบปฏิบัติการ μClinux เรียบร้อยแล้วก็จะต้องทำการทดสอบการทำงานของ system call ทั้งหมดที่พัฒนาขึ้นมาว่าสามารถทำงานได้อย่างถูกต้องและให้ผลลัพธ์ในการทำงานเหมือนกับการทำงานของ API ที่ใช้เป็นต้นแบบซึ่งทำงานอยู่บน user-mode ของระบบปฏิบัติการ μClinux หรือไม่ จึงได้ทำการพัฒนาแอพพลิเคชั่นสำหรับรับ-ส่งและประมวลผล RTP และ RTCP packet ขึ้นมาจาก system call ดังกล่าว (สามารถตรวจสอบรายชื่อ หน้าที่ และวิธีการใช้งานของ system call ดังกล่าวได้จากภาคผนวก ข) โดยให้มีลักษณะการทำงานเหมือนกับแอพพลิเคชั่นที่พัฒนาขึ้นมาจาก API ต้นแบบในขั้นตอนที่ 2 ทุกประการ หลังจากนั้นจึงนำเอาแอพพลิเคชั่นดังกล่าวไปติดตั้งและทดลองทำงานจริงกับระบบปฏิบัติการ μClinux ภายใน NBC board ซึ่งจะพบว่าแอพพลิเคชั่นดังกล่าวสามารถทำงานได้อย่างถูกต้องโดยไม่มีข้อผิดพลาดใดๆ เกิดขึ้น และเมื่อนำผลลัพธ์ที่ได้จากการทำงานของแอพพลิเคชั่นดังกล่าวไปเปรียบเทียบกับผลการทำงานของแอพพลิเคชั่นแบบ

เดียวกันที่พัฒนาขึ้นมาจาก API ต้นแบบซึ่งทำงานอยู่บน user-mode ของระบบปฏิบัติการ μClinux (จากขั้นตอนที่ 2) จะพบว่าผลการทำงานของแอพพลิเคชั่นทั้ง 2 แบบ (ทั้งแบบที่พัฒนาขึ้นมาจาก system call และแบบที่พัฒนาขึ้นมาจาก API ต้นแบบ) มีลักษณะเหมือนกันทุกประการ จึงแสดงให้เห็นว่า system call ของโปรโตคอล RTP และ RTCP ที่พัฒนาขึ้นมานี้สามารถทำงานอยู่บน kernel ของระบบปฏิบัติการ μClinux ได้เป็นอย่างดีและมีลักษณะการทำงานใกล้เคียงกับ API ต้นแบบเกือบทุกประการ

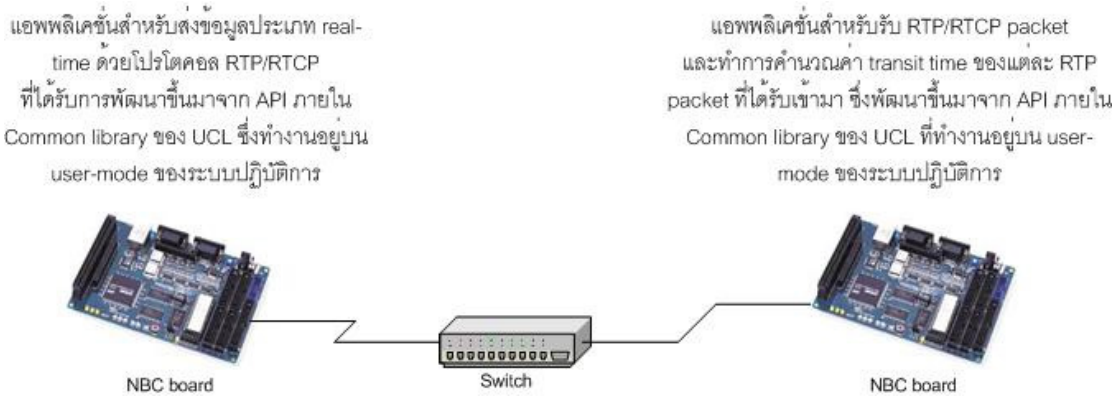
# ผลและวิจารณ์

## สภาพแวดล้อมในการทดลอง

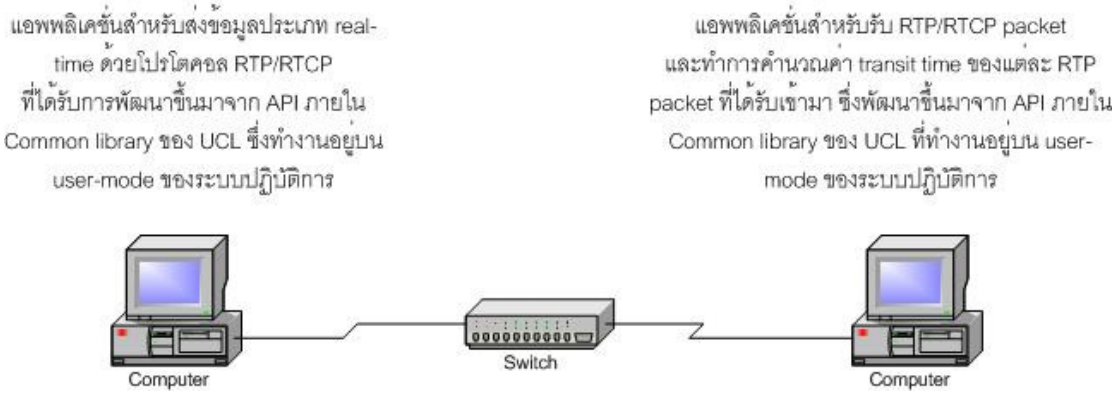
การวัดประสิทธิภาพการทำงานของ system call เปรียบเทียบกับ API ต้นแบบจาก Common Library ของ UCL ซึ่งทำงานอยู่บน user-mode ของระบบปฏิบัติการสามารถทำได้โดยพัฒนาแอพพลิเคชั่นสำหรับรับ-ส่งและประมวลผล RTP และ RTCP packet ขึ้นมา 3 ชุด ชุดที่ 1 พัฒนาขึ้นมาจาก system call, ชุดที่สองพัฒนาขึ้นมาจาก API ภายใน Common library ของ UCL โดยแอพพลิเคชั่นทั้งสองชุดนี้จะนำไปทดลองกับระบบปฏิบัติการ μClinux บน NBC board, และชุดที่ 3 จะพัฒนาขึ้นมาจาก API ภายใน Common library ของ UCL เช่นเดียวกับแอพพลิเคชั่นในชุดที่ 2 แต่แอพพลิเคชั่นในชุดที่ 3 นี้จะนำไปใช้งานกับระบบปฏิบัติการ Linux บนเครื่อง PC เพื่อใช้ทดสอบว่าทรัพยากรที่แตกต่างกันของ NBC board และเครื่อง PC มีผลต่อการทำงานของ API ต่างๆ หรือไม่ ซึ่งแอพพลิเคชั่นทั้ง 3 ชุดนี้จะต้องมีลักษณะการทำงานเหมือนกันทุกประการ โดยรูปแบบการทำงานของแอพพลิเคชั่นทั้ง 3 ชุดเป็นดังรูปที่ 5, 6, และ 7 ตามลำดับ




**ภาพที่ 5** แสดงการทำงานของแอพพลิเคชั่นในการทดลองชุดที่ 1




**ภาพที่ 6** แสดงการทำงานของแอพพลิเคชั่นในการทดลองชุดที่ 2




**ภาพที่ 7** แสดงการทำงานของแอพพลิเคชั่นในการทดลองชุดที่ 3

สำหรับการรับส่งข้อมูลประเภท real-time บนเครือข่าย IP ด้วยโปรโตคอล RTP และ RTCP นั้น ข้อมูลที่จะบรรจุลงใน RTP packet จะต้องได้รับการทำ encode ด้วย codec แบบใดแบบหนึ่งก่อนเสมอ ซึ่งในปัจจุบันมีมาตรฐานของ codec สำหรับข้อมูลประเภทเสียงให้เลือกใช้งานหลายแบบ โดยแต่ละแบบก็จะมีอัตราเร็วในการขนส่งข้อมูลแตกต่างกันออกไป ดังในตารางที่ 1

**ตารางที่ 1** แสดงลักษณะของ codec แบบต่างๆ ที่มีใช้งานอยู่ในปัจจุบัน ทั้งอัตราเร็ว, ขนาดของ payload, ระยะห่างในการส่งแต่ละ payload, และระยะห่างในการส่งแต่ละ payload ที่โปรแกรม Cisco CallManager ทำงานรองรับ (Cisco Systems, Inc, 2006)

| Codec | อัตราเร็ว (Kbps) | ค่า default สำหรับขนาดของ payload (Bytes) | ค่า default สำหรับระยะห่างในการส่งแต่ละ payload (ms) | ค่าระยะห่างในการส่งแต่ละ payload ที่โปรแกรม Cisco CallManager ทำงานรองรับ (ms) |
|---|---|---|---|---|
| G.711 | 64 | 160 | 20 | 10, 20, 30 |
| G.726 | 32 | 80 | 20 | - |
| G.728 | 16 | 60 | 30 | - |
| G.729 | 8 | 20 | 20 | 10, 20, 30, 40, 50, 60 |
| G.723.1 | 6.3 | 24 | 30 | 30, 60 |

ความสัมพันธ์ระหว่างแต่ละค่าในตารางเป็นดังสูตรต่อไปนี้

อัตราเร็ว = (ขนาดของ payload x 8) / ระยะห่างในการส่งแต่ละ payload

ถึงแม้ว่า codec แต่ละแบบจะมีอัตราเร็วในการขนส่งข้อมูลคงที่ แต่ก็สามารถรองรับ payload ที่มีขนาดและระยะห่างในการส่งแตกต่างกันได้ ขึ้นอยู่กับมาตรฐานของ codec เหล่านั้น แต่เมื่อนำขนาดของ payload และระยะห่างในการส่งดังกล่าวมาคำนวณตามสูตรด้านบนจะต้องได้ค่าเท่ากับอัตราเร็วของ codec นั้นๆ เสมอ ดังนั้นจึงสามารถใช้สูตรดังกล่าวในการหาค่าที่ต้องการใช้งานสำหรับ codec แบบต่างๆ ได้ เช่นโดยปกติโปรแกรม Cisco CallManager จะกำหนดค่าระยะห่างในการส่งแต่ละ payload สำหรับ codec แต่ละแบบที่โปรแกรมดังกล่าวสามารถทำงานรองรับได้มาให้เลือก ซึ่งถ้าเราต้องการทราบขนาดของ payload ที่ใช้งานคู่กับระยะห่างดังกล่าว ก็สามารถใช้สูตรด้านบนคำนวณได้ทันที เพื่อให้การส่งข้อมูลเป็นไปตามอัตราเร็วของ codec ที่ต้องการใช้งาน

ในการทดลองเพื่อหา codec ที่เหมาะสมสำหรับการทำงานบน NBC board นั้น จะใช้ codec ทั้งหมดที่มีอยู่ในตารางที่ 1 มาทำการทดลอง โดยนอกจากจะใช้ขนาดของ payload และระยะห่าง

ในการส่งแต่ละ payload ที่เป็นค่า default สำหรับ codec แบบต่างๆ แล้ว ยังนำเอาค่าระยะห่างในการส่งแต่ละ payload สำหรับ codec แบบต่างๆ ที่มีให้เลือกใช้งานอยู่ในโปรแกรม Cisco CallManager (คอลัมน์ที่ 5 ของตารางที่ 1) มาใช้ในการทดลองเพิ่มอีกด้วย เนื่องจากโปรแกรมดังกล่าวเป็นโปรแกรมสำหรับงาน VoIP ที่มีการใช้งานอย่างกว้างขวางในปัจจุบัน ดังนั้นค่าระยะห่างในการส่งแต่ละ payload สำหรับ codec แบบต่างๆ ที่มีให้เลือกใช้อยู่ในโปรแกรมดังกล่าวจึงเป็นเสมือนมาตรฐานที่สามารถยอมรับได้สำหรับการนำไปใช้งานจริง แต่ก่อนที่จะนำค่าระยะห่างดังกล่าวไปงานได้นั้นจะต้องใช้สูตรที่กล่าวไปก่อนหน้านี้ในการช่วยคำนวณค่าขนาดของ payload ที่สัมพันธ์กับระยะห่างนั้นๆ

โดยจะเริ่มทำการทดลองกับแอพพลิเคชั่นในการทดลองทั้ง 3 ชุด ด้วย codec แบบ G.711 (อัตราเร็ว 64 Kbps) ตามค่า default ของ codec ดังกล่าวก่อน (ขนาดของ payload เป็น 160 bytes และระยะห่างในการส่งแต่ละ payload เป็น 20 ms) เนื่องจาก codec G.711 มีอัตราเร็วในการขนส่งข้อมูลมากที่สุดซึ่งถ้าแอพพลิเคชั่นในการทดลองทั้ง 3 ชุดสามารถทำงานร่วมกับ codec ดังกล่าวได้อย่างถูกต้องแอพพลิเคชั่นเหล่านั้นก็จะสามารถรองรับการทำงานกับ codec ที่มีอัตราเร็วในการขนส่งข้อมูลต่ำกว่า 64 Kbps ได้เช่นกัน อีกทั้ง codec แบบ G.711 เป็น codec ที่ทำงานง่ายที่สุดทำให้สามารถตรวจสอบข้อผิดพลาดต่างที่เกิดขึ้นระหว่างการทำงานของแอพพลิเคชั่นได้ง่าย ซึ่งในการทดลองนี้จะนำเอาไฟล์ประเภท audio ที่ได้รับการทำ encode ด้วย codec แบบ G.711 มาใช้งาน โดยจะต้องกำหนดให้ฝ่ายส่งข้อมูล RTP packet ทำการอ่านข้อมูลจากไฟล์ดังกล่าวมาครั้งละ 160 bytes เพื่อมาสร้างเป็น RTP packet สำหรับส่งออกไปให้กับฝ่ายรับข้อมูลทุกๆ 20 ms (เพื่อให้ได้อัตราเร็วในการส่งข้อมูลเป็น 64 Kbps) ซึ่งในทางทฤษฎีฝ่ายรับข้อมูลควรจะได้รับ RTP packet 1 packet ทุกๆ 20 ms ด้วย ดังนั้นจึงได้กำหนดให้ฝ่ายรับข้อมูลทำการรับข้อมูลทุกๆ 20 ms เอาไว้ นอกจากนี้ที่ฝ่ายรับข้อมูลจะมีการเก็บค่า timestamp และ sequence number ของแต่ละ RTP packet ที่ได้รับเข้ามาเอาไว้เพื่อใช้ในการคำนวณค่า relative transit time ของแต่ละ RTP packet ซึ่งค่าดังกล่าวเป็นค่าที่สนใจในการทดลองนี้ เนื่องจากจะทำให้สามารถทราบได้ว่าการมาถึงของ RTP packet แต่ละ packet มีระยะห่างจาก packet ที่มาถึงก่อนหน้านี้มากน้อยเพียงใด สำหรับการคำนวณค่า relative transit time ของ RTP packet แต่ละ packet จะใช้สูตรดังนี้

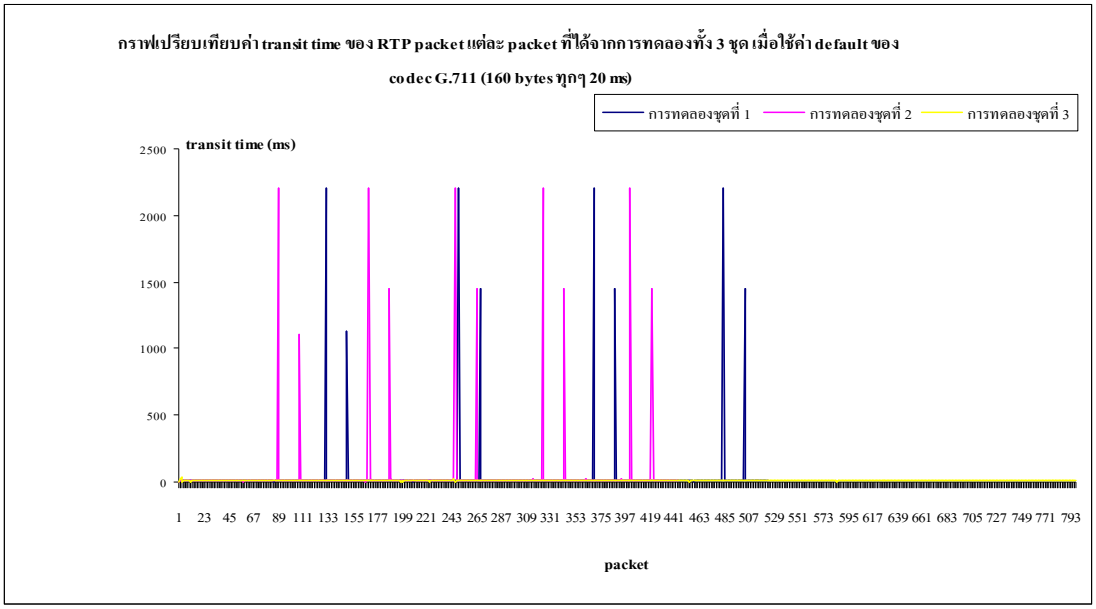
$$D(i, j) = (R_j - R_i) - (S_j - S_i)$$

- D(i,j) คือ ค่า relative transit time ระหว่าง RTP packet ที่ i และ j มีหน่วยเป็น ms
- $R_i$ และ $R_j$ คือเวลาที่ packet i และ j มาถึงยังฝั่งผู้รับ ตามลำดับ มีหน่วยเป็น ms

- $S_i$ และ $S_j$ เป็น ค่า RTP timestamp ภายใน packet ที่ i และ j ตามลำดับ โดยจะต้องนำค่าของ RTP timestamp ดังกล่าวไปคูณกับอัตราการสร้างข้อมูลตามชนิดของ encoding ที่ใช้ ซึ่งในการทดลองนี้ใช้ codec แบบ G.711 จึงต้องคูณค่าของ RTP timestamp ในแต่ละ packet ด้วยค่า 1/8 เพื่อแปลงหน่วยให้เป็น ms

นอกจากแอพพลิเคชั่นฝ่ายส่งและฝ่ายรับจะมีการส่ง-รับ RTP packet ถึงกันอย่างต่อเนื่องแล้ว แอพพลิเคชั่นทั้งสองยังมีการส่ง RTCP compound packet ถึงกันเป็นช่วงๆ อีกด้วย เพื่อให้การรับส่ง RTP packet ระหว่างกันเป็นไปอย่างถูกต้องและมีประสิทธิภาพ

**ผลการทำงานของแอพพลิเคชั่นทั้ง 3 ชุดเมื่อใช้ค่า default ของ codec แบบ G.711**




**ภาพที่ 8** แสดงการเปรียบเทียบค่า transit time ของ RTP packet แต่ละ packet ที่คำนวณโดยแอพพลิเคชั่นฝ่ายรับข้อมูลของการทดลองทั้ง 3 ชุด เมื่อใช้ค่า default ของ codec G.711

จากกราฟในรูปที่ 8 เมื่อเปรียบเทียบเฉพาะการทดลอง 2 ชุดแรกที่แอพพลิเคชั่นทำงานอยู่บนระบบปฏิบัติการ μClinux ภายใน NBC board พบว่าการทดลองชุดที่ 1 ซึ่งใช้ system call สามารถรับ RTP packet ได้มากกว่าการทดลองชุดที่ 2 ซึ่งใช้ API ภายใน Common library ของ

UCL โดยการทดลองชุดที่ 1 จะสามารถรับ RTP packet ได้ 525 packet ในขณะที่การทดลองชุดที่ 2 จะสามารถรับได้แค่ 444 packet และเมื่อตรวจสอบค่า transit time ที่คำนวณได้ของ RTP packet แต่ละ packet ในการทดลองทั้ง 2 ชุดพบว่า RTP packet ส่วนใหญ่จะมาถึงแอพพลิเคชั่นฝ่ายรับแต่ละ packet ห่างกันประมาณ 5 – 20 ms ซึ่งยังอยู่ในเกณฑ์ที่กำหนดว่าฝ่ายรับจะยอมรับ RTP packet ที่มาถึงในช่วง 0 – 200 ms สำหรับข้อมูลประเภทเสียง เนื่องจากถ้าระยะห่างระหว่างข้อมูลมีค่าเกินกว่าช่วงดังกล่าวจะทำให้มนุษย์รับทราบถึงการขาดหายไปของเสียง (Schulzrinne, H. and S. Casner. 2003.) แต่ก็จะมี RTP packet บาง packet ในการทดลองทั้ง 2 ชุดดังกล่าวที่มาถึงแอพพลิเคชั่นฝ่ายรับเกินกว่าช่วงเวลามาตรฐานที่กำหนดเอาไว้ ซึ่งจากกราฟในรูปที่ 8 พบว่าเหตุการณ์ดังกล่าวจะเกิดขึ้น 2 ครั้งด้วยกันดังนี้

1) เกิดจากแอพพลิเคชั่นฝ่ายส่งต้องทำการสร้างและส่ง RTCP compound packet ออกไปให้กับฝ่ายรับข้อมูล ซึ่งเหตุการณ์ดังกล่าวจะเกิดขึ้นเป็นช่วงๆ ตามมาตรฐานการรับ-ส่ง RTCP compound packet (ส่วนใหญ่ช่วงเวลาดังกล่าวจะมีค่าเป็น 5 วินาที) โดยเมื่อถึงช่วงเวลาดังกล่าว ฝ่ายส่งจะหยุดส่ง RTP packet ออกไปให้ฝ่ายรับชั่วคราว เพื่อไปจัดเตรียม RTCP compound packet แล้วส่ง packet ดังกล่าวออกไปให้ฝ่ายรับ และเมื่อฝ่ายส่งจัดการส่ง RTCP compound packet ออกไปให้กับฝ่ายรับเรียบร้อยแล้วฝ่ายส่งจะกลับมาเริ่มต้นส่ง RTP packet ไปให้ฝ่ายรับใหม่อีกครั้ง ดังนั้นเมื่อฝ่ายรับได้รับ RTP packet ดังกล่าวแล้วนำไปคำนวณค่า transit time กับ RTP packet ก่อนหน้านี้ (RTP packet สุดท้ายที่ฝ่ายส่งส่งมาให้ก่อนที่ฝ่ายส่งจะหยุดส่ง RTP packet แล้วไปเริ่มทำการจัดเตรียม RTCP compound packet แทน) จึงพบว่าค่า transit time ของ RTP packet ดังกล่าวที่คำนวณออกมาได้มีค่าสูงเกินกว่าเกณฑ์ (200 ms) ตามเวลาที่ฝ่ายส่งใช้ไปเพื่อจัดเตรียม RTCP compound packet สำหรับส่งให้กับฝ่ายรับ และเมื่อทำการตรวจสอบ sequence number ภายใน RTP packet ดังกล่าวที่ฝ่ายรับข้อมูล พบว่าจะต่อเนื่องกับ sequence number ที่อยู่ใน RTP packet ที่ได้รับมาก่อนหน้านี้ จึงแสดงให้เห็นไม่มีการสูญหายของ RTP packet เกิดขึ้นในช่วงเวลาดังกล่าว

2) เกิดจากแอพพลิเคชั่นฝ่ายรับต้องทำการสร้างและส่ง RTCP compound packet ออกไปให้กับฝ่ายส่งข้อมูล ซึ่งเหตุการณ์ดังกล่าวจะเกิดขึ้นเป็นช่วงๆ ตามมาตรฐานการรับ-ส่ง RTCP compound packet (ส่วนใหญ่ช่วงเวลาดังกล่าวจะมีค่าเป็น 5 วินาที) โดยในช่วงเวลาดังกล่าวฝ่ายรับจะหยุดรับ RTP packet ที่ฝ่ายส่งส่งมาให้ชั่วคราว เพื่อไปจัดเตรียม RTCP compound packet ให้เรียบร้อยแล้วส่ง packet ดังกล่าวออกไปให้ฝ่ายส่ง โดยเมื่อฝ่ายรับจัดการส่ง RTCP compound packet ออกไปให้กับฝ่ายส่งเรียบร้อยแล้ว ฝ่ายรับจึงจะกลับมาเริ่มต้นรับ RTP packet จากฝ่ายส่งอีก

ครั้ง ดังนั้นเมื่อฝ่ายรับได้รับ RTP packet จากฝ่ายส่งอีกครั้งฝ่ายรับจะนำ RTP packet ดังกล่าวไปคำนวณค่า transit time กับ RTP packet ที่ได้รับมาก่อนหน้านี้ (RTP packet สุดท้ายที่ฝ่ายรับได้รับเข้ามาก่อนที่ฝ่ายรับจะหยุดรับ RTP packet แล้วไปเริ่มทำการจัดเตรียม RTCP compound packet แทน) จึงพบว่าค่า transit time ของ RTP packet ดังกล่าวที่คำนวณออกมาได้มีค่าสูงเกินกว่าเกณฑ์ เนื่องจากในช่วงที่ฝ่ายรับหยุดรับ RTP packet จากฝ่ายส่งไป ฝ่ายส่งยังคงทำการส่ง RTP packet มาให้ฝ่ายรับอย่างต่อเนื่อง ทำให้มีการสูญหายของข้อมูลเกิดขึ้นที่ฝ่ายรับข้อมูล ส่งผลให้ค่า timestamp และค่า sequence number ที่อยู่ภายใน RTP packet ที่ได้รับเข้ามาใหม่กับที่อยู่ภายใน RTP packet ซึ่งได้รับมาก่อนหน้านี้ไม่ต่อเนื่องกัน จึงเป็นสาเหตุให้ค่า transit time ของ RTP packet ที่ได้รับเข้ามาใหม่เมื่อคำนวณออกมาแล้วมีค่าสูงเกินกว่าเกณฑ์ที่กำหนดไว้นั่นเอง

จากที่กล่าวไปข้างต้นว่าการทดลองชุดที่ 1 สามารถรับ RTP packet ได้มากกว่าการทดลองชุดที่ 2 โดยเมื่อสังเกตจากกราฟในรูปที่ 8 เพิ่มเติมพบว่า ที่การทดลองชุดที่ 1 สามารถรับ RTP packet ได้มากกว่าการทดลองชุดที่ 2 นั้นก็เนื่องมาจากการทดลองชุดที่ 1 สามารถรับ RTP packet ในแต่ละรอบก่อนที่ทั้งฝ่ายรับและฝ่ายส่งของการทดลองดังกล่าวจะทำการส่ง RTCP compound packet ถึงกันได้มากกว่าการทดลองชุดที่ 2 ที่เป็นเช่นนี้เนื่องจาก system call (ที่ใช้ในการทดลองที่ 1) ทำงานอยู่บน kernel-mode ของระบบปฏิบัติการ ทำให้สามารถติดต่อกับทรัพยากรต่างๆ ของระบบได้อย่างรวดเร็วกว่า API ที่ทำงานอยู่บน user-mode ของระบบปฏิบัติการ นอกจากนี้ scheduler ยังกำหนดให้งานของ system call บน kernel-mode มี priority สูงกว่างานของ API บน user-mode ทำให้การทำงานของ system call เป็นไปอย่างมีประสิทธิภาพมากกว่า

เมื่อนำผลการทดลองจากการทดลอง 2 ชุดแรกที่ทำงานอยู่บนระบบปฏิบัติการ μClinux ภายใน NBC board มาเปรียบเทียบกับการทดลองในชุดที่ 3 ซึ่งทำงานอยู่บนระบบปฏิบัติการ Linux ภายในเครื่อง PC พบว่าในการทดลองชุดที่ 3 จะไม่มี RTP packet ใดมีค่า transit time สูงกว่าเกณฑ์ที่กำหนดรวมทั้งไม่มีการสูญหายของ RTP packet ใดๆ เกิดขึ้นในระหว่างทำการทดลอง ซึ่งจะแตกต่างจากผลการทดลองที่ได้จากการทดลอง 2 ชุดแรกอย่างชัดเจน เนื่องจากความเร็วในการประมวลผลของหน่วยประมวลผลบนเครื่อง PC มีความเร็วสูงกว่าหน่วยประมวลผลที่อยู่บน NBC board (PC ที่ใช้ทดลองมีหน่วยประมวลผลความเร็ว 1.2 GHz ในขณะที่หน่วยประมวลผลบน NBC board มีความเร็ว 25 MHz) ทำให้การจัดเตรียม RTCP compound packet ในแต่ละครั้งเมื่อถึงช่วงเวลาที่ฝ่ายรับหรือฝ่ายส่งต้องทำการส่ง packet ดังกล่าวออกไปสามารถทำงานได้อย่างรวดเร็วและเสร็จทันก่อนการรับและส่ง RTP packet ในรอบถัดไป (ห่างกันรอบละ 20 ms ตาม codec

G.711 ที่เลือกใช้) จึงเสมือนว่าฝ่ายส่งไม่มีการหยุดส่ง RTP packet ออกไปและฝ่ายรับก็ไม่มีการหยุดรับ RTP packet เช่นกัน ทำให้การรับส่ง RTP packet ระหว่างฝ่ายรับและฝ่ายส่งในการทดลองชุดที่ 3 นี้เป็นไปอย่างต่อเนื่อง ในขณะที่การทดลอง 2 ชุดแรกทำอยู่บน NBC board ที่หน่วยประมวลผลมีความเร็วต่ำกว่า ทำให้แอพพลิเคชั่นในการทดลองทั้ง 2 ชุดดังกล่าวทำงานได้ช้ากว่าแอพพลิเคชั่นที่ใช้ในการทดลองชุดที่ 3 ส่งผลให้การจัดเตรียม RTCP compound packet ของฝ่ายรับและฝ่ายส่งในการทดลองทั้ง 2 ชุดนี้ต้องใช้เวลาในการทำงานนานกว่า ฝ่ายส่งจึงต้องหยุดส่ง RTP packet เป็นเวลานานเกินกว่า 20 ms (ซึ่งเป็นรอบเวลาที่ฝ่ายส่งต้องทำการส่ง RTP packet ให้ฝ่ายรับตาม codec G.711) และฝ่ายรับต้องหยุดรับ RTP packet เป็นเวลานานทำให้มีการสูญหายของ RTP packet สูญหายเกิดขึ้นที่ฝ่ายรับ เพื่อรอให้ขั้นตอนการจัดเตรียมและส่ง RTCP compound packet ของตนเองทำงานเสร็จสิ้นลงก่อน จึงจะสามารถกลับมารับและส่ง RTP packet ตามหน้าที่ของตนเองได้อีกครั้ง ซึ่งเป็นสาเหตุให้ค่า transit time ของ RTP packet บาง packet มีค่าสูงเกินกว่าเกณฑ์ที่กำหนดและมีการสูญหายของ RTP packet เกิดขึ้นที่ฝ่ายรับของการทดลองสองชุดแรกดังที่ได้กล่าวไปก่อนหน้านี้

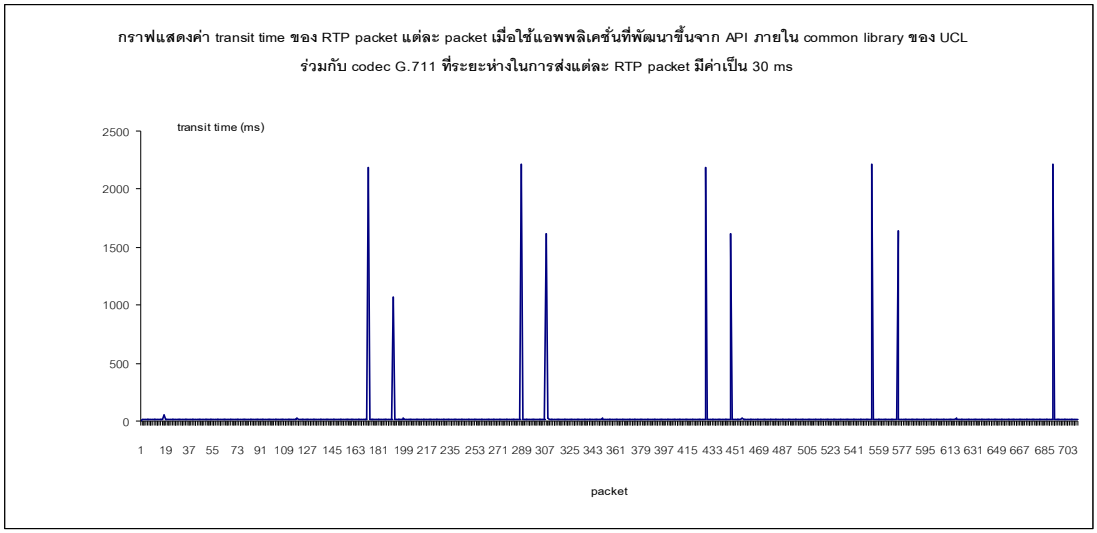
จากข้อมูลดังกล่าวจะเห็นได้ว่าความเร็วในการประมวลผลของหน่วยประมวลผลเป็นปัจจัยหลักที่ทำให้ RTP packet บาง packet มีค่า transit time สูงผิดปกติและมีการสูญหายของ RTP packet เกิดขึ้นที่ฝ่ายรับของการทดลองทั้งสองชุดบน NBC board เพื่อแก้ปัญหาเหล่านี้จึงทำการเปลี่ยนไปใช้ codec แบบอื่นแทน codec แบบ G.711 เพื่อเพิ่มระยะห่างระหว่างการส่ง RTP packet แต่ละ packet ให้มากกว่า 20 ms เนื่องจากระยะห่างของเวลาดังกล่าวยังไม่เหมาะสมสำหรับการรับส่ง RTP packet ให้ถูกต้องบน NBC board จึงได้ทำการทดลองกับแอพพลิเคชั่นในการทดลองชุดที่ 1 และชุดที่ 2 ซึ่งทำงานอยู่บน NBC board เพิ่มเติม โดยเปลี่ยนการใช้งานจาก codec G.711 ไปเป็น codec แบบอื่นๆ ที่อยู่ในตารางที่ 1 แทน เพื่อหาว่า codec แบบใดที่เหมาะสมสำหรับนำมาใช้ในการรับส่ง RTP packet บน NBC board อย่างถูกต้อง ซึ่งหมายถึงจะต้องไม่มี RTP packet ใดๆ ที่มีค่า transit time สูงเกินกว่า 200 ms และที่ฝ่ายรับจะต้องไม่มีการสูญหายของ RTP packet เกิดขึ้นตลอดการทดลอง

โดยการเปลี่ยนจาก codec แบบ G.711 ไปเป็น codec แบบอื่นๆ นั้น ก็จะต้องอาศัยไฟล์ประเภท audio ที่ได้รับการทำ encode เรียบร้อยแล้วด้วย codec แบบต่างๆ ตามในตารางที่ 1 มาใช้ในการทดลองแทนไฟล์ที่ใช้ในการทดลองก่อนหน้านี้ด้วย (ไฟล์ที่ใช้ในการทดลองก่อนหน้านี้ข้อมูลภายในได้รับการทำ encode ด้วย codec แบบ G.711) เพื่อจะได้นำเอาข้อมูลซึ่งอยู่ในไฟล์ต่างๆ

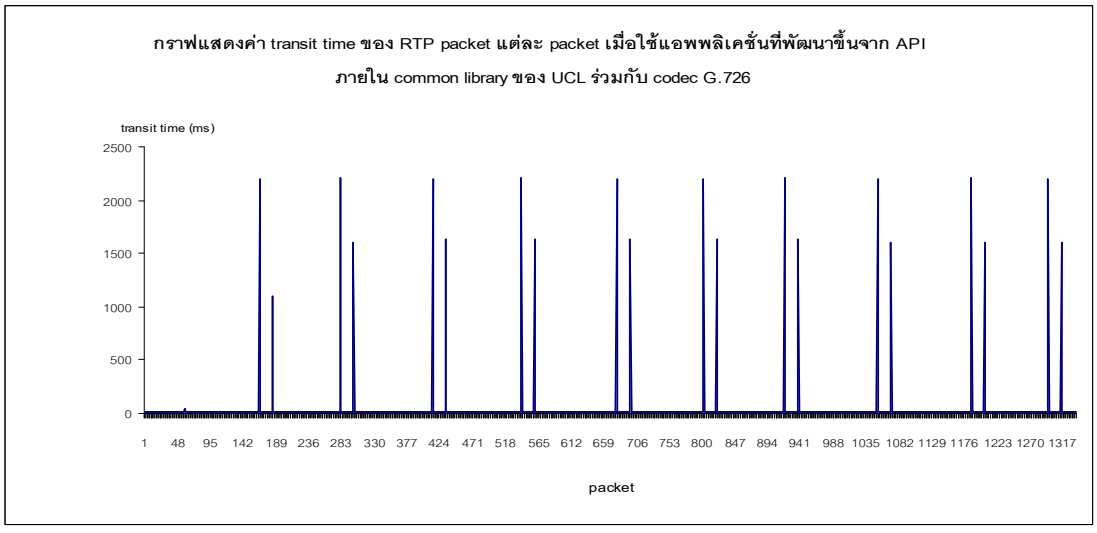
ดังกล่าวมาใช้สร้างเป็น RTP packet ที่เหมาะสมสำหรับ codec แต่ละแบบ นอกจากนี้จะต้องทำการแก้ไขแอพพลิเคชั่นทั้งฝ่ายรับและฝ่ายส่งข้อมูลเพิ่มเติม เพื่อให้แอพพลิเคชั่นดังกล่าวสามารถรองรับการทำงานกับ codec ใหม่ที่เลือกใช้ได้อย่างถูกต้อง โดยจะต้องกำหนดให้ฝ่ายส่งทำการอ่านข้อมูลในแต่ละครั้งจากไฟล์ที่ได้รับการ encode เรียบร้อยแล้วด้วย codec ที่ต้องการใช้งานให้มีขนาดเท่ากับขนาดของ payload ที่กำหนดเอาไว้สำหรับ codec นั้นๆ และเมื่อฝ่ายส่งนำ payload เหล่านั้นไปสร้างเป็น RTP packet เรียบร้อยแล้ว ฝ่ายส่งก็จะต้องกำหนดให้การส่ง RTP packet ออกไปในแต่ละครั้งมีช่วงเวลาห่างกันเท่ากับค่าของระยะห่างในการส่งแต่ละ payload ที่กำหนดเอาไว้สำหรับ codec ดังกล่าวและค่าระยะห่างนั้นจะต้องสัมพันธ์กับขนาดของ payload ที่กำหนดเอาไว้ก่อนหน้านี้ (ในขั้นตอนการอ่านข้อมูลจากไฟล์ของฝ่ายส่ง) อีกด้วย เพื่อให้อัตราเร็วในการขนส่งข้อมูลเป็นไปข้อกำหนดของ codec นั้นๆ นอกจากนี้ที่ฝ่ายรับข้อมูลก็จะต้องกำหนดให้ทำการรับ RTP packet แต่ละ packet ห่างกันตามค่าของระยะห่างในการส่งแต่ละ payload ที่กำหนดเอาไว้สำหรับ codec ที่เลือกใช้งาน เพื่อให้การรับส่ง RTP packet ระหว่างฝ่ายรับและฝ่ายส่งเป็นไปอย่างถูกต้อง และที่แอพพลิเคชั่นฝ่ายรับข้อมูลจะต้องแก้ไขตัวคูณค่า RTP timestamp ที่ใช้ในสูตรการค่าของ relative transit time จาก 1/8 สำหรับ codec แบบ G.711 (ดังที่ได้กล่าวถึงไปก่อนหน้านี้) ไปเป็นค่าอื่นๆ ตามอัตราเร็วของ codec ที่เลือกใช้งาน (1/4 สำหรับ 32 Kbps, 1/2 สำหรับ 16 Kbps, และ 1 สำหรับ 8 Kbps) เพื่อให้ค่า relative transit time ที่คำนวณได้จากการทดลองเมื่อใช้ codec แต่ละแบบเป็นไปอย่างถูกต้อง

โดยเริ่มต้นจะทำการทดลองโดยใช้ codec แบบต่างๆ ตามที่มีอยู่ในตารางที่ 1 ร่วมกับแอพพลิเคชั่นซึ่งพัฒนาขึ้นจาก API ภายใน Common Library ของ UCL ที่ทำงานอยู่บน NBC board ก่อน เพื่อตรวจสอบว่า codec และระยะห่างในการส่งแต่ละ RTP packet แบบใดจากในตารางที่ 1 ที่จะทำให้แอพพลิเคชั่นสำหรับรับ-ส่ง RTP และ RTCP packet บน NBC board ที่พัฒนาขึ้นจาก API ภายใน Common Library ของ UCL สามารถทำงานได้อย่างถูกต้อง ซึ่งก็คือจะต้องไม่มี RTP packet ใดที่มีค่า transit time สูงเกินกว่า 200 ms และจะต้องไม่มีการสูญหายของ RTP packet เกิดขึ้นเลยตลอดการทดลอง หลังจากนั้นจึงนำเอา codec และระยะห่างในการส่งแต่ละ RTP packet ที่ทดลองได้นี้ไปทดลองทำงานร่วมกับแอพพลิเคชั่นซึ่งพัฒนาขึ้นจาก system call ของโปรโตคอล RTP และ RTCP ภายในระบบปฏิบัติการ μClinux บน NBC board อีกครั้ง เพื่อใช้ในการเปรียบเทียบประสิทธิภาพการทำงานระหว่าง system call ของโปรโตคอล RTP และ RTCP ที่พัฒนาขึ้น กับ API ภายใน Common Library ของ UCL ต่อไป สำหรับผลการทดลองของแอพพลิเคชั่นซึ่งพัฒนาขึ้น

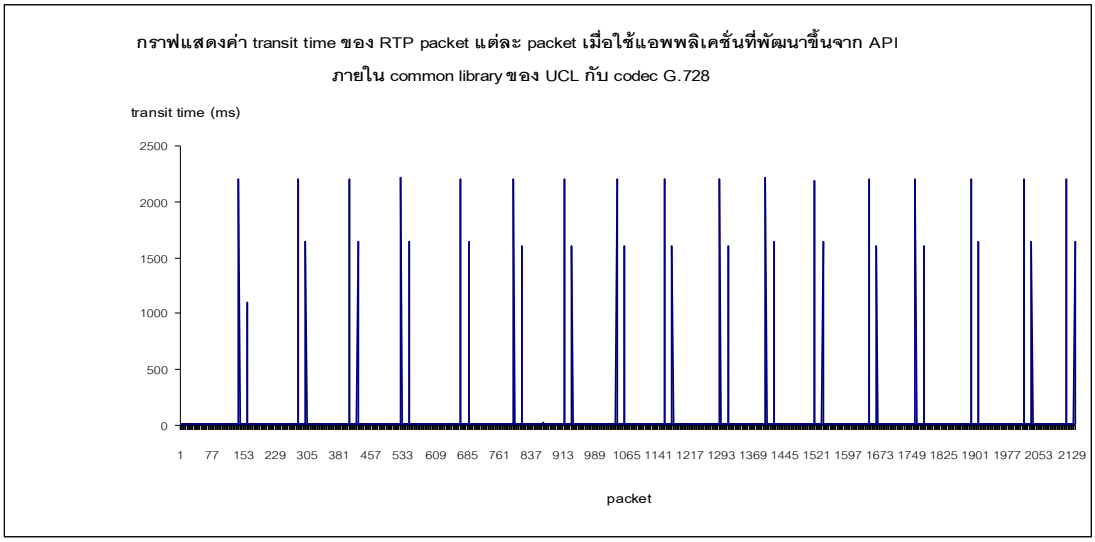
จาก API ภายใน Common Library ของ UCL ร่วมกับ codec แบบต่างๆ จากในตารางที่ 1 เป็นดังกราฟในรูปที่ 9 ถึง 13



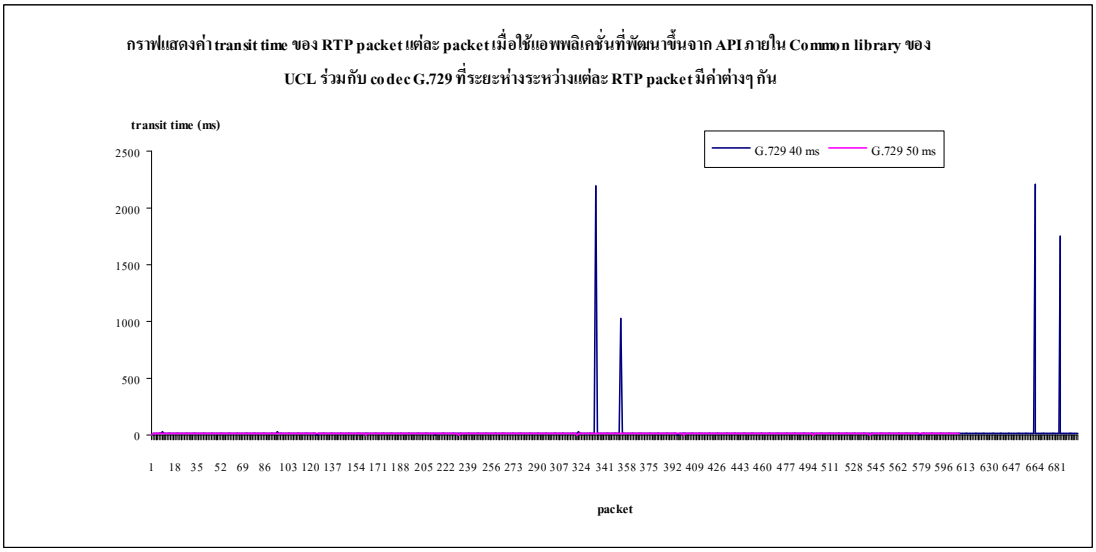

**ภาพที่ 9** แสดงค่า transit time ของ RTP packet แต่ละ packet เมื่อใช้ API ภายใน Common library ของ UCL ร่วมกับ codec G.711 ที่ระยะห่างในการส่งแต่ละ RTP packet เป็น 30 ms



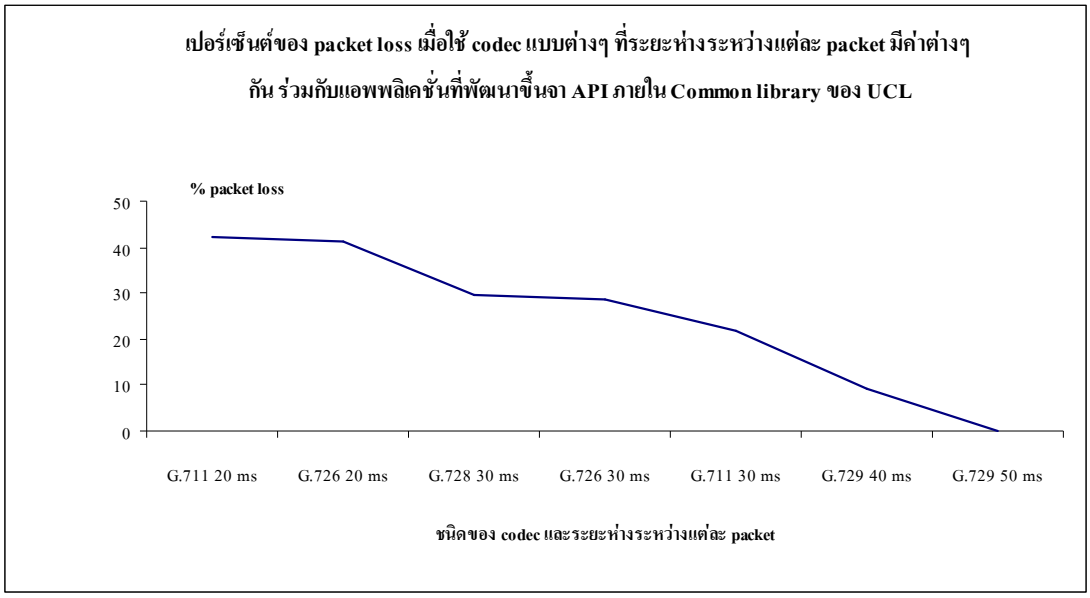

**ภาพที่ 10** แสดงค่า transit time ของ RTP packet แต่ละ packet เมื่อใช้ API ภายใน Common library ของ UCL ร่วมกับ codec G.726 (32 Kbps)




**ภาพที่ 11** แสดงค่า transit time ของ RTP packet แต่ละ packet เมื่อใช้ API ภายใน Common library ของ UCL ร่วมกับ codec G.728 (16 Kbps)




**ภาพที่ 12** แสดงค่า transit time ของ RTP packet แต่ละ packet เมื่อใช้แอพพลิเคชั่นที่พัฒนาขึ้นจาก API ภายใน Common library ของ UCL ร่วมกับ codec G.729 ที่ระยะห่างในการส่งแต่ละ RTP packet มีค่าต่างๆ กัน

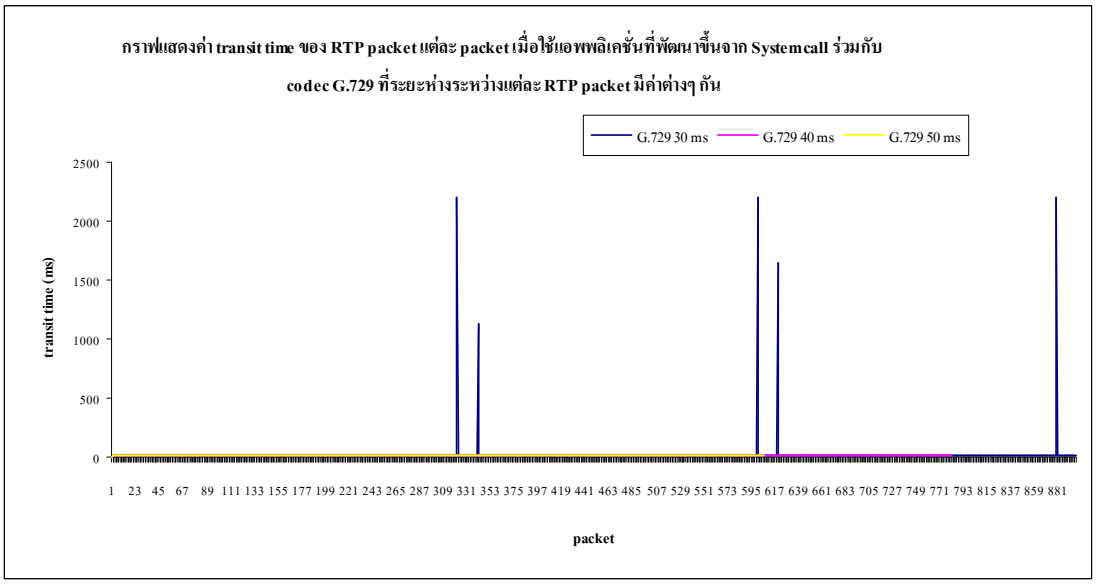



**ภาพที่ 13** แสดงเปอร์เซ็นต์การสูญหายของ RTP packet เมื่อใช้ codec แบบต่างๆ ที่ระยะห่างในการส่งแต่ละ RTP packet มีค่าต่างๆ กันร่วมกับแอพพลิเคชั่นที่พัฒนาขึ้นจาก API ภายใน Common library ของ UCL

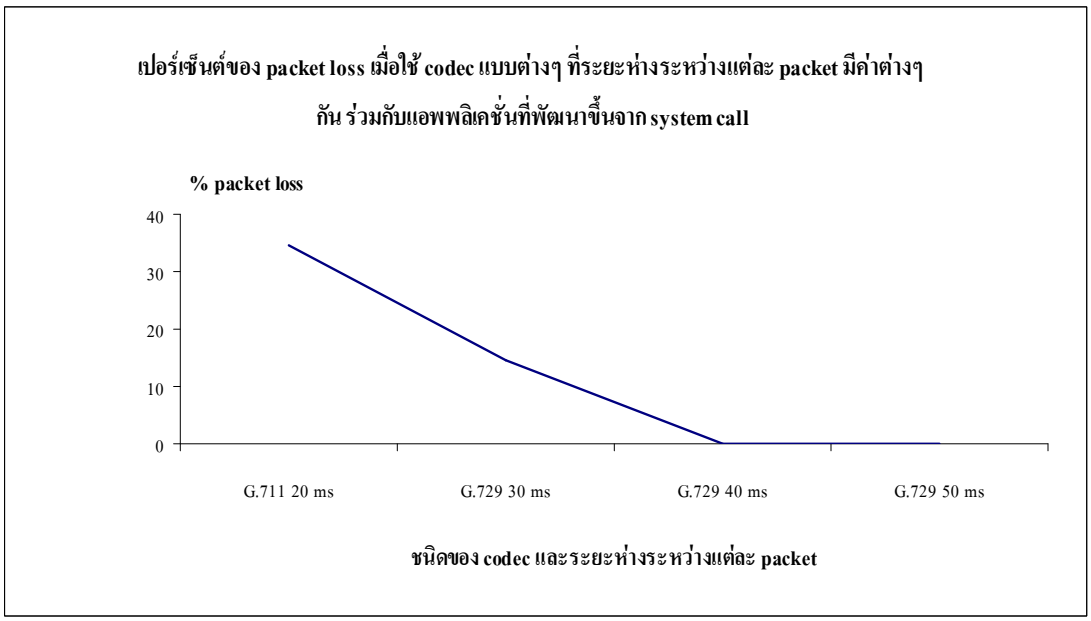
จากกราฟในรูปที่ 9 ถึง 12 ซึ่งแสดงค่า transit time ของ RTP packet แต่ละ packet ที่ฝ่ายรับข้อมูลคำนวณได้และกราฟในรูปที่ 13 ที่แสดงถึงเปอร์เซ็นต์การสูญหายของ RTP packet ที่ฝ่ายรับข้อมูล เมื่อใช้แอพพลิเคชั่นที่พัฒนาขึ้นจาก API ภายใน Common library ของ UCL ร่วมกับ codec แบบต่างๆ ที่ระยะห่างระหว่างการส่งแต่ละ RTP packet มีค่าต่างๆ กันสำหรับ codec แต่ละประเภทตามในตารางที่ 1 โดยจากกราฟทั้งหมดเมื่อใช้ codec G.711 (อัตราเร็ว 64 Kbps), G.726 (อัตราเร็ว 32 Kbps), G.728 (อัตราเร็ว 16 Kbps) ซึ่ง codec ดังกล่าวรองรับระยะห่างในการส่งแต่ละ RTP packet ที่ 20 และ 30 ms, และ G.729 (อัตราเร็ว 8 Kbps) ที่ระยะห่างในการส่งแต่ละ RTP packet เป็น 40 ms จะพบว่ายังคงมี RTP packet บาง packet มีค่า transit time สูงเกินกว่า 200 ms และยังคงมีการสูญหายของ RTP packet เกิดขึ้นที่ฝ่ายรับข้อมูลเช่นเดียวกับผลที่ได้ในการทดลองก่อนหน้านี้ จึงแสดงให้เห็นว่าระยะห่างระหว่างการส่งแต่ละ RTP packet ที่มีค่าเป็น 20, 30, และ 40 ms ไม่เหมาะสมต่อการทำงานของแอพพลิเคชั่นที่พัฒนาขึ้นจาก API ภายใน Common library ของ UCL บน NBC board แต่จากกราฟในรูปที่ 13 จะเห็นได้ว่าเปอร์เซ็นต์ในการสูญหายของ RTP packet จะลดลงเรื่อยๆ ตามระยะห่างระหว่างการส่งแต่ละ RTP packet ที่เพิ่มขึ้น และเมื่อใช้ codec G.729 ที่ระยะห่างในการส่งแต่ละ RTP packet มีค่าเป็น 50 ms ร่วมกับแอพพลิเคชั่นที่พัฒนาขึ้นจาก API ภายใน Common library ของ UCL จะพบว่าไม่มี RTP packet ใดมีค่า transit time สูงเกินกว่า 200

ms ดังแสดงในกราฟรูปที่ 12 และจากกราฟในรูปที่ 13 พบว่าเปอร์เซ็นต์การสูญหายของ RTP packet ของระยะห่างดังกล่าวจะมีค่าเป็น 0 ซึ่งแสดงให้เห็นว่าไม่มีการสูญหายของ RTP packet เกิดขึ้นตลอดการทดลอง ดังนั้นที่ระยะห่างในการส่งแต่ละ RTP packet เป็น 50 ms จึงทำให้การทำงานของแอพพลิเคชั่นที่พัฒนาขึ้นจาก API ภายใน Common library ของ UCL บน NBC board เป็นไปอย่างถูกต้อง

เมื่อทราบว่า codec แบบ G.729 ที่กำหนดระยะห่างในการส่งแต่ละ RTP packet มีค่าเป็น 50 ms สามารถทำให้แอพพลิเคชั่นซึ่งพัฒนาขึ้นจาก API ภายใน Common library ของ UCL ทำงานอยู่บน NBC board ได้อย่างถูกต้องโดยไม่มี RTP packet ใดมีค่า transit time สูงเกินกว่า 200 ms และตลอดการทดลองจะไม่มีการสูญหายของ RTP packet เกิดขึ้นที่ฝ่ายรับข้อมูล (เปอร์เซ็นต์ในการสูญหายของ RTP packet มีค่าเป็น 0) จึงได้อาศัย codec และระยะห่างในการส่งแต่ละ RTP packet ดังกล่าวไปทดลองร่วมกับแอพพลิเคชั่นที่พัฒนาขึ้นจาก system call บน NBC board เพื่อตรวจสอบว่าจะได้ผลการทำงานเช่นเดียวกับแอพพลิเคชั่นที่พัฒนาขึ้นจาก API ภายใน Common library ของ UCL ที่ทดลองไปก่อนหน้านี้หรือไม่ นอกจากนี้จะทดลองโดยใช้ codec G.729 ที่ระยะห่างในการส่งแต่ละ RTP packet มีค่าน้อยกว่า 50 ms ร่วมกับแอพพลิเคชั่นที่พัฒนาขึ้นจาก system call ด้วย เพื่อตรวจสอบว่าที่ระยะห่างในการส่งแต่ละ RTP packet น้อยกว่า 50 ms จะทำให้แอพพลิเคชั่นที่พัฒนาขึ้นจาก system call สามารถทำงานอยู่บน NBC board ได้อย่างถูกต้องหรือไม่ เพื่อจะได้ใช้ผลการทดลองดังกล่าวในการวัดประสิทธิภาพการทำงานระหว่าง system call และ API ภายใน Common library ของ UCL ต่อไป ซึ่งผลการทดลองระหว่างแอพพลิเคชั่นที่พัฒนาขึ้นจาก system call ร่วมกับ codec G.729 ที่ระยะห่างในการส่งแต่ละ RTP packet มีค่าต่างๆ กัน จะเป็นดังกราฟในรูปที่ 14 และ 15




**ภาพที่ 14** กราฟแสดงค่า transit time ของ RTP packet แต่ละ packet เมื่อใช้แอพพลิเคชั่นที่พัฒนาขึ้นจาก system call ร่วมกับ codec G.729 ที่ระยะห่างระหว่างการส่งแต่ละ RTP packet มีค่าต่างๆ กัน




**ภาพที่ 15** แสดงเปอร์เซ็นต์การสูญหายของ RTP packet เมื่อใช้ codec แบบต่างๆ ที่ระยะห่างในการส่งแต่ละ RTP packet มีค่าต่างๆ กันร่วมกับแอพพลิเคชั่นที่พัฒนาขึ้นจาก system call

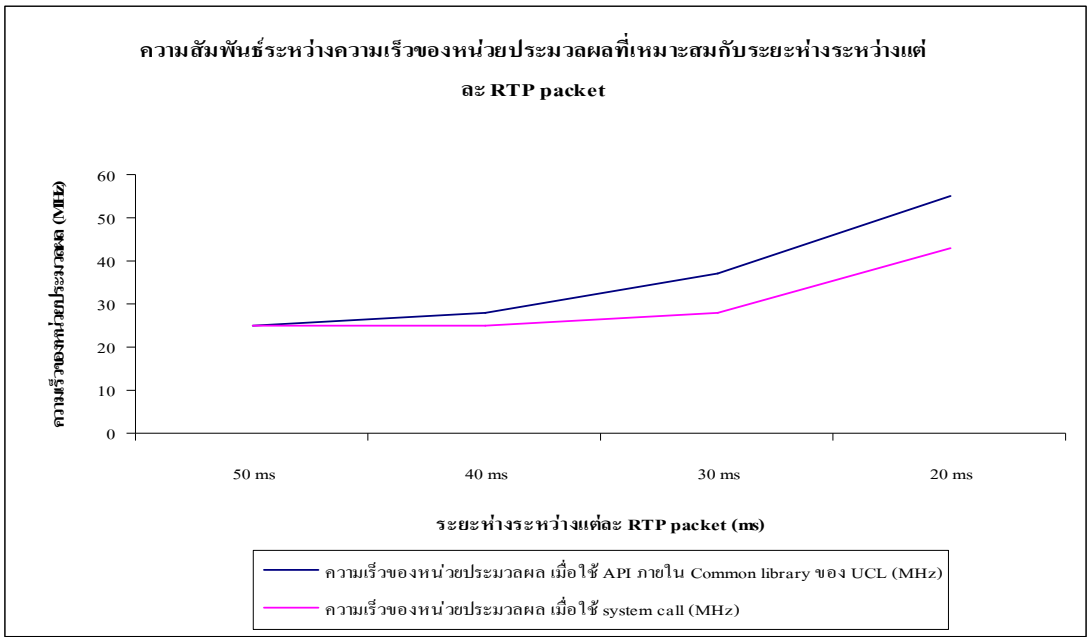
จากกราฟในรูปที่ 14 ซึ่งแสดงค่า transit time ของ RTP packet แต่ละ packet ที่ฝ่ายรับข้อมูลคำนวณได้และกราฟในรูปที่ 15 ที่แสดงถึงเปอร์เซ็นต์การสูญหายของ RTP packet ที่ฝ่ายรับข้อมูล เมื่อใช้แอพพลิเคชั่นที่พัฒนาขึ้นจาก system call ร่วมกับ codec G.729 ที่ระยะห่างในการส่งแต่ละ RTP packet มีค่าเป็น 50, 40, และ 30 ms โดยจากกราฟในรูปที่ 14 จะพบว่าเมื่อใช้ระยะห่างระหว่างการส่งแต่ละ RTP packet มีค่าเป็น 50 ms ก็จะทำให้การทำงานของแอพพลิเคชั่นที่พัฒนาขึ้นจาก system call บน NBC board เป็นไปอย่างถูกต้องโดยไม่มี RTP packet ใดมีค่า transit time สูงเกินกว่า 200 ms และตลอดการทดลองจะไม่มีการสูญหายของ RTP packet เกิดขึ้นที่ฝ่ายรับข้อมูลเช่นเดียวกับผลที่ได้จากการทำงานของแอพพลิเคชั่นที่พัฒนาขึ้นจาก API ภายใน Common library ของ UCL เมื่อใช้ร่วมกับระหว่างการส่งแต่ละ RTP packet ค่าดังกล่าว (50 ms) แต่เมื่อใช้ระยะห่างในการส่งแต่ละ RTP packet มีค่าเป็น 40 ms จะพบว่าแอพพลิเคชั่นที่พัฒนาขึ้นจาก system call ก็จะสามารถทำงานบน NBC board ได้อย่างถูกต้องเช่นเดียวกับเมื่อใช้ระยะห่างในการส่งแต่ละ RTP packet เป็น 50 ms โดยจากกราฟในรูปที่ 15 จะเห็นได้ว่าเปอร์เซ็นต์การสูญหายของ RTP packet เมื่อใช้ระยะห่างดังกล่าว (40 ms) มีค่าเป็น 0 จึงแสดงให้เห็นว่าไม่มีการสูญหายของ RTP packet ตลอดการทดลองเมื่อใช้ระยะห่างในการส่งแต่ละ RTP packet เป็น 40 ms ร่วมกับแอพพลิเคชั่นที่พัฒนาขึ้นจาก system call บน NBC board ซึ่งผลที่ได้จะแตกต่างจากผลการทำงานของแอพพลิเคชั่นที่พัฒนาขึ้นจาก API ภายใน Common library ของ UCL บน NBC board ร่วมกับ codec G.729 ที่ระยะห่างระหว่างแต่ละ RTP packet มีค่าเป็น 40 ms ในการทดลองก่อนหน้านี้ (มี RTP packet บาง packet มีค่า transit time สูงเกินกว่า 200 ms และมีการสูญหายของ RTP packet เกิดขึ้น)

จากผลการทดลองทั้งหมดที่ได้จึงแสดงให้เห็นว่าแอพพลิเคชั่นที่พัฒนาขึ้นมาจาก system call เมื่อนำมาใช้งานร่วมกับ codec G.729 ที่กำหนดให้ระยะห่างระหว่างการส่งแต่ละ payload มีค่าตั้งแต่ 40 ms ขึ้นไป สามารถทำงานบน NBC board ได้อย่างถูกต้องโดยไม่มี RTP packet ที่มีค่า transit time สูงเกินกว่า 200 ms และตลอดการทดลองจะไม่มีการสูญหายของ RTP packet เกิดขึ้นที่ฝ่ายรับข้อมูลอีกด้วย ในขณะที่แอพพลิเคชั่นที่พัฒนาขึ้นจาก API ภายใน Common library ของ UCL ต้องทำงานร่วมกับ codec G.729 ที่กำหนดให้ระยะห่างในการส่งแต่ละ payload มีค่าตั้งแต่ 50 ms ขึ้นไปเท่านั้นจึงสามารถทำงานบน NBC board ได้อย่างถูกต้อง จึงแสดงให้เห็นว่า system call ที่ทำงานอยู่บน kernel-mode ของระบบปฏิบัติการสามารถทำงานได้รวดเร็วกว่า API ภายใน Common library ของ UCL ที่ทำงานอยู่บน user-mode ของระบบปฏิบัติการ นอกจากนี้การทดลองดังกล่าวยังช่วยให้ทราบว่าถ้าจะนำ NBC board มาพัฒนาขึ้นเป็นโทรศัพท์ IP ระยะห่างระหว่างแต่

ละ RTP packet ที่เหมาะสมสำหรับการประมวลผลของ ARM7TDMI (ความเร็ว 25 MHz) ภายใน NBC board คือ 40 ms ขึ้นไปสำหรับการใช้ system call และ 50 ms ขึ้นไปสำหรับการใช้ API ภายใน Common Library ของ UCL ดังนั้น codec ใดที่สามารถทำงานรองรับกับช่วงเวลาดังกล่าวได้ก็สามารถนำมาใช้งานกับ NBC board ได้

จากกราฟในรูปที่ 9 ถึง 13 จะพบว่าระยะห่างระหว่างแต่ละ RTP packet ที่เหมาะสมสำหรับการทำงานบน NBC board สำหรับแอพพลิเคชั่นชุดที่พัฒนาขึ้นจาก API ภายใน common library ของ UCL คือ ตั้งแต่ 50 ms ขึ้นไป และจากกราฟในรูปที่ 14 จะพบว่าระยะห่างระหว่างแต่ละ RTP packet ที่เหมาะสมสำหรับการทำงานบน NBC board สำหรับแอพพลิเคชั่นที่พัฒนาขึ้นจาก system call คือ ตั้งแต่ 40 ms ขึ้นไป ซึ่งแสดงว่าช่วงเวลาดังกล่าวจะทำให้หน่วยประมวลผล ARM7TDMI ความเร็ว 25 MHz ภายใน NBC board สามารถรับส่งและประมวลผล RTP packet รวมถึง RTCP packet ได้ทันในแต่ละรอบของการทำงาน ทำให้ไม่มี RTP packet ใดมีค่า transit time สูงเกินกว่า 200 ms และไม่มีการสูญหายของ RTP packet เกิดขึ้นที่ฝ่ายรับข้อมูลตลอดการทดลองนั่นเอง โดยเมื่อตรวจสอบจากความเร็วในการประมวลผลของหน่วยประมวลผล (25 MHz) กับเวลาที่หน่วยประมวลดังกล่าวสามารถทำงานกับโปรโตคอล RTP และ RTCP ได้อย่างถูกต้อง (50 ms สำหรับ API ภายใน Common library ของ UCL และ 40 ms สำหรับ system call) แล้ว จะพบว่าการทำงานทั้งหมดของฟังก์ชันที่ใช้ในการสร้างและส่ง RTCP compound packet ในแต่ละรอบจะอยู่ในช่วงระหว่าง 1000000 cycle (25000 x 40) ถึง 1250000 (25000 x 50) cycle สำหรับแอพพลิเคชั่นที่พัฒนาขึ้นจาก API ภายใน Common library ของ UCL และจะอยู่ในช่วงระหว่าง 750000 cycle (25000 x 30) ถึง 1000000 cycle (25000 x 40) สำหรับแอพพลิเคชั่นที่พัฒนาขึ้นจาก system call ซึ่งถ้าใช้หน่วยประมวลผลที่มีความเร็วเพิ่มขึ้น (มากกว่า 25 MHz) หน่วยประมวลผลดังกล่าวก็น่าจะรองรับกับระยะห่างระหว่างแต่ละ RTP packet ในช่วงที่สั้นลงกว่านี้ได้ เนื่องจากหน่วยประมวลผลที่มีความเร็วสูงขึ้นก็จะสามารถทำงานในแต่ละวินาทีได้จำนวนของ cycle เพิ่มมากขึ้น ดังนั้นหน่วยประมวลผลในลักษณะดังกล่าวก็จะสามารถประมวลผลฟังก์ชันที่ใช้ในการสร้างและส่ง RTCP compound packet ในแต่ละรอบ (ระหว่าง 1000000 cycle ถึง 1250000 cycle สำหรับ API ภายใน Common library ของ UCL และ 750000 cycle ถึง 1000000 cycle สำหรับ system call) ได้ด้วยเวลาที่น้อยกว่า 50 ms สำหรับแอพพลิเคชั่นที่พัฒนาขึ้นจาก API ภายใน Common library ของ UCL และ 40 ms สำหรับแอพพลิเคชั่นที่พัฒนาขึ้นจาก system call ซึ่งจะส่งผลให้หน่วยประมวลผลที่มีความเร็วเพิ่มขึ้นนี้สามารถทำงานกับระยะห่างระหว่างแต่ละ RTP packet ที่มีค่าน้อยลงได้นั่นเอง

โดยความสัมพันธ์ระหว่างความเร็วของหน่วยประมวลผลที่เหมาะสมกับระยะห่างระหว่างแต่ละ RTP packet ที่คาดการณ์ไว้เป็นดังกราฟในรูปที่ 16




**ภาพที่ 16** แสดงความสัมพันธ์ระหว่างความเร็วของหน่วยประมวลผลที่เหมาะสมกับระยะห่างระหว่างแต่ละ RTP packet

จากกราฟในรูปที่ 16 จะเห็นได้ว่าเมื่อเพิ่มความเร็วของหน่วยประมวลผลให้มากขึ้น หน่วยประมวลผลดังกล่าวก็จะสามารถรองรับการทำงานกับ codec ที่มีระยะห่างในการส่งแต่ละ RTP packet ที่มีค่าน้อยลงได้ และเมื่อเปรียบเทียบระหว่างแอพพลิเคชั่นที่พัฒนาขึ้นจาก API ภายใน Common library ของ UCL กับแอพพลิเคชั่นที่พัฒนาขึ้นจาก system call จะพบว่าแอพพลิเคชั่นที่พัฒนาขึ้นจาก system call สามารถทำงานร่วมกับ codec ที่มีระยะห่างระหว่างแต่ละ RTP packet ซึ่งมีค่าน้อยๆ ได้โดยใช้หน่วยประมวลผลที่มีความเร็วน้อยกว่าที่แอพพลิเคชั่นซึ่งพัฒนาขึ้นจาก API ภายใน Common library ของ UCL ใช้เพื่อให้แอพพลิเคชั่นดังกล่าวทำงานร่วมกับระยะห่างเหล่านั้นได้ (สำหรับแอพพลิเคชั่นที่พัฒนาขึ้นจาก system call ถ้าทำงานร่วมกับหน่วยประมวลผลความเร็ว 43 MHz ก็จะสามารถทำงานรองรับกับระยะห่างระหว่างการส่งแต่ละ RTP packet ที่มีค่าเป็น 20 ms ได้ ในขณะที่แอพพลิเคชั่นที่พัฒนาขึ้นจาก API ภายใน Common library ของ UCL ต้องทำงานร่วมกับหน่วยประมวลผลที่มีความเร็ว 55 MHz ขึ้นไป จึงจะทำให้แอพพลิเคชั่นดังกล่าวสามารถทำงานกับระยะห่างในการส่งแต่ละ RTP packet ที่มีค่าเป็น 20 ms ได้)

# สรุปและข้อเสนอแนะ

## สรุป

เมื่อทดลองโดยใช้ codec G.711 (64 Kbps), codec G.726 (32 Kbps) และ G.728 (16 Kbps) ร่วมกับแอพพลิเคชั่นที่พัฒนาขึ้นจาก system call และ API ภายใน Common library ของ UCL บน NBC board พบว่าจะทำให้มี RTP packet บาง packet มีค่า transit time สูงเกินกว่า 200 ms (ถ้าสูงเกินกว่าค่านี้จะทำให้มนุษย์รับทราบถึงการขาดหายไปของเสียง) และมีการสูญหายของ RTP packet เกิดขึ้นที่ฝ่ายรับข้อมูลอีกด้วย เนื่องจากระยะห่างระหว่างแต่ละ RTP packet 20 และ 30 ms ที่ codec ดังกล่าวทำงานรองรับได้มีค่าน้อยเกินไป ทำให้หน่วยประมวลผล ARM7TDMI ความเร็ว 25 MHz ของ NBC board ไม่สามารถประมวลผลได้ทัน แสดงให้เห็นว่า codec ดังกล่าวไม่เหมาะสมกับการใช้งานบน NBC board แต่เมื่อใช้ codec G.729 (8 Kbps) ซึ่งรองรับระยะห่างระหว่างแต่ละ RTP packet เป็น 10, 20, 30, 40, 50, และ 60 ms (จากตารางที่ 1) มาทดลองใช้งานร่วมกับแอพพลิเคชั่นทั้ง 2 แบบบน NBC board พบว่าเมื่อกำหนดระยะห่างในการส่งแต่ละ RTP packet ให้มีค่าเป็น 40 ms (payload ที่อ่านจากไฟล์ต้องมีขนาดเป็น 40 byte) สำหรับแอพพลิเคชั่นชุดที่พัฒนาขึ้นจาก system call และ 50 ms (payload ที่อ่านจากไฟล์ต้องมีขนาดเป็น 50 byte) สำหรับแอพพลิเคชั่นชุดที่พัฒนาขึ้นจาก API ภายใน Common library ของ UCL จะทำให้ไม่มี RTP packet ใดมีค่า transit time ที่คำนวณออกมาได้สูงเกินกว่า 200 ms และตลอดการทดลองจะไม่มีการสูญหายของ RTP packet เกิดขึ้นด้วย จึงแสดงให้เห็นว่าระยะห่างระหว่างแต่ละ RTP packet ดังกล่าว (40 ms สำหรับ system call และ 50 ms สำหรับ API ภายใน Common library ของ UCL) จะทำให้หน่วยประมวลผลของ NBC board สามารถทำงานได้ทัน ซึ่งจะส่งผลให้โปรโตคอล RTP และ RTCP สามารถทำงานบน NBC board ได้อย่างถูกต้อง ดังนั้น codec แบบใดที่สามารถรองรับการทำงานกับระยะห่างระหว่างแต่ละ RTP packet ตั้งแต่ 40 ms ขึ้นไปสำหรับการใช้ system call และ ตั้งแต่ 50 ms ขึ้นไปสำหรับ API ภายใน Common library ของ UCL ก็จะสามารถใช้ codec ดังกล่าวบน NBC board ได้ นอกจากนี้จะเห็นได้ว่า system call สามารถรองรับการทำงานกับระยะห่างระหว่างแต่ละ RTP packet ที่มีค่าน้อยกว่า API ภายใน Common library ของ UCL (40 ms ขึ้นไปกับ 50 ms ขึ้นไปตามลำดับ) จึงแสดงให้เห็นว่า system call ที่ทำงานอยู่บน kernel-mode ของระบบปฏิบัติการสามารถทำงานได้อย่างรวดเร็วกว่า API ที่ทำงานอยู่บน user-mode ของระบบปฏิบัติการ ดังนั้นการนำ system call มาใช้งานจึงเป็นการช่วยเพิ่มประสิทธิภาพในการทำงานของโปรโตคอล RTP และ RTCP บนระบบปฏิบัติการ μClinux ภายใน NBC board ได้เป็นอย่างดี

และจากกราฟในรูปที่ 13 และ 15 ซึ่งแสดงถึงเปอร์เซ็นต์การสูญหายของ RTP packet เมื่อใช้แอพพลิเคชั่นที่พัฒนาขึ้นจาก API ภายใน Common library ของ UCL ร่วมกับ codec แบบต่างๆ ที่มีระยะห่างระหว่างแต่ละ RTP packet มีค่าต่างๆ กันและเมื่อใช้แอพพลิเคชั่นที่พัฒนาขึ้นจาก system call ร่วมกับ codec แบบต่างๆ ที่มีระยะห่างระหว่างแต่ละ RTP packet มีค่าต่างๆ กันตามลำดับ จะพบว่าที่ระยะห่างระหว่างการส่งแต่ละ RTP packet ที่เท่ากัน แอพพลิเคชั่นที่พัฒนาขึ้นจาก system call จะมีเปอร์เซ็นต์การสูญหายของ RTP packet น้อยกว่าแอพพลิเคชั่นที่พัฒนาขึ้นจาก API ภายใน Common library ของ UCL จึงแสดงให้เห็นว่าการนำ system call จึงเป็นการช่วยเพิ่มประสิทธิภาพในการทำงานของโปรโตคอล RTP และ RTCP บนระบบปฏิบัติการ μClinux ภายใน NBC board ได้มากยิ่งขึ้น

และจากกราฟในรูปที่ 16 จะแสดงให้เห็นว่าถ้าใช้ระบบคอมพิวเตอร์แบบฝังตัวที่หน่วยประมวลผลภายในมีความเร็วมากกว่า ARM7TDMI (25 MHz) ในการขนส่งข้อมูลแบบ real-time บนเครือข่าย IP โดยอาศัยโปรโตคอล RTP และ RTCP ระบบคอมพิวเตอร์แบบฝังตัวดังกล่าวก็น่าจะสามารถทำงานรองรับกับ codec ซึ่งกำหนดระยะห่างระหว่างการส่งแต่ละ RTP packet มีค่าน้อยกว่าที่ NBC board สามารถทำงานรองรับได้ นอกจากนี้ถ้าใช้แอพพลิเคชั่นที่พัฒนาขึ้นจาก system call บนระบบคอมพิวเตอร์แบบฝังตัวใด ก็จะช่วยให้ระบบคอมพิวเตอร์แบบฝังตัวนั้นสามารถทำงานร่วมกับ codec ซึ่งใช้ระยะห่างในการส่งแต่ละ RTP packet มีค่าน้อยกว่าที่แอพพลิเคชั่นซึ่งพัฒนาขึ้นจาก API ภายใน Common library ของ UCL สามารถทำงานรองรับได้บนระบบคอมพิวเตอร์แบบฝังตัวแบบเดียวกัน เนื่องจาก system call นั้นทำงานอยู่บน kernel-mode ของระบบปฏิบัติการ จึงสามารถทำงานได้อย่างรวดเร็วกว่า API ภายใน Common library ของ UCL ที่ทำงานอยู่บน user-mode ของระบบปฏิบัติการ ทำให้การประมวลผลฟังก์ชั่นต่างๆ ภายใน system call ใช้เวลาน้อยกว่า ส่งผลให้การทำงานของโปรโตคอล RTP และ RTCP เมื่อใช้ system call บนระบบคอมพิวเตอร์แบบฝังตัวเป็นไปอย่างรวดเร็วและมีประสิทธิภาพมากกว่าการใช้ API ภายใน Common library ของ UCL

นอกจาก system call จะช่วยเพิ่มประสิทธิภาพการทำงานของโปรโตคอล RTP และ RTCP ที่ทำงานอยู่บน NBC board แล้ว การใช้งานยังช่วยอำนวยความสะดวกให้กับผู้ที่ต้องการพัฒนาแอพพลิเคชั่นหรือโปรโตคอลระดับสูงที่เกี่ยวข้องกับการขนส่งข้อมูลประเภท real-time บนเครือข่าย IP สำหรับนำไปใช้งานอยู่บนระบบปฏิบัติการ μClinux ภายในระบบคอมพิวเตอร์แบบฝัง

ตัวใดๆ เนื่องจากผู้ใช้งานไม่จำเป็นต้องติดตั้ง library ของโปรโตคอล RTP และ RTCP ลงไปบน user-mode ของระบบปฏิบัติการ μClinux เพิ่มเติม เพราะ system call สามารถกำหนดให้มีอยู่ใน kernel ของระบบปฏิบัติการได้ทันทีตั้งแต่กระบวนการคอมไพล์และ build ระบบปฏิบัติการนั้นๆ และเมื่อพิจารณาขนาดของแอพพลิเคชั่นที่พัฒนาขึ้นมาจาก system call เปรียบเทียบกับแอพพลิเคชั่นที่พัฒนาขึ้นจาก API ภายใน library ที่ติดตั้งอยู่บน user-mode ของระบบปฏิบัติการ พบว่าแอพพลิเคชั่นที่พัฒนาขึ้นมาจาก system call จะมีขนาดเล็กกว่ามากเนื่องจากแอพพลิเคชั่นดังกล่าวจะ link เข้ากับ system call ต่างๆ ที่ตนเองเรียกใช้ก็ต่อเมื่อมีการเรียกใช้แอพพลิเคชั่นนั้นให้ทำงานเท่านั้น ซึ่งจะแตกต่างจากแอพพลิเคชั่นที่พัฒนาขึ้นจาก API ภายใน library ที่จะต้องทำการ link API ที่ตนเองต้องใช้งานจาก library ดังกล่าวรวมเข้าด้วยกันตั้งแต่ในขั้นตอนการ build แอพพลิเคชั่นนั้น ซึ่งการที่แอพพลิเคชั่นที่พัฒนาขึ้นจาก system call มีขนาดเล็กนี้จะทำให้ประหยัดเนื้อที่ในการจัดเก็บแอพพลิเคชั่นดังกล่าวในหน่วยความจำของระบบคอมพิวเตอร์แบบฝังตัว

## ข้อเสนอแนะและแนวทางการพัฒนาต่อ

โปรโตคอล RTP และ RTCP ถือว่าเป็นโปรโตคอลขั้นพื้นฐานสำหรับการทำงานของ VoIP โดยโปรโตคอลดังกล่าวมีหน้าที่ในการขนส่งข้อมูลประเภท real-time บนเครือข่าย IP ให้เป็นไปอย่างมีประสิทธิภาพการจะพัฒนาระบบคอมพิวเตอร์แบบฝังตัวให้กลายเป็นโทรศัพท์ IP ที่สมบูรณ์ได้นั้น นอกจากต้องมีโปรโตคอล RTP และ RTCP แล้ว ยังจำเป็นต้องมีโปรโตคอลที่ทำหน้าที่จัดการเกี่ยวกับการ call เช่น โปรโตคอล SIP และ H.323 ทำงานอยู่ภายในระบบคอมพิวเตอร์แบบฝังตัวนั้นๆ ด้วย ดังนั้นสิ่งที่ต้องพัฒนาต่อจากงานวิจัยชิ้นนี้ก็คือการพัฒนากลุ่มของโปรโตคอลที่ใช้ในการจัดการด้านการ call ขึ้นมาทำงานบนระบบปฏิบัติการ μClinux ซึ่งจะนำไปติดตั้งลงบนระบบคอมพิวเตอร์แบบฝังตัวที่ต้องการให้ทำงานเป็นโทรศัพท์ IP ได้ต่อไป โดยสามารถนำเอา system call ที่พัฒนาขึ้นในงานวิจัยชิ้นนี้มาใช้ในการพัฒนาโปรโตคอลดังกล่าวได้อีกด้วย

นอกจากนี้ system call ที่พัฒนาขึ้นยังทำงานรองรับเฉพาะ IPv4 เท่านั้น เนื่องจากในปัจจุบันระบบปฏิบัติการ μClinux ยังไม่สามารถรองรับการทำงานกับ IPv6 ได้อย่างสมบูรณ์ (อยู่ในขั้นตอนของการพัฒนา ปีที่ทำการวิจัย : 2549) ดังนั้นในอนาคตเมื่อมีการใช้งาน IPv6 อย่างแพร่หลายมากขึ้นและระบบปฏิบัติการ μClinux สามารถทำงานรองรับกับ IPv6 ได้อย่างสมบูรณ์

แล้ว ก็อาจจะต้องทำการแก้ไข system call ของโปรโตคอล RTP และ RTCP ให้สามารถทำงานรองรับกับ IPv6 ได้ด้วย เพื่อตอบสนองความต้องการของผู้ใช้งานให้มากที่สุด

จากการทดลองที่ผ่านมาพบว่าจะมีการทดลองการทำงานของ system call ในลักษณะของ unicast เท่านั้น เนื่องจาก system call ของโปรโตคอล RTP และ RTCP ที่พัฒนาขึ้นในงานวิจัยชิ้นนี้ยังไม่ได้พัฒนาให้รองรับการขนส่งข้อมูลในลักษณะของ multicast ดังนั้นเพื่อให้ระบบคอมพิวเตอร์แบบฝังตัวที่จะพัฒนาขึ้นเป็นโทรศัพท์ IP สามารถขนส่งข้อมูลข้อมูลประเภท real-time บนเครือข่าย IP ได้ทั้งแบบ unicast และ multicast จึงต้องพัฒนา system call ดังกล่าวให้สามารถทำงานรองรับการขนส่งข้อมูลในลักษณะของ multicast ได้ด้วย

# เอกสารและสิ่งอ้างอิง


Cisco Systems, Inc. 2006. **Voice Over IP - Per Call Bandwidth Consumption.** www.cisco.com/en/US/tech/tk652/tk698. Available Source: technologies_tech_note09186a0080094ae2.shtml, December 10, 2007.

Goode, B. 2002. Voice Over Internet Protocol (VoIP). **Proceedings of the IEEE** 9 (90): -.

Kurose, J. and K. Ross. 2002. **Computer Networking: A Top-Down Approach Featuring the Internet.** 2 ed. Addison Wesley, USA.

Lingxia Wang, C., B. Yao, Y. Yang and Z. Zhu. -. **A Survey of Embedded Operating System.** www.cs.ucsd.edu/classes/fa01/cse221/projects. Available Source: group2.pdf, December 22, 2005.

Mecklenburg, R. 2004. **Managing Projects with GNU Make.** 3 ed. O' Reilly Media, USA.

Mitra, D. 2001. **Technology Review#2001-2 Network Convergence and Voice over IP.** 40.

Nikkanen, K. 2003. **uClinux as an Embedded Solution.** Bachelor thesis, Turku Polytechnic.

Perkins, C. 2003. **RTP : Audio and Video for the Internet.** 1 ed. Addison-Wesley Professional, USA.

Schulzrinne, H. and S. Casner. 2003. **RTP Profile for Audio and Video Conferences with Minimal Control.** (Request for Comments : 3551). Network Working Group.

Schulzrinne, H., S. Casner, R. Frederick and V. Jacobson. 2003. **RTP : A Transport Protocol for Real-Time Applications.** (Request for Comments : 3550). Network Working Group.

Stevens, R., B. Fenner and A.M. Rudoff. 2004. **UNIX Network Programming The Sockets Networking API Volume 1.** 3 ed. Addison-Wesley Professional, USA.

Yaghmour, K. 2003. **Building Embedded Linux Systems.** 1 ed. O' Reilly Media, USA.

# ภาคผนวก

# ภาคผนวก ก

การติดตั้งระบบปฏิบัติการ μClinux ลงบนระบบคอมพิวเตอร์แบบฝังตัว

ในการติดตั้งระบบปฏิบัติการ μClinux ลงบนระบบคอมพิวเตอร์แบบฝังตัวใดๆ หลักสำคัญก็คือต้องตรวจสอบระบบคอมพิวเตอร์แบบฝังตัวที่มีอยู่ว่าประกอบด้วยอุปกรณ์อะไรบ้าง เพื่อใช้ข้อมูลเหล่านี้ในการกำหนดและปรับแต่งค่าต่างๆ ภายใน kernel ของระบบปฏิบัติการ μClinux ให้เหมาะสมกับระบบคอมพิวเตอร์แบบฝังตัวนั้นๆ อีกทั้งยังช่วยให้สามารถติดตั้ง driver ของอุปกรณ์ต่างๆ ที่มีอยู่บนระบบคอมพิวเตอร์แบบฝังตัวได้อย่างถูกต้อง ซึ่งจะทำให้ระบบคอมพิวเตอร์แบบฝังตัวสามารถทำงานได้เต็มประสิทธิภาพ เนื้อหาในช่วงถัดไปจะกล่าวถึงขั้นตอนหลักๆ ที่ใช้ในการติดตั้งระบบปฏิบัติการ μClinux ลงบน NBC board โดยผู้ที่สนใจสามารถนำขั้นตอนเหล่านี้ไปปรับใช้กับระบบคอมพิวเตอร์แบบฝังตัวที่ต้องการได้ต่อไป

## ขั้นตอนการติดตั้งระบบปฏิบัติการ μClinux ลงบน NBC board

### 1. ติดตั้ง cross-tool ลงบนระบบปฏิบัติการ Linux

cross-tool เป็นชุดเครื่องมือที่ติดตั้งอยู่บน platform หนึ่งแต่สามารถใช้ในการคอมไพล์หรือ build แอพพลิเคชั่นสำหรับ platform อื่นๆ ได้ ซึ่งระบบปฏิบัติการ μClinux ก็มีผู้พัฒนา cross-tool ขึ้นมาเพื่อใช้ติดตั้งลงบนระบบปฏิบัติการ Linux สำหรับใช้ในการคอมไพล์หรือ build แอพพลิเคชั่นรวมไปถึง image ของระบบปฏิบัติการ μClinux สำหรับ platform ที่ต้องการได้สะดวกยิ่งขึ้น ดังนั้นก่อนที่จะเริ่มคอมไพล์และ build image ของระบบปฏิบัติการ μClinux สำหรับทำงานกับ NBC board จึงจำเป็นต้องติดตั้ง cross-tool ที่เหมาะสมลงบนระบบปฏิบัติการ Linux ให้เรียบร้อย

ซึ่ง cross-tool ที่เหมาะสมในการนำมาทำงานกับ NBC board นั้น คือ cross-tool ที่ได้รับการพัฒนาขึ้นมาสำหรับ ARM platform เนื่องจาก NBC board ใช้ microcontroller Samsung S3C4530A ที่หน่วยประมวลผลภายในเป็น ARM7TDMI นั่นเอง โดยสามารถ download ไฟล์ binary ของ cross-tool สำหรับ ARM platform ที่มีชื่อว่า arm-elf-tools-20030314.sh ได้จากเว็บไซต์ http://www.μClinux.org/pub/μClinux/m68k-elf-tools/ สำหรับวิธีการติดตั้ง cross-tool ดังกล่าวลงบนระบบปฏิบัติการ Linux สามารถทำได้โดยการสั่งรัน arm-elf-tools-20030314.sh ที่ command prompt ของระบบปฏิบัติการ Linux ได้ทันที เมื่อรันคำสั่งนี้แล้วจะเห็นรายชื่อของไฟล์ต่างๆ ที่ติดตั้งสู่ระบบปฏิบัติการ Linux และเมื่อติดตั้งเรียบร้อยแล้วไฟล์ binary สำหรับการใช้งานทั้งหมดของ

cross-tool จะอยู่ภายใน directory /usr/local/bin (สำหรับ Red Hat Linux ถ้าเป็น distribution อื่นๆ ของ Linux อาจจะอยู่ที่ directory ที่แตกต่างออกไป)

**2. download source code ของระบบปฏิบัติการ µClinux**

สามารถ download source code ทั้งหมดของระบบปฏิบัติการ µClinux ที่เรียกว่า µClinux-dist เวอร์ชันล่าสุดได้ที่เว็บไซต์ http://www.µClinux.org/pub/µClinux/dist/ โดยภายใน µClinux-dist มีทั้งส่วนที่เป็น source code ของ µClinux kernel เวอร์ชันต่างๆ (ตั้งแต่ 2.0 – 2.6 ในปัจจุบัน), source code ของแอพพลิเคชั่นต่างๆที่มีให้ใช้งานอยู่บนระบบปฏิบัติการ µClinux, ไฟล์ config ต่างๆ สำหรับกำหนดค่าให้กับระบบคอมพิวเตอร์แบบฝังตัวแต่ละแบบ, รวมไปถึง uClibc ซึ่งเป็น library ภาษา C สำหรับทำงานกับระบบปฏิบัติการ µClinux เป็นต้น µClinux-dist ที่ download มาได้จะมีรูปแบบเป็นไฟล์ zip ซึ่งจะต้องทำการคลาย zip ใส่ไว้ใน directory ใดๆ ก็ได้ตามต้องการ ก่อนจะเริ่มคอมไพล์และ build image ของระบบปฏิบัติการ µClinux ต่อไป หลังจากคลาย zip ของ µClinux-dist เรียบร้อยแล้วจะพบว่าภายใน µClinux-dist ประกอบด้วย directory ต่างๆ ดังนี้

- /bin เป็น directory ที่รวมรวม utility ต่างๆ ในลักษณะของไฟล์ binary สำหรับใช้งานบนระบบปฏิบัติการ µClinux
- /Documentation เป็น directory ของเอกสารต่างๆ สำหรับใช้งานระบบปฏิบัติการ µClinux
- /config เป็น directory ที่เก็บไฟล์ configuration สำหรับแอพพลิเคชั่นต่างๆ ที่ทำงานบน usermode ของระบบปฏิบัติการ µClinux
- /freeswan เป็น directory ของแอพพลิเคชั่น IPSec ที่ใช้งานบนระบบปฏิบัติการ µClinux
- /glibc เป็น directory ที่เก็บ library ซึ่งนำมาจาก glibc ของระบบปฏิบัติการ Linux เพื่อใช้เป็น library เสริมในการคอมไพล์และ build แอพพลิเคชั่น
- /lib เป็น directory ที่เก็บ library เสริมอื่นๆ (นอกเหนือจาก uClibc) สำหรับใช้ในการคอมไพล์และ build แอพพลิเคชั่น
- /linux-2.0.x เป็น directory ที่เก็บ source code สำหรับ kernel เวอร์ชัน 2.0 ของระบบปฏิบัติการ µClinux
- /linux-2.4.x เป็น directory ที่เก็บ source code สำหรับ kernel เวอร์ชัน 2.4 ของระบบปฏิบัติการ µClinux

- /linux-2.6.x เป็น directory ที่เก็บ source code สำหรับ kernel เวอร์ชัน 2.6 ของระบบปฏิบัติการ μClinux

- /uClibc เป็น directory ของ library uClibc

- /tools เป็น directory ที่เก็บ utility ต่างๆ สำหรับการติดตั้ง ROM file system ลงบนระบบคอมพิวเตอร์แบบฝังตัว

- /user เป็น directory ที่เก็บ source code ของแอพพลิเคชั่นทั้งหมดที่มีใช้งานอยู่บนระบบปฏิบัติการ μClinux

- /vendors เป็น directory ที่เก็บไฟล์ configuration และ script เริ่มต้นการทำงาน สำหรับระบบคอมพิวเตอร์แบบฝังตัวประเภทต่างๆ

หลังจากทำการคอมไพล์และ build image ของระบบปฏิบัติการ μClinux สำเร็จแล้วจะมี directory ภายใน μClinux-dist เพิ่มขึ้นอีก 2 directory ดังนี้

- /romfs เป็น directory ที่เก็บโครงสร้างของ rom file system เอาไว้ ซึ่งเป็นโครงสร้างของ directory และไฟล์ต่างๆ ที่จะเกิดขึ้นจริงหลังจากที่ทำการบูทระบบปฏิบัติการ μClinux ขึ้นมาทำงานกับระบบคอมพิวเตอร์แบบฝังตัวเรียบร้อยแล้ว ดังนั้นสามารถใช้โครงสร้างที่อยู่ภายใน directory นี้ในการตรวจสอบความถูกต้องครบถ้วนของ directory และไฟล์ของแอพพลิเคชั่นต่างๆ ที่ต้องการใช้งานบนระบบปฏิบัติการ μClinux เพื่อไม่ให้มีข้อผิดพลาดเกิดขึ้นระหว่างที่ระบบปฏิบัติการ μClinux ทำงานอยู่บนระบบคอมพิวเตอร์แบบฝังตัว โดยปกติแล้วโครงสร้างของ rom file system ภายในระบบปฏิบัติการ μClinux จะคล้ายคลึงกับโครงสร้าง file system ภายในระบบปฏิบัติการ Linux เกือบทั้งหมด ส่งผลให้สภาพแวดล้อมการทำงานของระบบปฏิบัติการและแอพพลิเคชั่นต่างๆ ภายในระบบปฏิบัติการ μClinux ไม่แตกต่างจากระบบปฏิบัติการ Linux มากนัก ผู้ใช้งานที่เคยใช้ระบบปฏิบัติการ Linux มาก่อนจึงสามารถใช้งานระบบปฏิบัติการ μClinux ได้เป็นอย่างดี

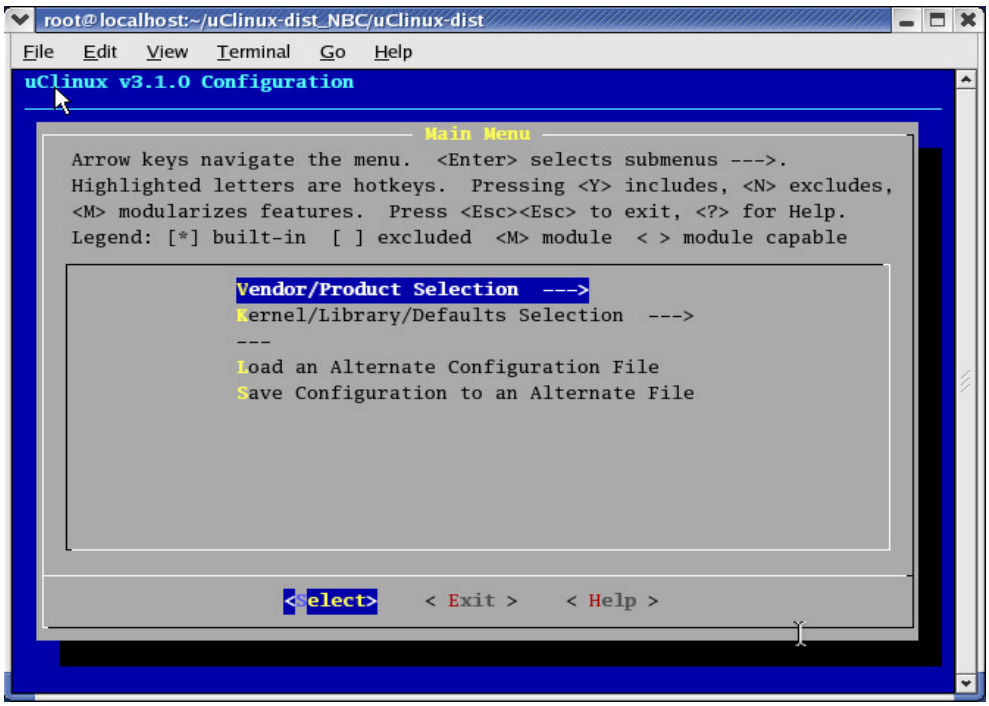
- /images เป็น directory สำหรับเก็บไฟล์ image ของระบบปฏิบัติการ μClinux ที่ build สำเร็จแล้วเอาไว้ ซึ่งโดยทั่วไปถ้ากระบวนการในการ build image สำเร็จหมดทุกขั้นตอนจะปรากฏไฟล์ภายใน directory นี้ 4 ไฟล์คือ image.bin, linux.data, linux.text, และ romfs.img โดยไฟล์ image.bin คือไฟล์ image ของระบบปฏิบัติการ μClinux ที่จะต้องนำไปติดตั้งลงบนระบบคอมพิวเตอร์แบบฝังตัวต่อไป

**3. คอมไพล์และ build image ของระบบปฏิบัติการ μClinux สำหรับ NBC board**

3.1 ทำการกำหนดค่าต่างๆ ของ kernel และแอพพลิเคชั่นต่างๆ ของระบบปฏิบัติการ μClinux

ก่อนที่จะเริ่มต้นคอมไพล์และ build image ของระบบปฏิบัติการ μClinux สิ่งสำคัญที่สุดที่ต้องทำเป็นสิ่งแรกคือการกำหนดค่าต่างๆ ของ kernel ของระบบปฏิบัติการ μClinux ให้ตรงกับระบบคอมพิวเตอร์แบบฝังตัวที่มีอยู่ เพื่อให้ระบบปฏิบัติการ μClinux สามารถทำงานกับระบบคอมพิวเตอร์แบบฝังตัวนั้นๆ ได้อย่างถูกต้อง นอกจากในส่วนของ kernel ของระบบปฏิบัติการ μClinux แล้วอีกส่วนหนึ่งที่จำเป็นต้องกำหนดค่าต่างๆ ให้ถูกต้องก็คือ แอพพลิเคชั่นสำหรับใช้งานบนระบบปฏิบัติการ μClinux ซึ่งในปัจจุบันบนระบบปฏิบัติการ μClinux มีแอพพลิเคชั่นให้เลือกใช้งานมากมาย สำหรับงานในหลายๆ ด้าน ดังนั้นผู้ใช้จึงต้องเลือกแอพพลิเคชั่นที่ต้องการและกำหนดค่าต่างๆ สำหรับแอพพลิเคชั่นนั้นๆ ให้ถูกต้อง เพื่อให้แอพพลิเคชั่นสามารถทำงานบนระบบปฏิบัติการ μClinux ได้อย่างสมบูรณ์

สำหรับวิธีการในการกำหนดค่าต่างๆ ทั้งในส่วนของ kernel และแอพพลิเคชั่นของระบบปฏิบัติการ μClinux ให้ตรงกับระบบคอมพิวเตอร์แบบฝังตัวที่มีอยู่นั่น สามารถทำได้โดยใช้คำสั่ง make menuconfig ซึ่งหลังจากป้อนคำสั่งนี้ลงไปบน command prompt ที่ชี้ไปยัง directory ของ μClinux-dist แล้วจะปรากฏเป็น GUI ขึ้นมาให้เราสามารถกำหนดค่าต่างๆ ได้สะดวกยิ่งขึ้น ดังรูปที่ 1

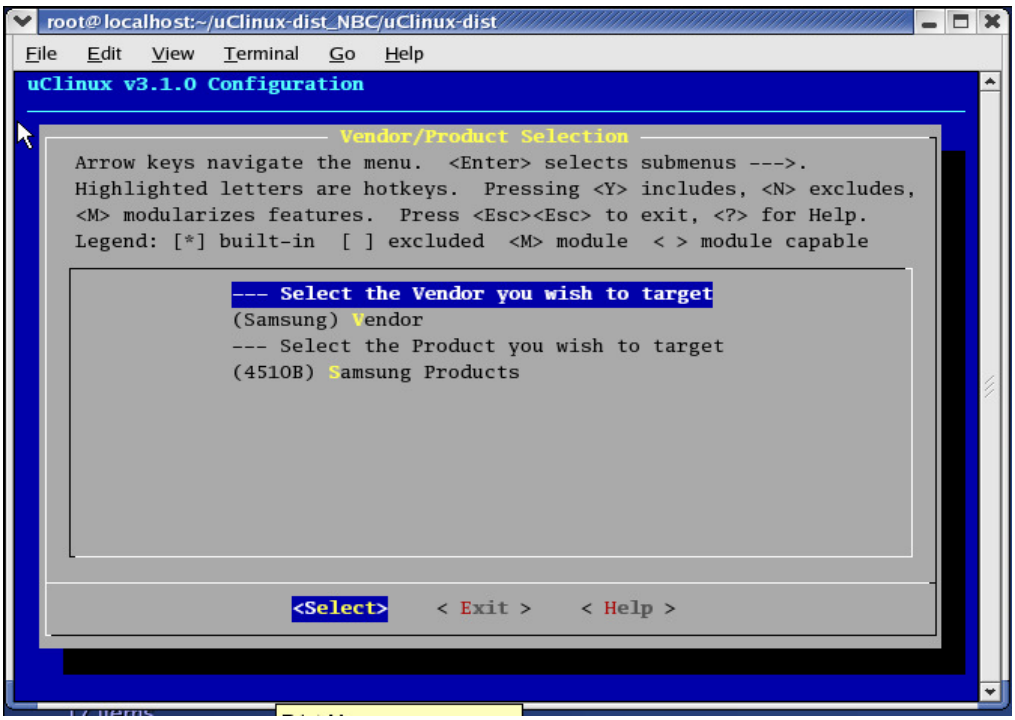



**ภาพผนวกที่ ก1** แสดงเมนูในลักษณะ GUI ที่ใช้กำหนดค่าต่างๆ ให้กับระบบปฏิบัติการ μClinux หลังจากใช้คำสั่ง make menuconfig

จากเมนูหลักในรูปที่ 2 นี้จะพบว่ามีเมนูย่อยให้เลือกอีก 2 เมนู คือ

- Vendor/Product Selection เป็นเมนูที่ใช้สำหรับเลือก platform ของระบบคอมพิวเตอร์แบบฝังตัวให้ตรงกันหรือให้ใกล้เคียงกับระบบคอมพิวเตอร์แบบฝังตัวที่เรามีอยู่ ซึ่งในปัจจุบันระบบปฏิบัติการ μClinux รองรับการทำงานกับระบบคอมพิวเตอร์แบบฝังตัวหลาย platform และหลายบริษัทผู้ผลิต โดยสามารถตรวจสอบค่าที่อยู่ในไฟล์ configuration และไฟล์ script ต่างๆ สำหรับแต่ละ platform ได้จาก directory /vendors/platform ภายใน μClinux-dist ได้ทันที สำหรับการเลือก platform ของระบบคอมพิวเตอร์แบบฝังตัวภายในเมนูนี้ให้ตรงหรือใกล้เคียงกับระบบคอมพิวเตอร์แบบฝังตัวที่เรามีอยู่ ก็เพื่อที่จะได้นำค่าที่กำหนดอยู่ภายในไฟล์ configuration และ script ต่างๆ ทั้งหมดของ platform ดังกล่าวมาใช้เป็นค่าเริ่มต้นให้กับ kernel, แอพพลิเคชั่นหลักๆ และ environment ต่างๆของระบบปฏิบัติการ μClinux สำหรับระบบคอมพิวเตอร์แบบฝังตัวของเรา ดังนั้นยิ่งเราเลือก platform ภายในเมนูนี้ให้ตรงหรือให้ใกล้เคียงกับระบบคอมพิวเตอร์แบบฝังตัวของเรามากเท่าไหร่ก็จะยิ่งช่วยให้เราสามารถกำหนดค่าต่างๆ ของระบบปฏิบัติการ μClinux ให้ตรงกับระบบคอมพิวเตอร์แบบฝังตัวของเราได้สะดวกมากยิ่งขึ้นเท่านั้น จากที่กล่าวมาข้างต้นในกรณีที่

ไม่สามารถหาค่าในเมนูนี้ที่จะใช้กำหนดให้ตรงกับ platform ของระบบคอมพิวเตอร์แบบฝังตัวที่มีอยู่ได้ ให้เลือกเอา platform ที่มีลักษณะใกล้เคียงกับที่ต้องการให้มากที่สุด ซึ่งหลังจากตรวจสอบในเมนูนี้แล้วพบว่าไม่มี platform Samsung S3C4530A ให้เลือกโดยตรง จึงเลือกใช้ platform Samsung S3C4510B ที่มีให้เลือกอยู่ภายในเมนูดังกล่าวแทน เนื่องจาก platform นี้มีลักษณะใกล้เคียงกับ Samsung S3C4530A มากที่สุด ดังแสดงในรูปที่ 2




**ภาพผนวกที่ ก2** แสดงการกำหนดค่าภายในเมนู Vendor/Product Selection สำหรับ NBC board

เราสามารถตรวจสอบการกำหนดค่าต่างๆ ของ platform Samsung S3C4510B เพื่อเป็นแนวทางในการกำหนดค่าให้กับ platform Samsung S3C4530A ได้จากไฟล์ต่างๆ ภายใน directory /vendors/Samsung/4510B ดังนี้

- config.linux-2.4.x เป็นไฟล์ configuration ที่กำหนดค่าพื้นฐานต่างๆ ใน kernel ของระบบปฏิบัติการ μClinux สำหรับ platform นั้นๆ

- config.vendor-2.4.x เป็นไฟล์ configuration ที่กำหนดแอพพลิเคชั่นเริ่มต้นซึ่งต้องการนำไปใช้งานบนระบบปฏิบัติการ μClinux สำหรับ platform นั้นๆ โดยแอพพลิเคชั่นที่กำหนดอยู่ในไฟล์นี้จะได้รับการ build รวมอยู่ภายใน image ของระบบปฏิบัติการ μClinux โดยอัตโนมัติ

- rc เป็นไฟล์ script สำหรับเริ่มต้นการทำงานของระบบคอมพิวเตอร์แบบฝังตัว platform นั้นๆ ซึ่งโดยปกติไฟล์นี้จะต้องนำไปติดตั้งอยู่ใน directory /etc ภายใน ROM file system (romfs) ของระบบปฏิบัติการ μClinux โดยระบบปฏิบัติการ μClinux จะเรียกคำสั่งต่างๆ ที่อยู่ในไฟล์ rc ขึ้นมาทำงานทันทีหลังจากที่ระบบปฏิบัติการ μClinux สามารถ boot ขึ้นมาทำงานบนระบบคอมพิวเตอร์แบบฝังตัวนั้นๆ เรียบร้อยแล้ว ซึ่งคำสั่งต่างๆ ดังกล่าวจะทำหน้าที่กำหนดค่าหรือทำงานบางอย่างเพื่อช่วยให้ระบบคอมพิวเตอร์แบบฝังตัว สามารถเริ่มต้นการทำงานได้อย่างถูกต้องครบถ้วนและพร้อมสำหรับการทำงาน เช่น กำหนดอินเทอร์เฟส loop back และเรียกอินเทอร์เฟสดังกล่าวให้เริ่มต้นทำงาน, เรียกเซอร์วิส dhcpd ขึ้นมาทำงานเพื่อกำหนด IP address ให้กับอินเทอร์เฟส Ethernet ของระบบคอมพิวเตอร์แบบฝังตัวนั้นๆ , mount file system ต่างๆ ที่ต้องการโดยอัตโนมัติ เป็นต้น ซึ่งถ้าเราต้องการให้มีการทำงานหรือกำหนดค่าใดๆ เพิ่มเติมเพื่อให้ระบบคอมพิวเตอร์แบบฝังตัว สามารถเริ่มต้นการทำงานได้อย่างถูกต้องครบถ้วน เราสามารถเขียนคำสั่งของงานดังกล่าวใส่ลงในไฟล์ rc ของระบบคอมพิวเตอร์แบบฝังตัว platform นั้นๆ ได้ทันที

- config.arch เป็นไฟล์ configuration ที่กำหนดลักษณะเฉพาะบางอย่างของ platform นั้นๆ เพื่อช่วยในการคอมไพล์และ build image ของระบบปฏิบัติการ μClinux ให้เหมาะสมกับ platform ดังกล่าว

- config.modules ถ้ามีการกำหนด module พิเศษที่นอกเหนือจากที่ระบบปฏิบัติการ μClinux จัดเตรียมไว้ให้ การกำหนดค่าของ module ดังกล่าวสำหรับ platform นั้นๆ จะเก็บเอาไว้ในไฟล์นี้

- config.uClibC เป็นไฟล์ configuration ที่กำหนดค่าพื้นฐานต่างๆ ของ library uClibc ให้ตรงกับ platform นั้นๆ

- inittab เป็นไฟล์ script ที่ใช้ในการกำหนดค่าเริ่มต้นของโปรเซสต่างๆ ให้สามารถเริ่มต้นการทำงานบนระบบปฏิบัติการ μClinux ได้อย่างถูกต้องสำหรับ platform นั้นๆ ซึ่งโดยปกติไฟล์นี้จะต้องนำไปติดตั้งอยู่ใน directory /etc ภายใน romfs ของระบบปฏิบัติการ μClinux

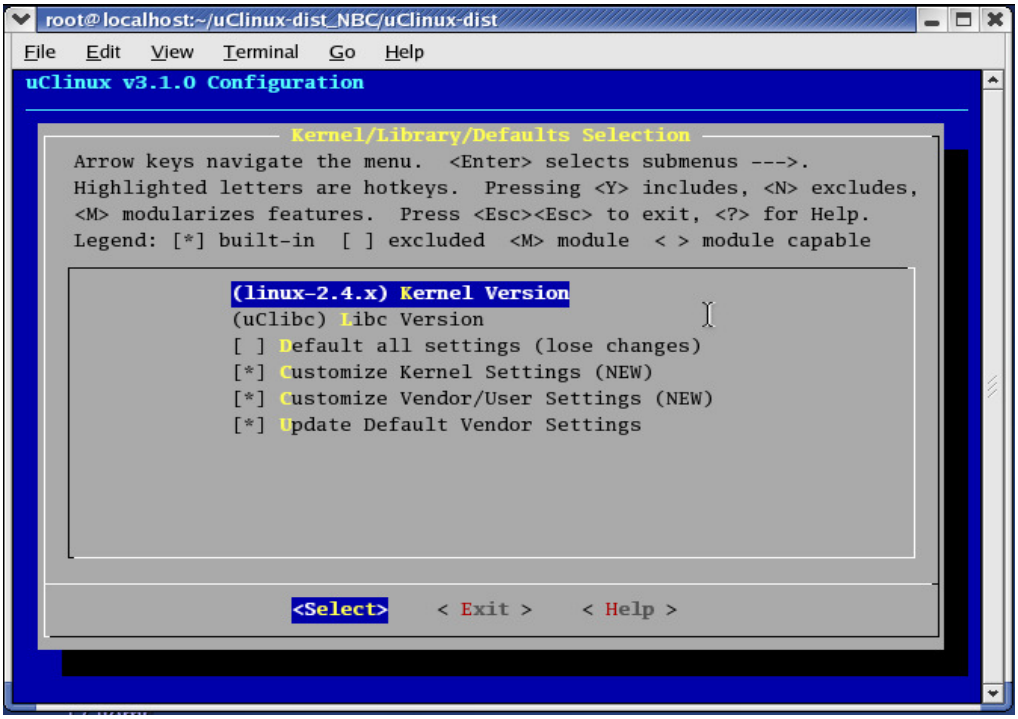
- Makefile เป็นไฟล์ที่มี instruction ต่างๆ สำหรับใช้ในการ build romfs และ image ของระบบปฏิบัติการ μClinux ให้ตรงกับ platform นั้นๆ นอกจากนี้ยังมี instruction ที่ใช้ในการกำหนด directory และติดตั้งไฟล์ script ต่างๆ (rc, inittab) ลงใน romfs ของระบบปฏิบัติการ μClinux อีกด้วย โดยจะมีการเรียกใช้งาน Makefile นี้ในขั้นตอนของการ build romfs และ build image ของระบบปฏิบัติการ μClinux ซึ่งจะกล่าวถึงต่อไป

การกำหนดค่าของ vendor และ product ภายในเมนู Vendor/Product Selection ให้ถูกต้องและใกล้เคียงกับ platform ที่เรามีจะช่วยให้เราสามารถใช้ค่าที่กำหนดอยู่ในไฟล์ configuration ต่างๆ รวมถึงไฟล์ script ทั้งหมดของ platform นั้นๆ มาใช้กับ platform ของเราได้ทันที โดยไม่ต้องแก้ไขมากนัก

- Kernel/Library/Defaults Selection เป็นเมนูที่ใช้ในการเลือกเวอร์ชันของ kernel ของระบบปฏิบัติการ μClinux ที่ต้องการ รวมทั้งใช้เลือก library สำหรับการคอมไพล์และ build image ของระบบปฏิบัติการ μClinux ในขั้นตอนต่อๆ ไป นอกจากนี้ยังมีตัวเลือกที่ใช้สำหรับกำหนดค่าบางอย่างดังนี้

- Default all settings (lose changes) เป็นการกำหนดว่าจะใช้ค่าต่างๆ ตามที่กำหนดเอาไว้ในไฟล์ configuration ของ Vendor/Product ที่เลือกเอาไว้ในเมนูก่อนหน้านี้หรือไม่
- Customize Kernel Settings (NEW) เป็นการกำหนดว่าจะทำการปรับแต่งและแก้ไขค่าต่างๆ ภายใน kernel ของระบบปฏิบัติการ μClinux ด้วยตนเองหรือไม่
- Customize Vender/User Settings (NEW) เป็นการกำหนดว่าจะทำการปรับแต่งรายละเอียดต่างๆ ของแอพพลิเคชั่นและ library เสริมอื่นๆ ที่ต้องการนำไปใช้งานบนระบบปฏิบัติการ μClinux ด้วยตนเองหรือไม่
- Update Default Vendor Settings (NEW) เป็นการกำหนดว่าต้องการบันทึกการเปลี่ยนแปลงทั้งหมดที่เกิดขึ้นระหว่างการกำหนดค่าให้กับ kernel และแอพพลิเคชั่นต่างๆ ของระบบปฏิบัติการ μClinux ลงในไฟล์ configuration ต่างๆ ของ vendor หรือ product ตามที่เลือกไว้ในเมนู Vendor/Product Selection หรือไม่ (Samsung S3C4510B)

ในที่นี้จะเลือกใช้ kernel ของระบบปฏิบัติการ μClinux เวอร์ชัน 2.4 และกำหนดให้ uClibc เป็น library หลักที่ใช้ในการคอมไพล์และ build ส่วนต่างๆ ของระบบปฏิบัติการ μClinux นอกจากนี้ยังจำเป็นต้องกำหนดให้สามารถปรับแต่งและแก้ไขค่าต่างๆ ภายใน kernel และแอพพลิเคชั่นทั้งหมดได้ด้วยตนเอง โดยจะไม่ใช้ค่าตามที่กำหนดอยู่ในไฟล์ configuration สำหรับ platform Samsung S3C4510B ในการนำไปใช้กับ platform Samsung S3C4530A ทั้งหมด เนื่องจากรายละเอียดภายใน platform ทั้ง 2 แบบมีความแตกต่างกันพอสมควร ทั้งนี้เพื่อให้ image ของระบบปฏิบัติการ μClinux ที่ build สำเร็จออกมาสามารถนำไปใช้งานกับ NBC board ได้อย่างถูกต้อง ซึ่งรูปแบบการกำหนดค่าต่างๆ ภายในเมนูนี้เป็นดังรูปที่ 3

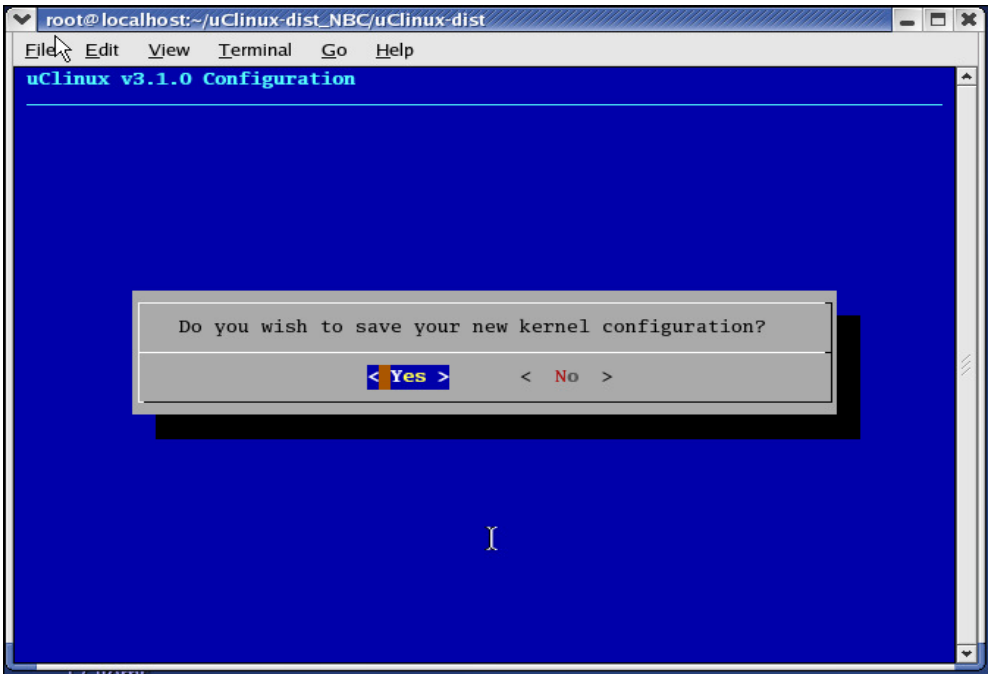



**ภาพผนวกที่ ก3** แสดงการกำหนดค่าภายในเมนู Kernel/Library/Defaults Selection

จากรูปที่ 3 พบว่าได้ทำการกำหนดค่าให้กับตัวเลือก Update Default Vendor Settings เพื่อให้ทำการบันทึกการเปลี่ยนแปลงต่างๆ ที่เกิดขึ้นในระหว่างการกำหนดค่าให้กับ kernel และแอพพลิเคชั่นทั้งหมดของระบบปฏิบัติการ µClinux สำหรับ NBC board ลงในไฟล์ configuration ต่างๆ ของ platform Samsung S3C4510B ตามที่เลือกเอาไว้ในเมนูก่อนหน้านี้ (Vendor/Product Selection) เพื่อใช้เป็นไฟล์ configuration ของ NBC board โดยเฉพาะ ซึ่งสามารถนำไปใช้งานได้ในครั้งต่อไป

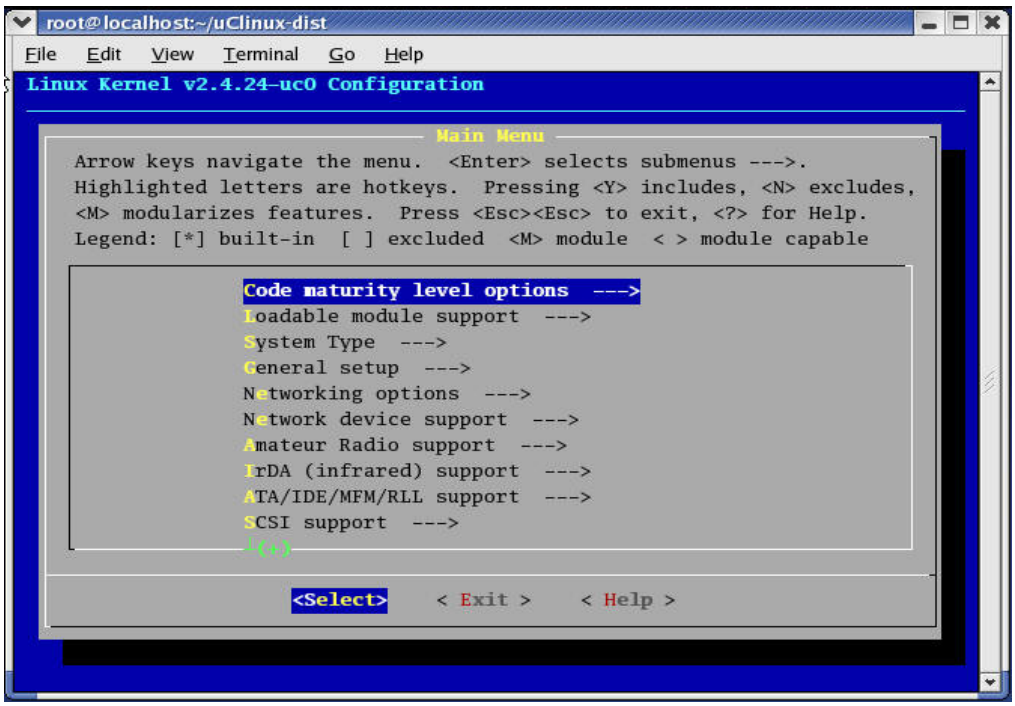
หลังจากที่ได้กำหนดค่าให้กับเมนูทั้งหมดที่กล่าวข้างต้นเป็นที่เรียบร้อยแล้ว ให้เลือกไปที่ Exit บนเมนูเพื่อออกจากเมนูนี้ เมื่อทำการเลือก Exit เรียบร้อยจะมีเมนูขึ้นมายืนยันว่าต้องการบันทึกการกำหนดค่าทั้งหมดในเมนู Vendor/Product Selection และเมนู Kernel/Library/Defaults Selection หรือไม่ ถ้าการกำหนดค่าต่างๆ ในเมนูทั้ง 2 นี้ถูกต้องก็ให้ทำการบันทึกเอาไว้โดยเลือก Yes เพื่อจะได้ใช้ค่าเหล่านี้ในการคอมไพล์และ build image ของระบบปฏิบัติการ µClinux ในขั้นตอนต่อไป แต่ถ้าการกำหนดค่าที่ทำไปยังไม่ถูกต้องก็สามารถเลือก No เพื่อออกจากเมนูโดยไม่มี

การบันทึกค่าใดๆ ทั้งสิ้น และผู้ใช้งานสามารถเริ่มต้นการกำหนดค่าที่ต้องการได้ใหม่อีกครั้ง ดังรูปที่ 4




**ภาพผนวกที่ ก4** แสดงเมนูที่ปรากฏขึ้นมายืนยันการบันทึกค่าที่ได้กำหนดให้กับเมนูก่อนหน้านี้

หลังจากบันทึกค่าต่างๆ ในเมนูที่ผ่านมาเรียบร้อยแล้ว ในกรณีที่เรากำหนดค่าให้กับตัวเลือก Customize Kernel Settings (NEW) ที่เมนู Kernel/Library/Defaults Selection เอาไว้ จะทำให้มีเมนูที่ใช้สำหรับกำหนดค่าต่างๆ ให้กับ kernel ของระบบปฏิบัติการ µClinux ปรากฏขึ้น ซึ่งเราสามารถใช้เมนูนี้กำหนดรายละเอียดของ kernel ตามที่เราต้องการได้ด้วยตนเอง ดังรูปที่ 5




**ภาพผนวกที่ ก5** แสดงเมนูที่ใช้ในการกำหนดค่าให้กับ kernel ของระบบปฏิบัติการ μClinux

จากเมนูที่แสดงในรูปที่ 5 พบว่าประกอบด้วยเมนูย่อยหลายเมนู สำหรับกำหนดค่าในด้านต่างๆ ให้กับ kernel ของระบบปฏิบัติการ μClinux เช่น เมนูย่อย Network device support ใช้กำหนด driver ของอุปกรณ์ network ให้ถูกต้องตรงกับที่มีอยู่บนระบบคอมพิวเตอร์แบบฝังตัว, เมนูย่อย Character devices ใช้กำหนด driver ให้กับ serial port สำหรับต่อเข้ากับ console เพื่อแสดงผลการทำงานของระบบปฏิบัติการ μClinux ทางหน้าจอคอมพิวเตอร์ เป็นต้น โดยขั้นตอนในการกำหนดค่าต่างๆ ให้กับ kernel ของระบบปฏิบัติการ μClinux เพื่อนำไปใช้งานกับ NBC board เป็นดังนี้

3.1.1 ขั้นตอนการกำหนดค่าให้กับ kernel ของระบบปฏิบัติการ μClinux สำหรับนำไปใช้งานกับ NBC board

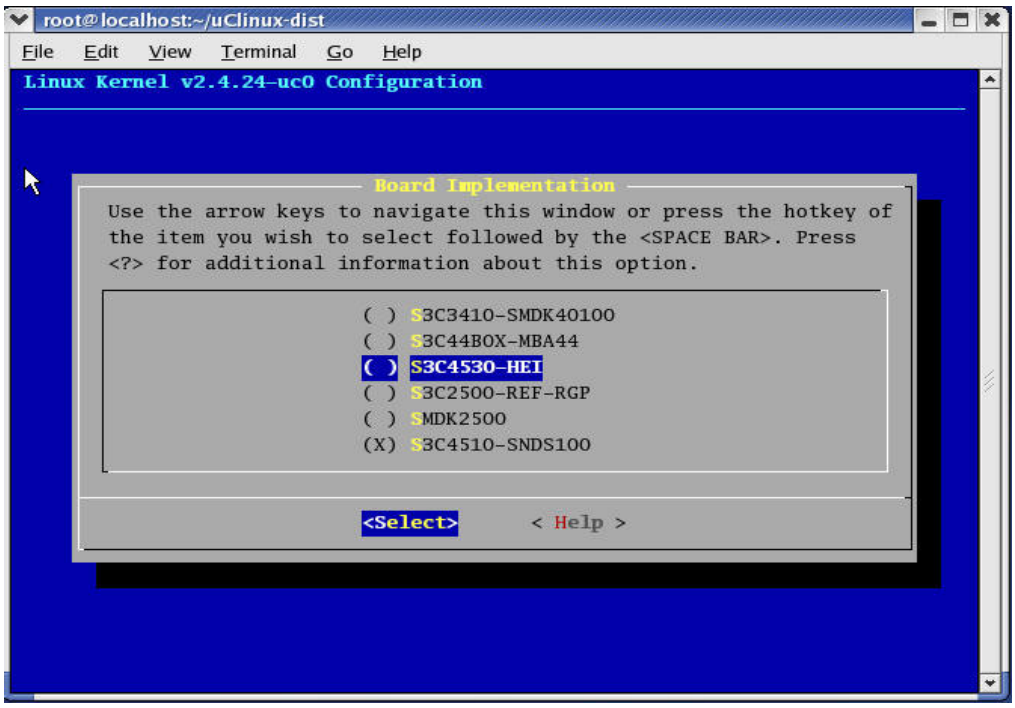
เริ่มต้นเมื่อเมนูสำหรับกำหนดค่าต่างๆ ให้กับ kernel ของระบบปฏิบัติการ μClinux ปรากฏขึ้นมา (ภาพผนวกที่ 5) พบว่าภายในเมนูย่อยๆ ของเมนูดังกล่าวจะมีการกำหนดค่าเริ่มต้นเอาไว้ให้อยู่แล้ว ซึ่งค่าเริ่มต้นเหล่านี้ก็คือค่าที่อยู่ในไฟล์ configuration config.linux-2.4.x ของ vendor หรือ product ที่เราได้ทำการเลือกเอาไว้ตั้งแต่ในเมนู Vendor/Product Selection ซึ่งในที่นี้ก็คือ Samsung

S3C4510B โดยเราสามารถใช้ค่าเริ่มต้นเหล่านี้บางค่าสำหรับ NBC board ได้ทันทีโดยไม่ต้องเปลี่ยนแปลง แต่ก็มีบางค่าที่ต้องทำการเปลี่ยนแปลงเพื่อให้ได้ kernel ของระบบปฏิบัติการ μClinux ที่เหมาะสมสำหรับ NBC board ซึ่งการกำหนดค่าต่างๆ ให้กับ kernel ของระบบปฏิบัติการ μClinux มีขั้นตอนหลักๆ ดังนี้

3.1.1.1 กำหนดชนิดของระบบคอมพิวเตอร์แบบฝังตัว

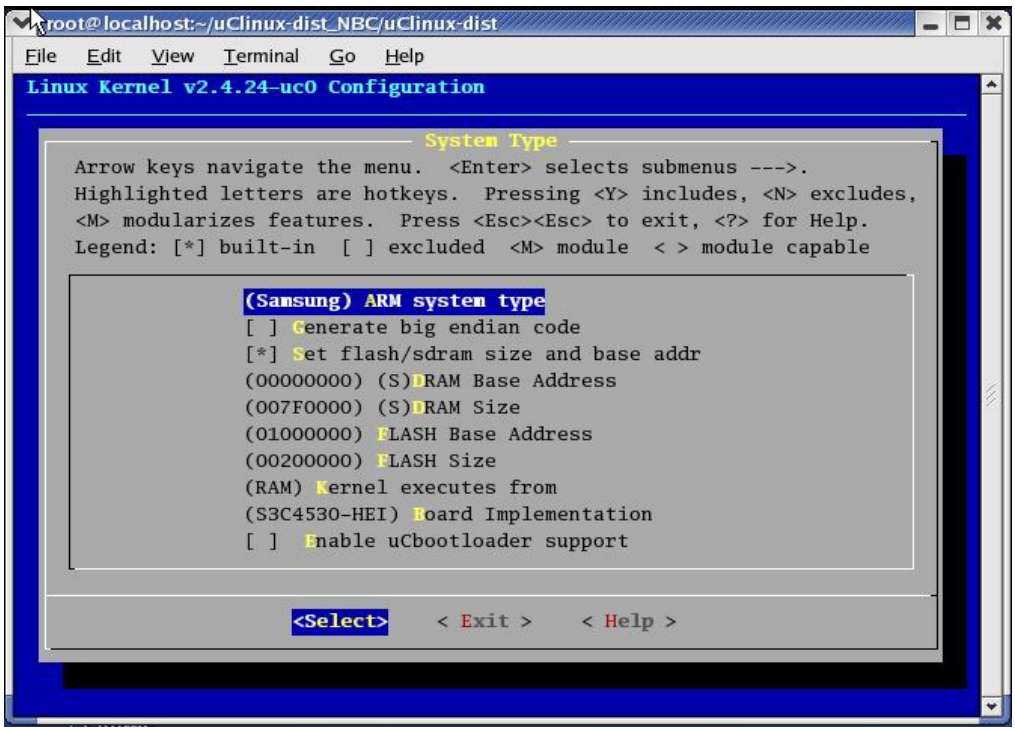
ในการกำหนดค่าต่างๆ ให้กับ kernel ของระบบปฏิบัติการ μClinux นั้น สิ่งแรกที่ต้องทำก็คือเลือกชนิดของระบบคอมพิวเตอร์แบบฝังตัวให้ตรงกับที่มีอยู่ ในกรณีที่ไม่มีชนิดของระบบคอมพิวเตอร์แบบฝังตัวตรงกับที่เราต้องการ ให้เลือกไปยังระบบคอมพิวเตอร์แบบฝังตัวที่ใช้ microcontroller แบบเดียวกับที่ระบบคอมพิวเตอร์แบบฝังตัวของเราใช้อยู่ ถ้าไม่มีให้เลือกทั้งระบบคอมพิวเตอร์แบบฝังตัวตามที่เราต้องการและระบบคอมพิวเตอร์แบบฝังตัวที่ใช้ microcontroller แบบเดียวกับระบบคอมพิวเตอร์แบบฝังตัวที่เรามีอยู่ ผู้ใช้งานจำเป็นสร้างไฟล์ configuration และเพิ่มเติมรายละเอียดต่างๆ สำหรับระบบคอมพิวเตอร์แบบฝังตัวนั้นๆ ลงไปยัง μClinux-dist ด้วยตนเอง แต่โดยส่วนใหญ่ระบบคอมพิวเตอร์แบบฝังตัวและ microcontroller เกือบทุกรูปแบบที่มีใช้งานกันอยู่ในปัจจุบัน จะมีให้เลือกใช้งานภายใน μClinux-dist อยู่แล้ว

ตัวเลือกภายในเมนู System Type ที่ใช้ในการกำหนดระบบคอมพิวเตอร์แบบฝังตัวที่ต้องการคือ Board Implementation โดยเมื่อเข้าไปในตัวเลือกดังกล่าวจะพบว่ามีระบบคอมพิวเตอร์แบบฝังตัวหลายแบบให้เลือก แต่ทั้งหมดจะเป็นระบบคอมพิวเตอร์แบบฝังตัวที่ใช้ microcontroller ของ Samsung ทั้งสิ้น และค่าเริ่มต้นที่เลือกเอาไว้จะเป็น S3C4510-SNDS100 ตามค่าที่เราได้กำหนดเอาไว้ในเมนูก่อนหน้านี้ (ตามค่าในเมนู Vendor/Product Selection) หลังจากตรวจดูรายชื่อของระบบคอมพิวเตอร์แบบฝังตัวทั้งหมดที่มีให้เลือก พบว่าไม่มี NBC board ให้เลือกได้โดยตรง จึงต้องเลือกระบบคอมพิวเตอร์แบบฝังตัวที่ใช้ microcontroller แบบเดียวกับที่มีอยู่บน NBC board (Samsung S3C4530A) ดังนั้นจึงเลือกเอาระบบคอมพิวเตอร์แบบฝังตัว S3C4530A-HEI ซึ่งใช้ microcontroller Samsung S3C4530A เหมือนกับ NBC board ดังรูปที่ 6




**ภาพผนวกที่ ก6** แสดงการเลือกระบบคอมพิวเตอร์แบบฝังตัว S3C4530A-HEI ที่ใช้ microcontroller เหมือนกับ NBC board

นอกจากตัวเลือก Board Implementation ภายในเมนู System Type ยังมีตัวเลือกย่อยอีก 2 ตัวเลือกที่เราต้องกำหนดค่าให้ถูกต้อง คือ ตัวเลือก Kernel executes from สำหรับใช้กำหนดว่า kernel ของระบบปฏิบัติการ µClinux จะทำงานที่ไหน (ROM/RAM) สำหรับ NBC board จะกำหนดค่าในตัวเลือกนี้ให้เป็น RAM อีกตัวเลือกที่ต้องกำหนดเพิ่มเติมคือตัวเลือก Set flash/sdram size and base addr ซึ่งเมื่อเราเลือกที่ตัวเลือกนี้จะปรากฏตัวเลือกย่อยขึ้นมาให้เรากำหนดขนาดและ address เริ่มต้นของ flash และ sdram สำหรับ microcontroller ที่เรามี แต่จะพบว่าค่าของตัวเลือกดังกล่าวได้รับการกำหนดเอาไว้แล้วตามระบบคอมพิวเตอร์แบบฝังตัวที่เราได้เลือกเอาไว้ในเมนู Board Implementation ก่อนหน้านี้ ดังนั้นเราสามารถใช้ค่าที่กำหนดเอาไว้โดยอัตโนมัติเหล่านี้ได้ทันที หลังจากที่ได้กำหนดค่าให้กับตัวเลือกที่จำเป็นในเมนู System Type เรียบร้อยแล้ว จะได้ดังรูปที่ 7




**ภาพผนวกที่ ก7** แสดงการกำหนดค่าต่างๆ ภายในเมนู System Type สำหรับ NBC board

หลังจากกำหนดค่าภายในเมนูย่อย System Type ตามที่ต้องการเรียบร้อยแล้วขั้นตอนต่อไปคือทำการกำหนด driver ที่จะใช้กับอุปกรณ์และ controller ต่างๆ ที่มีอยู่ภายใน microcontroller และบนระบบคอมพิวเตอร์แบบฝังตัวให้ถูกต้อง เพื่อให้อุปกรณ์และ controller เหล่านั้นทำงานได้อย่างมีประสิทธิภาพ

3.1.1.2 กำหนด driver ให้กับอุปกรณ์ต่างๆ ภายใน NBC board

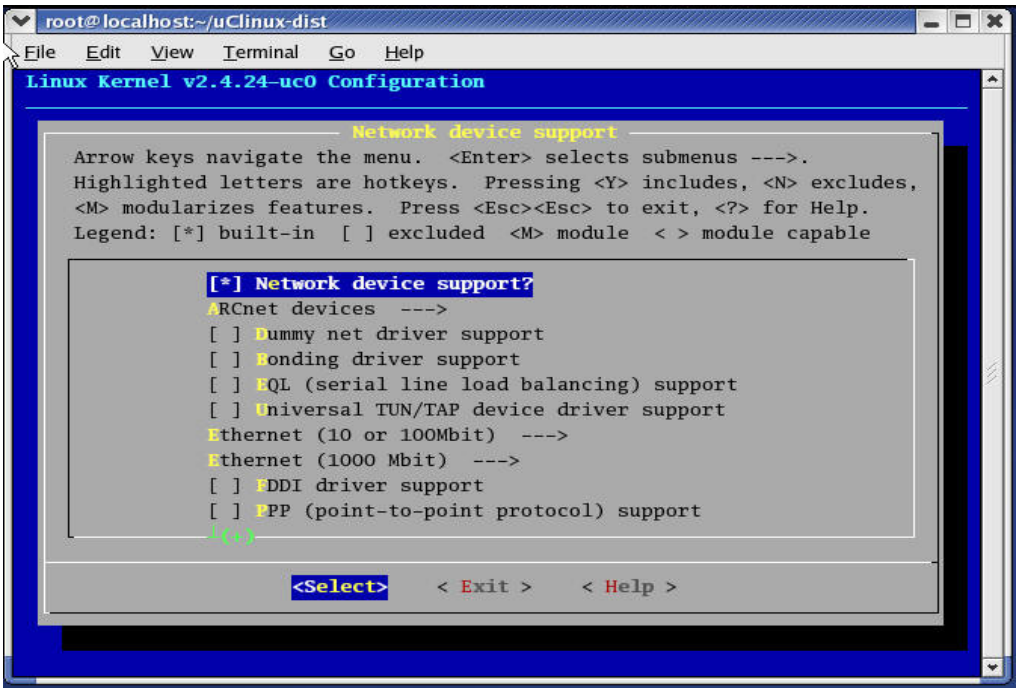
นอกจากส่วนของ System Type โดยรวมของระบบคอมพิวเตอร์แบบฝังตัวที่เราต้องกำหนดให้ถูกต้องตรงกับระบบคอมพิวเตอร์แบบฝังตัวที่เรามีแล้ว เราจำเป็นต้องกำหนด driver ให้กับ controller และอุปกรณ์ต่างๆ ที่อยู่ภายใน microcontroller และบนระบบคอมพิวเตอร์แบบฝังตัวให้ถูกต้องตรงกับที่มีอยู่บนระบบคอมพิวเตอร์แบบฝังตัวของเรา เพื่อให้ controller และอุปกรณ์เหล่านั้นสามารถทำงานได้อย่างมีประสิทธิภาพ

สำหรับ NBC board มี controller ที่สำคัญ 2 ตัวซึ่งอยู่ภายใน microcontroller Samsung S3C4530A คือ controller ของ Ethernet 10/100 mbps และ controller ของ UART สำหรับ serial

port โดยจะต้องติดตั้ง driver สำหรับ controller เหล่านี้ให้ถูกต้องเพื่อให้การทำงานของ controller ดังกล่าวเป็นไปอย่างมีประสิทธิภาพ วิธีการในการกำหนดค่า driver ให้กับ controller ทั้ง 2 เป็นดังนี้

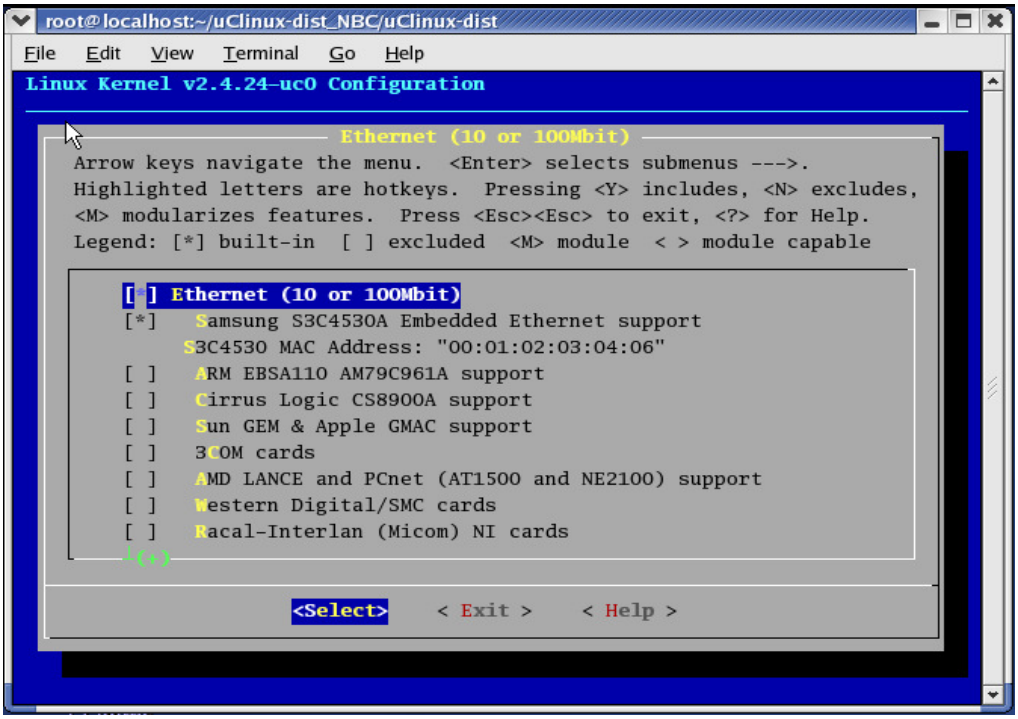
1) กำหนด driver ให้กับ controller ของ Ethernet 10/100 mbps

จากเมนูหลักสำหรับการกำหนดค่าในด้านต่างๆ ให้กับ kernel ของระบบปฏิบัติการ µClinux ให้เลือกไปที่ตัวเลือก Network device support ---> จะปรากฏเป็นเมนูย่อยสำหรับให้เลือกอุปกรณ์เน็ตเวิร์คแบบต่างๆ ดังรูปที่ 8




**ภาพผนวกที่ ก8** แสดงตัวเลือกต่างๆ ภายในเมนู Network device support

ในกรณีที่ไม่ปรากฏรายการของอุปกรณ์เน็ตเวิร์คให้เลือกดังรูปที่ 8 ให้ทำการเลือกที่ตัวเลือก Network device support ก่อน จึงจะปรากฏรายการของอุปกรณ์เน็ตเวิร์คทั้งหมดขึ้นมาให้เราทำการเลือกได้ เนื่องจาก controller ภายใน microcontroller Samsung S3C4530A เป็น controller สำหรับ Ethernet 10/100 mbps ดังนั้นเราจะทำการเลือกที่เมนูย่อย Ethernet (10 or 100Mbit) ---> หลังจากที่เลือกเมนูดังกล่าวแล้ว (Ethernet (10 or 100Mbit) --->) จะปรากฏเมนูย่อยสำหรับเลือก driver ให้ตรงกับ controller ของเรา ดังรูปที่ 9




**ภาพผนวกที่ ก9** แสดงเมนูที่ใช้ในการเลือก driver ให้เหมาะสมกับ controller ของ Ethernet 10/100 mbps ภายใน microcontroller Samsung S3C4530A

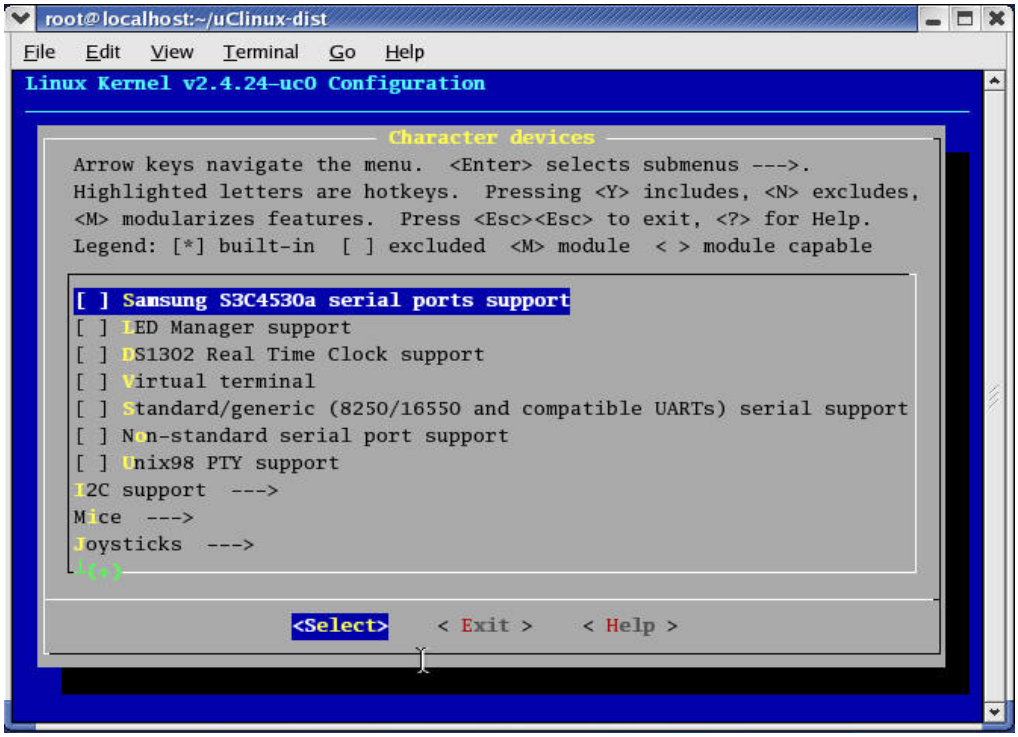
จากเมนูในรูปที่ 9 จะปรากฏรายชื่อของ driver สำหรับ controller และอุปกรณ์เน็ตเวิร์คแบบ Ethernet 10/100 mbps ทั้งหมด จากรายชื่อเหล่านี้จะทำการเลือกที่ตัวเลือก Samsung S3C4530A Embedded Ethernet support ดังแสดงในรูปที่ 9 เพื่อให้เหมาะสมกับ microcontroller ภายใน NBC board ของเรา สำหรับตัวเลือกดังกล่าวจะปรากฏขึ้นเองโดยอัตโนมัติเมื่อเรากำหนดค่าของ Board Implementation ภายในเมนู System Type ให้เป็น S3C4530-HEI ดังที่ได้กล่าวไปแล้วในหัวข้อ 3.1.1.1

นอกจากการเลือกที่ตัวเลือก Samsung S3C4530A Embedded Ethernet support สำหรับกำหนด driver ให้เหมาะสมกับ controller ของ Ethernet 10/100 mbps ที่อยู่ภายใน microcontroller Samsung S3C4530A แล้ว จะต้องทำการกำหนดค่า MAC address ให้กับ controller ดังกล่าว เพื่อใช้เสมือนเป็น ID เริ่มต้นของ controller นั้นๆ ในการทำงานบนเครือข่าย Ethernet ซึ่งการกำหนดค่า MAC address เอาไว้ จะช่วยให้เซอร์วิส dhcpd ที่จะเริ่มทำงานหลังจากระบบปฏิบัติการ µClinux บูทขึ้นมาทำงานบน NBC board เรียบร้อยแล้ว สามารถกำหนด IP address ที่เหมาะสมให้กับ

controller ดังกล่าวได้อย่างถูกต้อง ทำให้ระบบปฏิบัติการ μClinux และ แอพพลิเคชั่นทางด้านเน็ตเวิร์คทั้งหมดที่อยู่บน NBC board สามารถใช้อินเทอร์เฟสของ controller นี้ในการทำงานทางด้านเน็ตเวิร์คที่ตนเองรับผิดชอบได้ต่อไป

2) กำหนด driver ให้กับ controller ของ UART สำหรับ serial port

จากเมนูหลักสำหรับการกำหนดค่าในด้านต่างๆ ให้กับ kernel ของระบบปฏิบัติการ μClinux ให้เลือกไปที่ตัวเลือก Character devices ---> จะปรากฏเมนูย่อยสำหรับให้เลือก driver ของ Character device แบบต่างๆ ดังรูปที่ 10

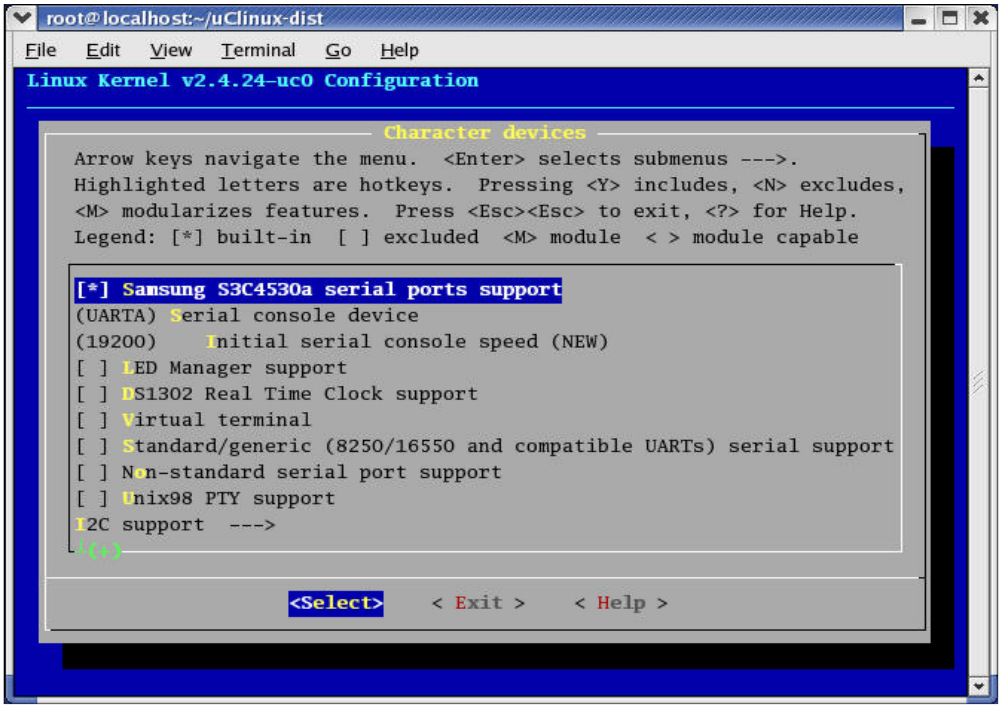



**ภาพผนวกที่ ก10** แสดงเมนูที่ใช้ในการเลือก driver ให้เหมาะสมกับ Character device ที่ต้องการ

จากเมนูที่แสดงในรูปที่ 10 จะทำการเลือกที่ตัวเลือก Samsung S3C4530A serial ports support เพื่อให้เหมาะสมกับ controller ของ UART สำหรับ serial port ที่จะใช้ต่อเข้ากับ console เพื่อแสดงผลการทำงานของ NBC board ออกทางหน้าจอคอมพิวเตอร์ โดยตัวเลือกดังกล่าวจะ

ปรากฏขึ้นมาโดยอัตโนมัติเมื่อเรากำหนดค่าของ Board Implementation ภายในเมนู System Type ให้เป็น S3C4530-HEI ดังที่ได้กล่าวไปแล้วในหัวข้อ 3.1.1.1

เนื่องจากบน NBC board ได้จัดเตรียม UART เอาไว้ให้ 2 channel ซึ่ง channel แรกใช้ต่อเข้ากับ console ในขณะที่อีก channel ไว้ต่อเข้ากับ modem โดยในที่นี้จะใช้เฉพาะ channel สำหรับต่อเข้ากับ console เพื่อใช้รับ input และแสดง output โต้ตอบกับผู้ใช้งานผ่านหน้าจอของคอมพิวเตอร์เท่านั้น ซึ่งต้องกำหนดในตัวเลือก Serial console device ให้ถูกต้องว่าจะใช้ channel ใดในการทำหน้าที่ดังกล่าว จากข้อมูลที่ได้กล่าวมาก่อนหน้านี้จึงเลือก UARTA ให้กับตัวเลือกนี้ นอกจาก channel ที่ต้องกำหนดให้ถูกต้องแล้ว ความเร็วในการขนส่งข้อมูลก็เป็นอีกส่วนที่จะต้องทำการกำหนดค่าเอาไว้ โดยในที่นี้จะกำหนดความเร็วในการขนส่งข้อมูลเอาไว้ที่ 19200 bit/วินาที ซึ่งค่าดังกล่าวจะต้องนำไปกำหนดให้กับแอพพลิเคชั่นบนเครื่องคอมพิวเตอร์ที่ใช้ในการติดต่อกับ NBC board เพื่อให้การรับส่งข้อมูลระหว่าง NBC board และเครื่องคอมพิวเตอร์เป็นไปอย่างถูกต้อง สำหรับการกำหนดค่าต่างๆ ตามที่ได้กล่าวไป จะเป็นดังรูปที่ 11




**ภาพผนวกที่ ก11** แสดงการกำหนดค่าต่างๆ ให้กับ controller ของ UART ภายใน microcontroller Samsung S3C4530A

จากการกำหนดค่าให้กับ controller ของ Ethernet 10/100 mbps และ controller ของ UART ซึ่งอยู่ภายใน microcontroller Samsung S3C4530A พบว่าถ้าทำการกำหนดค่าของ Board Implementation ภายในเมนู System Type ให้ตรงกับระบบคอมพิวเตอร์แบบฝังตัวที่เรามีอยู่หรือให้เป็นระบบคอมพิวเตอร์แบบฝังตัวซึ่งใช้งาน microcontroller แบบเดียวกับระบบคอมพิวเตอร์แบบฝังตัวของเรา จะทำให้ตัวเลือกของ driver สำหรับ controller และอุปกรณ์ต่างๆ ที่อยู่ภายใน microcontroller และระบบคอมพิวเตอร์แบบฝังตัวนั้นๆ ปรากฏขึ้นมาตามเมนูต่างๆ โดยอัตโนมัติ

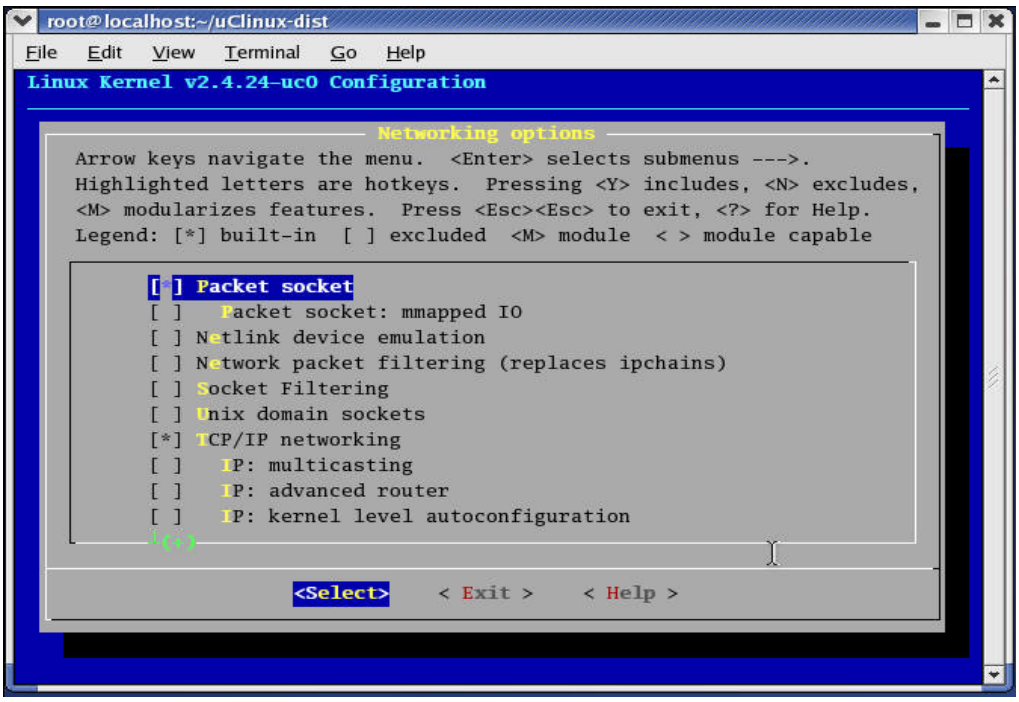
ดังนั้นถ้าในเมนูย่อย Board Implementation ภายในเมนู System Type ไม่มีตัวเลือกใดที่ตรงกับระบบคอมพิวเตอร์แบบฝังตัวที่เรามีอยู่รวมถึงไม่มีตัวเลือกใดที่ใช้ microcontroller ตรงกับระบบคอมพิวเตอร์แบบฝังตัวของเรา ผู้ใช้งานจำเป็นต้องพัฒนาองค์ประกอบต่างๆ สำหรับระบบคอมพิวเตอร์แบบฝังตัวนั้นๆ เข้าไปใน μClinux-dist ด้วยตนเอง ซึ่งองค์ประกอบดังกล่าวมีทั้งการสร้างและแก้ไขไฟล์ configuration, การพัฒนา source code สำหรับการทำงานพิเศษบางอย่างภายในระบบคอมพิวเตอร์แบบฝังตัวนั้นๆ , รวมทั้ง driver ของ controller และอุปกรณ์ต่างๆ ภายใน microcontroller และระบบคอมพิวเตอร์แบบฝังตัวนั้นๆ ที่ยังไม่มีรองรับอยู่ภายใน μClinux-dist โดยการพัฒนาองค์ประกอบต่างๆ เหล่านี้ควรเริ่มศึกษาจากเอกสารคู่มือของระบบคอมพิวเตอร์แบบฝังตัวนั้นๆ เพื่อให้ทราบถึงส่วนประกอบและหน้าที่การทำงานของส่วนประกอบเหล่านั้นอย่างละเอียด และใช้ความรู้ที่ได้นี้ไปพัฒนาองค์ประกอบต่างๆ (ไฟล์ configuration ที่ตรงกับระบบคอมพิวเตอร์แบบฝังตัว, driver ของอุปกรณ์และ controller ต่างๆ ที่มีอยู่ภายในระบบคอมพิวเตอร์แบบฝังตัว เป็นต้น) ภายใน μClinux-dist สำหรับระบบคอมพิวเตอร์แบบฝังตัวนั้นๆ ต่อไป โดยสามารถนำองค์ประกอบที่มีอยู่แล้วภายใน μClinux-dist และมีความใกล้เคียงกับองค์ประกอบที่เราต้องการพัฒนาขึ้น มาใช้เป็นต้นแบบได้

หลังจากกำหนด driver ที่เหมาะสมให้กับ controller ต่างๆ ภายใน microcontroller Samsung S3C4530A ของ NBC board แล้ว ขั้นตอนต่อไปในการกำหนดค่าให้กับ kernel ของระบบปฏิบัติการ μClinux ก็คือ การกำหนด module และ file system ที่จะใช้งานบนระบบปฏิบัติการ μClinux ซึ่งจะกล่าวถึงในหัวข้อถัดไป

3.1.1.3 กำหนด module และ file system ที่จะใช้งานบนระบบปฏิบัติการ μClinux

ขั้นตอนนี้เป็นการกำหนด module พิเศษที่ใช้กับงานเฉพาะอย่างเพิ่มเติมจาก module หลักที่มีเป็นมาตรฐานอยู่แล้วภายในระบบปฏิบัติการ μClinux โดยการที่ kernel ของระบบปฏิบัติการ μClinux จะมี module พิเศษอะไรเพิ่มขึ้นมาบ้างนั้นขึ้นอยู่กับลักษณะและหน้าที่การทำงานของระบบคอมพิวเตอร์แบบฝังตัวที่จะนำระบบปฏิบัติการ μClinux ไปใช้งานด้วยเป็นหลัก

เมื่อตรวจสอบเมนูย่อยต่างๆ ที่ใช้ในการกำหนด module พิเศษให้กับ kernel ของระบบปฏิบัติการ μClinux พบว่าภายในเมนูย่อยดังกล่าวมีการกำหนดค่า module พิเศษมาให้อยู่แล้วโดยอัตโนมัติ สำหรับค่าที่กำหนดเอาไว้แล้วเหล่านี้ได้มาจากไฟล์ configuration config.linux-2.4.x ของ platform ที่เราได้ทำการเลือกไว้จากเมนู Vendor/Product Selection ก่อนหน้านี้ (Samsung S3C4510B) นั่นเอง ซึ่งเราสามารถใช้ค่าที่กำหนดให้กับ module พิเศษภายในเมนูย่อยต่างๆ สำหรับ platform ดังกล่าวกับ NBC board ที่เรามีอยู่ได้ทันที โดยไม่ต้องปรับเปลี่ยนอะไรเพิ่มเติม (นอกจากกรณีที่ต้องการเพิ่ม module พิเศษให้กับ kernel ของระบบปฏิบัติการ μClinux เพื่อทำงานพิเศษบางอย่างบน NBC board โดยเฉพาะ) เนื่องจาก platform ทั้ง 2 (Samsung S3C4510B และ Samsung S3C4530A) มีลักษณะและหน้าที่การทำงานที่คล้ายคลึงกัน คือได้รับการออกแบบมาให้ทำงานทางด้านเน็ตเวิร์คบนเครือข่าย Ethernet ทั้ง 2 platform ดังนั้น module พิเศษที่ควรได้รับการกำหนดเพิ่มขึ้นสำหรับทั้ง 2 platform จึงต้องเป็น module พิเศษในกลุ่มที่เกี่ยวข้องกับงานทางด้านเน็ตเวิร์คโดยmodule พิเศษในกลุ่มดังกล่าวจะรวบรวมอยู่ภายในเมนูย่อย Networking options ซึ่งเมื่อเปิดเมนูย่อยนี้ขึ้นมาพบว่ามีการกำหนดค่าของ module พิเศษสำหรับงานทางด้านเน็ตเวิร์คบางตัวเอาไว้อยู่แล้ว ดังรูปที่ 12



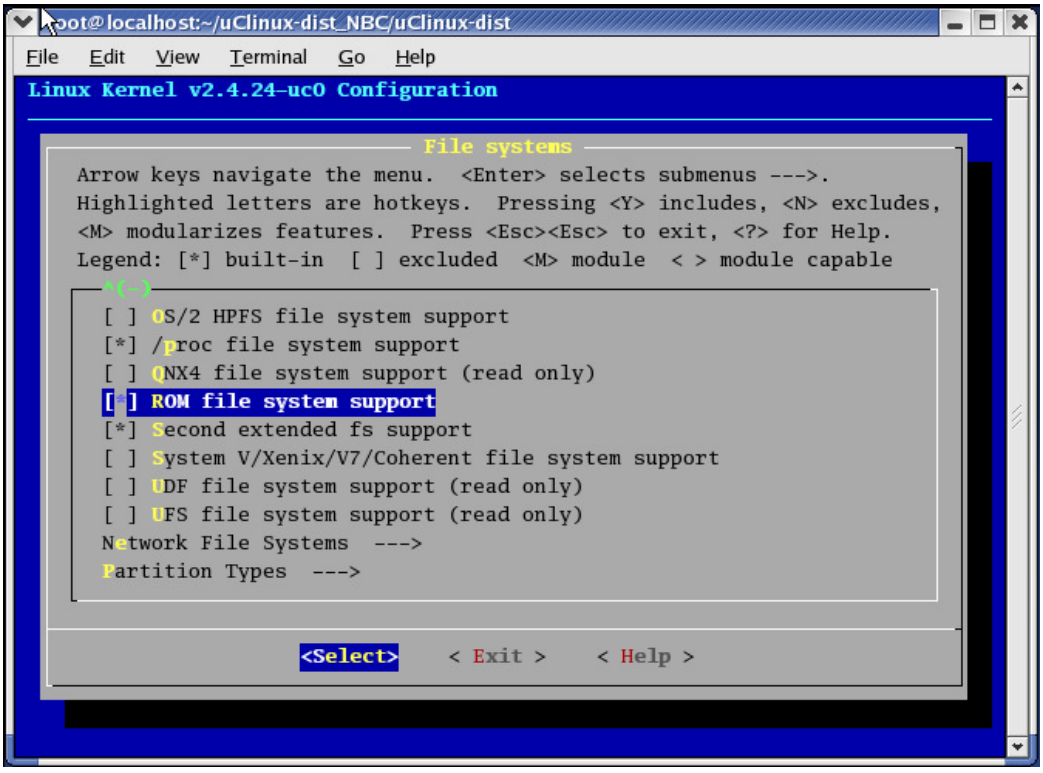

**ภาพผนวกที่ ก12** แสดงการกำหนดค่าของ module พิเศษที่เกี่ยวข้องกับงานทางด้านเน็ตเวิร์คภายในเมนูย่อย Networking options

จากรูปที่ 12 นี้พบว่า module พิเศษที่กำหนดอยู่ภายในเมนูดังกล่าวทั้ง Packet socket และ TCP/IP networking ล้วนแต่เป็น module หลักสำหรับการสื่อสารข้อมูลบนเครือข่าย Ethernet ทั้งสิ้น ดังนั้นเราจึงจำเป็นต้องรวม module พิเศษเหล่านี้ลงใน kernel ของระบบปฏิบัติการ μClinux ที่จะนำไปใช้กับ NBC board ด้วย เพื่อที่ว่าเมื่อทำการติดตั้งระบบปฏิบัติการ μClinux ที่มี module ดังกล่าวลงไปยัง NBC board เรียบร้อยแล้ว ระบบปฏิบัติการ μClinux จะช่วยทำให้ NBC board สามารถทำงานรองรับกับการสื่อสารข้อมูลบนเครือข่าย Ethernet ได้อย่างสมบูรณ์

แต่ในบางครั้งอาจจะต้องทำการกำหนด module พิเศษเพิ่มเติมจากที่มีการกำหนดอยู่ในเมนูย่อยต่างๆ ตามไฟล์ configuration config.linux-2.4.x ของ platform ที่เลือกเอาไว้ในเมนู Vendor/Product Selection เนื่องจาก platform ที่เราเลือกเอาไว้ในเมนูดังกล่าวมีลักษณะและหน้าที่การทำงานไม่ตรงกับ platform ที่เรามีอยู่จริงทั้งหมดหรือเราอาจจะต้องการเพิ่มความสามารถบางอย่างให้กับ kernel ของระบบปฏิบัติการ μClinux เพื่อให้ระบบปฏิบัติการ μClinux สามารถรองรับงานพิเศษบางอย่างบนระบบคอมพิวเตอร์แบบฝังตัวของเราได้ ซึ่งจากกรณีต่างๆ เหล่านี้เราจึงจำเป็นต้องกำหนด module พิเศษบางอย่างเพิ่มเติมเข้าไปจากของเดิมที่มีอยู่แล้ว เพื่อให้ kernel

ของระบบปฏิบัติการ μClinux มี module พิเศษสำหรับทำงานบนระบบคอมพิวเตอร์แบบฝังตัวของเราอย่างครบถ้วน ซึ่งจะทำให้ระบบคอมพิวเตอร์แบบฝังตัวนั้นๆ สามารถทำงานได้อย่างถูกต้องและมีประสิทธิภาพ

นอกจากส่วนของ module พิเศษต่างๆ แล้ว การกำหนดค่าที่สำคัญอีกอย่างหนึ่งภายใน kernel ของระบบปฏิบัติการ μClinux คือเรื่องของ file system ที่จะกำหนดให้ระบบปฏิบัติการ μClinux รองรับ เพื่อจะได้ทำการกำหนดค่าให้กับ module ที่ใช้สำหรับทำงานกับ file system เหล่านั้น ลงไปยังระบบปฏิบัติการ μClinux โดยอัตโนมัติ สำหรับการกำหนดค่าชนิดของ file system ที่ต้องการให้ระบบปฏิบัติการ μClinux สามารถทำงานรองรับได้ ให้เข้าไปกำหนดที่เมนูย่อย File systems ซึ่งอยู่ภายในเมนูหลักสำหรับการกำหนดค่าในด้านต่างๆ ให้กับ kernel ของระบบปฏิบัติการ μClinux โดยภายในเมนูย่อยดังกล่าวจะมี file system หลายชนิดให้เลือก ดังรูปที่ 13




**ภาพผนวกที่ ก13** เมนูย่อย File systems ใช้สำหรับกำหนดชนิดของ file system ที่ต้องการให้ระบบปฏิบัติการ μClinux ทำงานรองรับ

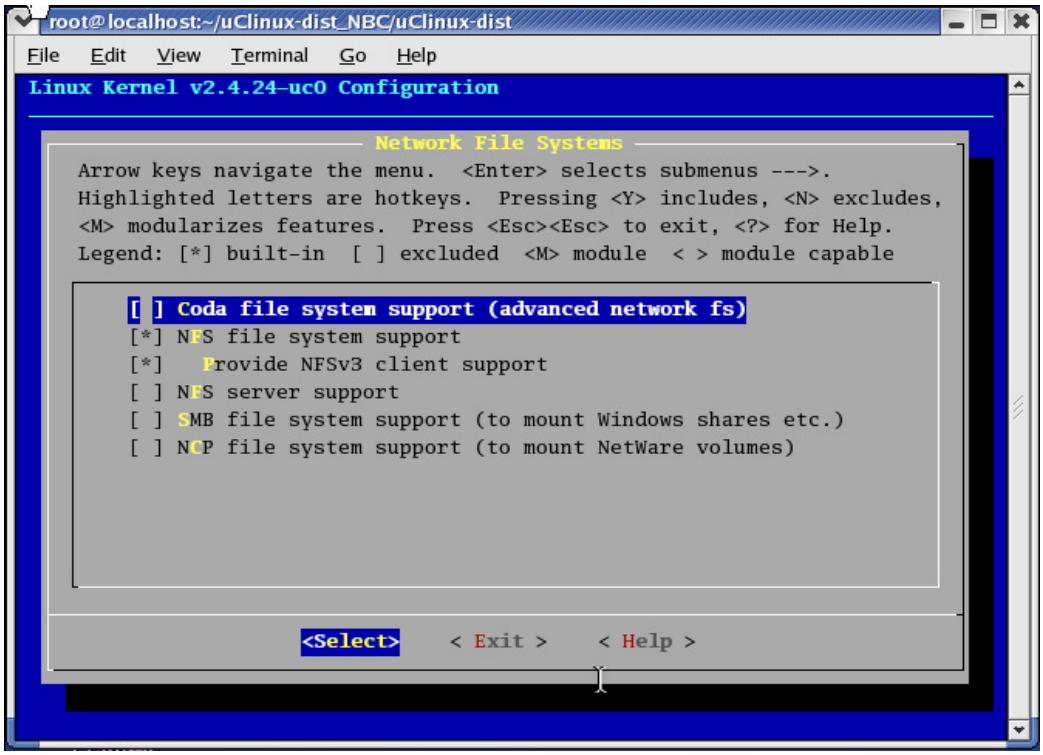
เมื่อเมนูย่อย File systems ปรากฏขึ้นมาพบว่าจะมีการกำหนดค่าของ file system บางตัวที่ต้องการให้ระบบปฏิบัติการ μClinux ทำงานรองรับมาให้โดยอัตโนมัติ ซึ่งค่าที่กำหนดให้กับ file system ดังกล่าวภายในเมนูนี้มาจากไฟล์ configuration config.linux-2.4.x ของ platform ที่ได้เลือกไว้จากเมนู Vendor/Product Selection ก่อนหน้านี้ (Samsung S3C4510B) นั่นเอง โดยเราสามารถใช้ file system ที่กำหนดเอาไว้แล้วเหล่านี้สำหรับ platform S3C4510B มาใช้งานกับ NBC board ได้ทันที โดยไม่ต้องปรับเปลี่ยนอะไรเพิ่มเติม (นอกจากในกรณีที่ต้องการเพิ่ม file system พิเศษบางอย่างให้ระบบปฏิบัติการ μClinux ทำงานรองรับ) เนื่องจาก platform ทั้ง 2 (Samsung S3C4510B และ Samsung S3C4530A) มีลักษณะคล้ายคลึงกัน

จากเมนู File System ในรูปที่ 13 พบว่ามีการกำหนดชนิดของ file system เพื่อให้ระบบปฏิบัติการ μClinux ทำงานรองรับอยู่ 3 ชนิดด้วยกันคือ ROM file system, /proc file system และ ext2 file system โดยเฉพาะ ROM file system ซึ่งเป็น file system แบบ read-only ที่นำมาใช้เป็น root file system ให้กับระบบปฏิบัติการ μClinux ซึ่งข้อดีของการนำ ROM file system มาใช้งาน คือ จะทำให้ image ของระบบปฏิบัติการ μClinux มีขนาดเล็กลง เนื่องจาก module ต่างๆ ของระบบปฏิบัติการ μClinux จะ link รวมกันในขณะทำการ boot ระบบปฏิบัติการ μClinux ขึ้นมาทำงานเท่านั้น (ทำการ boot บน RAM) ซึ่งช่วยประหยัดหน่วยความจำ (Flash ROM) ในการจัดเก็บ image ระบบปฏิบัติการ μClinux ได้เป็นอย่างดี

นอกจาก file system ที่มีการกำหนดค่าเอาไว้แล้วดังกล่าว เราสามารถกำหนดค่าของ file system แบบอื่นให้ระบบปฏิบัติการ μClinux ทำงานรองรับได้อีก เนื่องจากระบบปฏิบัติการ μClinux มีคุณสมบัติสามารถรองรับการทำงานกับ file system ได้หลายรูปแบบเช่นเดียวกับระบบปฏิบัติการ Linux นั่นเอง โดย file system อีกประเภทที่จะกล่าวถึงในที่นี้คือ NFS ซึ่งเป็น file system ที่ช่วยให้ระบบคอมพิวเตอร์แบบฝังตัวสามารถใช้งานไฟล์ที่อยู่บนเครื่องคอมพิวเตอร์เครื่องอื่นๆ ภายในระบบเครือข่ายได้ โดยเครื่องคอมพิวเตอร์ดังกล่าวต้องใช้งานระบบปฏิบัติการ UNIX หรือระบบปฏิบัติการ Linux และต้องทำการแชร์ไฟล์ที่ต้องการให้เครื่องคอมพิวเตอร์เครื่องอื่นๆ เข้าถึงได้เอาไว้ ซึ่งการใช้ NFS นี้จะช่วยให้ระบบคอมพิวเตอร์แบบฝังตัวไม่จำเป็นต้องรวมแอพพลิเคชั่นหรือไฟล์ที่ต้องการใช้งานเอาไว้ภายใน image ของระบบปฏิบัติการ μClinux ทั้งหมด แต่สามารถนำข้อมูลเหล่านั้นไปเก็บและทำการแชร์เอาไว้บนเครื่องคอมพิวเตอร์เครื่องอื่นๆ ภายในระบบเครือข่าย เมื่อมีความจำเป็นต้องใช้งานข้อมูลดังกล่าวก็ทำการ mount ข้อมูลเหล่านั้นเข้ากับระบบคอมพิวเตอร์แบบฝังตัว ซึ่งจะทำให้ระบบคอมพิวเตอร์แบบฝังตัวสามารถเข้าถึงข้อมูลดังกล่าว

ได้ทันทีเสมือนข้อมูลเหล่านั้นอยู่บนระบบคอมพิวเตอร์แบบฝังตัวนั้นๆ โดยตรง การจัดการข้อมูลในลักษณะนี้จะทำช่วยให้ image ของระบบปฏิบัติการ µClinux มีขนาดเล็กลงและเมื่อต้องการพัฒนาแอพพลิเคชั่นใหม่ๆ สำหรับทำงานกับระบบคอมพิวเตอร์แบบฝังตัวก็ไม่จำเป็นต้องทำการคอมไพล์และ build image ของระบบปฏิบัติการ µClinux ขึ้นมาใหม่ทุกครั้ง

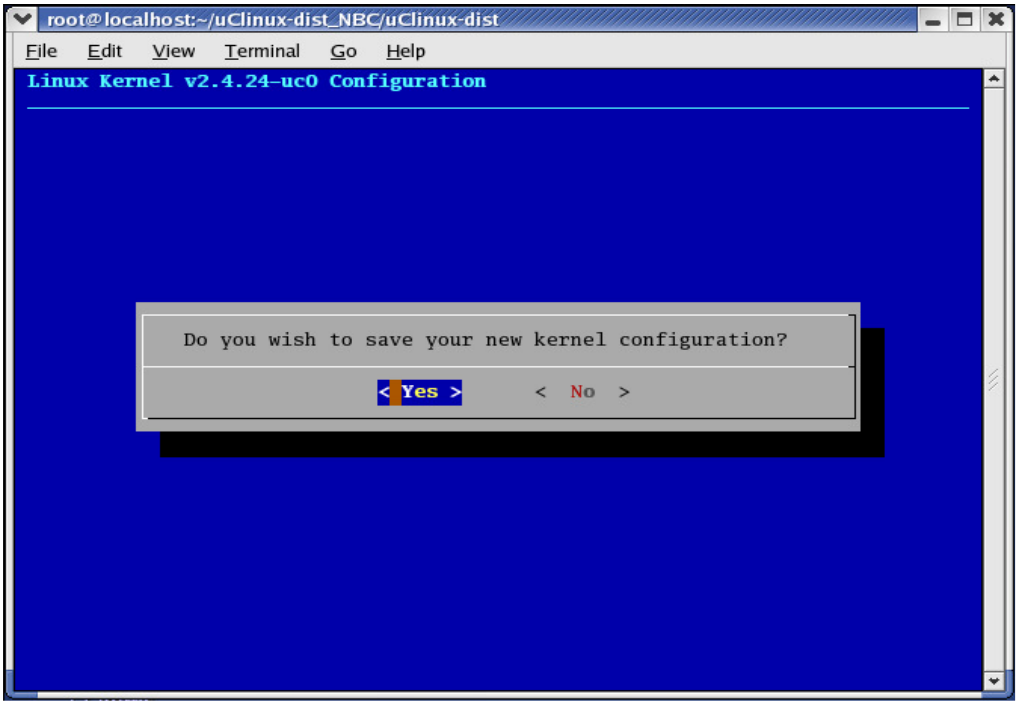
การกำหนดให้ระบบปฏิบัติการ µClinux สามารถทำงานรองรับกับ NFS ได้นั้นให้เข้าไปที่เมนูย่อย Network File System ---> ภายในเมนู File systems ซึ่งมีวิธีการกำหนดค่าให้กับ NFS ดังรูปที่ 14




**ภาพผนวกที่ ก14** แสดงการกำหนดค่าให้ระบบปฏิบัติการ µClinux สามารถทำงานรองรับ NFS

จากรูปที่ 14 พบว่าจะทำการกำหนดเฉพาะส่วนที่เป็น client เท่านั้น เนื่องจากเราจะใช้ระบบคอมพิวเตอร์แบบฝังตัวในการ mount ข้อมูลที่แชร์เอาไว้บนเครื่องคอมพิวเตอร์เครื่องอื่น ซึ่งเครื่องคอมพิวเตอร์ดังกล่าวทำหน้าที่เปรียบเสมือน server ดังนั้นบนระบบปฏิบัติการ µClinux เราจึงกำหนดค่าให้เฉพาะ module ที่ทำหน้าที่เป็น NFS client เท่านั้น

หลังจากกำหนดค่าที่จำเป็นต่างๆ ให้กับเมนูที่ใช้ในการกำหนดค่าสำหรับ kernel ของระบบปฏิบัติการ μClinux เรียบร้อยแล้ว เราสามารถออกจากเมนูดังกล่าวได้ทันที โดยการเลือกไปที่ปุ่ม Exit ตรงด้านล่างของเมนู ซึ่งเมื่อเราเลือกปุ่มดังกล่าวแล้วจะปรากฏข้อความมาถามว่าต้องการบันทึกค่าต่างๆ ของ kernel ที่เราได้กำหนดให้กับเมนูก่อนหน้านี้หรือไม่ ดังรูปที่ 15




**ภาพผนวกที่ ก15** แสดงข้อความยืนยันการบันทึกค่าต่างๆ ที่ได้ทำการกำหนดให้กับ kernel ของระบบปฏิบัติการ μClinux ในเมนูก่อนหน้านี้

จากรูปที่ 15 ถ้าค่าต่างๆ ที่เรากำหนดไปให้กับ kernel ของระบบปฏิบัติการ μClinux ถูกต้องตามความต้องการทั้งหมด เราก็ทำการเลือกที่ปุ่ม Yes เพื่อทำการบันทึกค่าเหล่านี้เอาไว้สำหรับใช้ในขั้นตอนของการคอมไพล์และ build image ของระบบปฏิบัติการ μClinux ต่อไป รวมทั้งถ้าก่อนหน้านี้เราทำการกำหนดค่าให้กับตัวเลือก Update Default Vendor Settings (NEW) ไว้ ค่าต่างๆ ที่เรากำหนดให้กับ kernel ของระบบปฏิบัติการ μClinux สำหรับนำไปทำงานบน NBC board เหล่านี้ จะได้รับการบันทึกเก็บไว้ในไฟล์ configuration config.linux-2.4.x ของ vendor หรือ product ที่เราได้ทำการเลือกไว้ในเมนู Vendor/Product Selection ก่อนหน้านี้ (Samsung S3C4510B) โดยอัตโนมัติ เพื่อสร้างเป็นไฟล์ configuration config.linux-2.4.x สำหรับ NBC board ขึ้นมาโดยเฉพาะ ซึ่งจะช่วยอำนวยความสะดวกในการกำหนดค่าต่างๆ ให้กับ kernel ของ

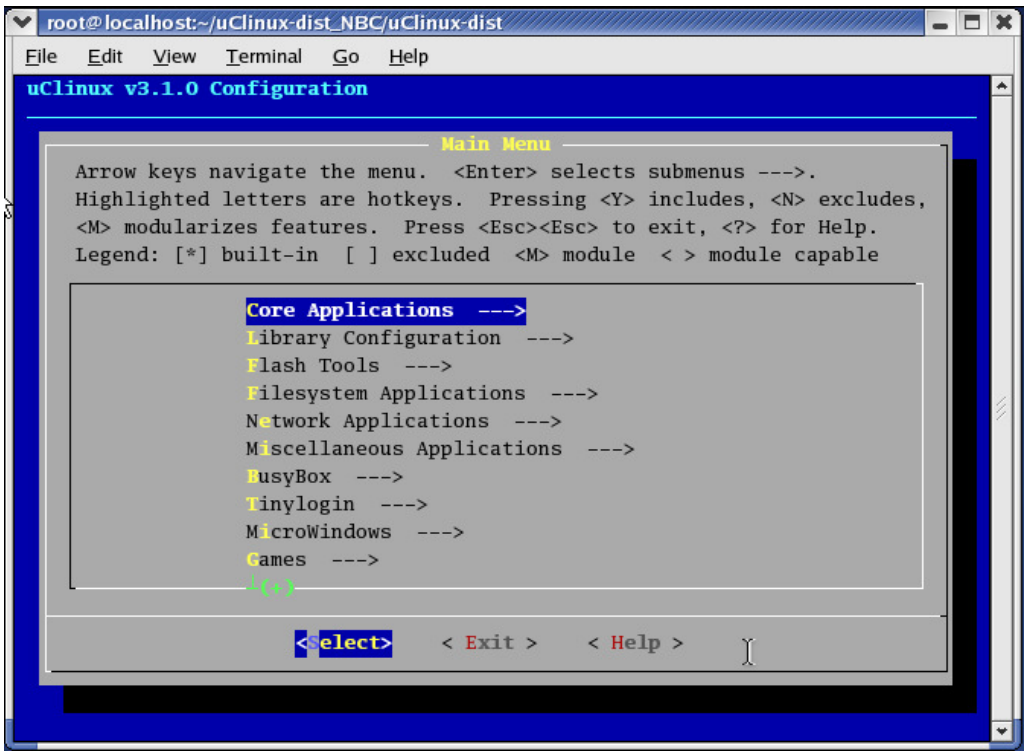
ระบบปฏิบัติการ μClinux สำหรับ NBC board ในคราวต่อๆ ไป แต่ถ้าค่าที่เราได้กำหนดไปยังไม่ถูกต้องก็สามารถทำการเลือกไปที่ No เพื่อออกจากการกำหนดค่าให้กับ kernel ของระบบปฏิบัติการ μClinux ทันที โดยไม่มีการบันทึกค่าต่างๆ ที่ได้กำหนดไปก่อนหน้านี้เก็บไว้เลย

หลังจากทำการกำหนดค่าให้กับ kernel ของระบบปฏิบัติการ μClinux เรียบร้อยแล้วขึ้นตอนต่อไปจะเป็นการกำหนดค่าให้กับแอพพลิเคชั่นที่ต้องการใช้งานบนระบบปฏิบัติการ μClinux ซึ่งการกำหนดค่าดังกล่าวจะกล่าวถึงในหัวข้อถัดไป

3.1.2 ขั้นตอนการกำหนดค่าให้กับแอพพลิเคชั่นที่ต้องการนำมาใช้งานบนระบบปฏิบัติการ μClinux

หลังจากกำหนดค่าต่างๆ ให้กับ kernel ของระบบปฏิบัติการ μClinux ให้ตรงกับ NBC board เรียบร้อยแล้ว ขั้นตอนต่อไปเป็นการกำหนดค่าให้กับแอพพลิเคชั่นต่างๆ ที่ต้องการใช้งานบนระบบปฏิบัติการ μClinux โดยแอพพลิเคชั่นที่เรากำหนดค่าเอาไว้ในขั้นตอนนี้จะได้รับการคอมไพล์และ build รวมอยู่ใน image ของระบบปฏิบัติการ μClinux ถ้าเรายิ่งกำหนดค่าให้กับแอพพลิเคชั่นในขั้นตอนนี้เอาไว้มาก ก็จะทำให้ image ของระบบปฏิบัติการ μClinux มีขนาดใหญ่ตามไปด้วย ดังนั้นเราควรเลือกเฉพาะแอพพลิเคชั่นที่จำเป็นที่เราต้องการใช้งานเท่านั้น เพื่อทำให้ image ของระบบปฏิบัติการ μClinux มีขนาดไม่ใหญ่จนเกินไปนักและสามารถนำไปเก็บลงบน Flash ROM ภายในระบบคอมพิวเตอร์แบบฝังตัวได้

สำหรับเมนูที่ใช้ในการกำหนดค่าให้กับแอพพลิเคชั่นต่างๆ ที่ต้องการนำไปใช้งานบนระบบปฏิบัติการ μClinux จะประกอบด้วยรายการของแอพพลิเคชั่นกลุ่มต่างๆ ดังรูปที่ 16 และเมนูดังกล่าวจะปรากฏขึ้นก็ต่อเมื่อได้ทำการกำหนดค่าให้กับตัวเลือก Customize Vender/User Settings (NEW) เอาไว้ภายในเมนูที่กล่าวถึงไปก่อนหน้านี้เท่านั้น (ภาพผนวกที่ ก3)

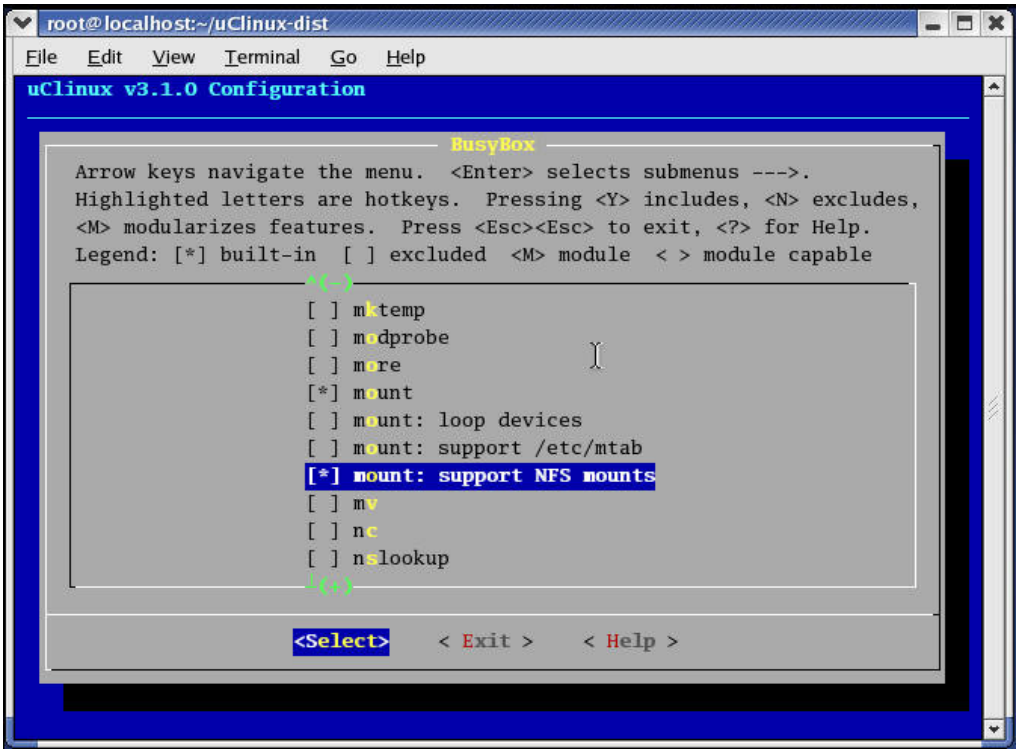



**ภาพผนวกที่ ก16** แสดงเมนูที่ใช้ในการกำหนดค่าของแอพพลิเคชั่นต่างๆ ที่ต้องการนำไปใช้งานบนระบบปฏิบัติการ µClinux

จากรูปที่ 16 เมื่อเปิดเข้าไปในเมนูย่อยต่างๆ พบว่าจะมีการกำหนดค่าของแอพพลิเคชั่นบางแอพพลิเคชั่นเอาไว้ให้อยู่แล้วโดยอัตโนมัติ ซึ่งแอพพลิเคชั่นที่มีการกำหนดเอาไว้ก่อนเหล่านี้มาจากเหล่านี้ได้มาจากไฟล์ configuration config.vendor-2.4.x ของ platform ที่เราได้ทำการเลือกไว้จากเมนู Vendor/Product Selection ก่อนหน้านี้ (Samsung S3C4510B) นั่นเอง โดยเราสามารถใช้แอพพลิเคชั่นที่กำหนดเอาไว้แล้วเหล่านี้สำหรับ platform S3C4510B มาใช้งานกับ NBC board ได้ทันที เนื่องจาก platform ทั้ง 2 (Samsung S3C4510B และ Samsung S3C4530A) มีลักษณะและหน้าที่การทำงานที่คล้ายคลึงกัน จึงต้องการแอพพลิเคชั่นสำหรับใช้งานในลักษณะคล้ายๆ กัน โดยเฉพาะอย่างยิ่งแอพพลิเคชั่นทางด้านเน็ตเวิร์ค เช่น ifconfig, ping, dhcpd เป็นต้น ซึ่งแอพพลิเคชั่นดังกล่าวมีการกำหนดค่าเอาไว้แล้วอย่างครบถ้วน

นอกจากแอพพลิเคชั่นที่มีการกำหนดเอาไว้ให้อยู่แล้วภายในเมนูย่อยต่างๆ เราสามารถกำหนดแอพพลิเคชั่นอื่นๆ ตามที่เราต้องการเพิ่มเข้าไปได้อีกด้วย เพื่อให้บนระบบปฏิบัติการ µClinux มีแอพพลิเคชั่นสำหรับรองรับการใช้งานของผู้ใช้งานอย่างครบถ้วน ยกตัวอย่างเช่น ในที่นี้

จะทำการเพิ่มแอพพลิเคชั่นบางตัวที่ทำงานเกี่ยวกับ NFS ลงไปบนระบบปฏิบัติการ µClinux เพื่อให้ผู้ใช้งานสามารถทำการ mount ใช้งานไฟล์และข้อมูลของเครื่องคอมพิวเตอร์เครื่องอื่นในระบบเครือข่ายมาใช้งานบน NBC board ได้ โดยแอพพลิเคชั่นตัวนี้จะอาศัยการทำงานของ module ต่างๆ ที่เกี่ยวข้องกับ NFS ภายใน kernel ของระบบปฏิบัติการ µClinux อีกต่อหนึ่ง ซึ่ง module ดังกล่าวเราได้กำหนดค่าเอาไว้แล้วในขั้นตอนก่อนหน้านี้ (ขั้นตอน 3.1.1.3) โดยวิธีการกำหนดค่าให้กับแอพพลิเคชั่นดังกล่าว ให้เราเข้าไปในเมนูย่อย BusyBox ซึ่งเป็นแอพพลิเคชั่นที่รวบรวม utility หลักๆ ของระบบปฏิบัติการ UNIX ให้มาทำงานอยู่บนระบบปฏิบัติการ µClinux หลังจากนั้นให้ทำการกำหนดค่าให้กับตัวเลือก mount : support NFS mounts เพื่อติดตั้งแอพพลิเคชั่นที่ใช้ในการ mount NFS ลงบนระบบปฏิบัติการ µClinux ต่อไป ดังรูปที่ 17



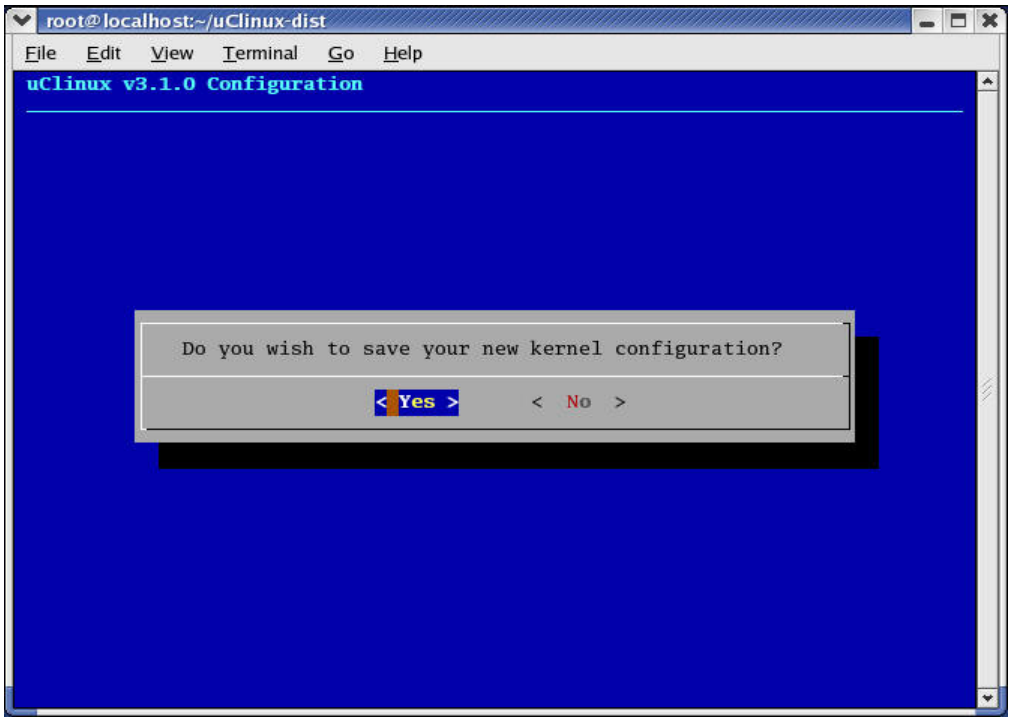

**ภาพผนวกที่ ก17** แสดงการกำหนดค่าให้กับแอพพลิเคชั่นที่ใช้ในการ mount NFS

นอกจากนี้เรายังสามารถกำหนดให้ไม่ต้องติดตั้งแอพพลิเคชั่นบางตัวที่มีการกำหนดเอาไว้ให้อยู่แล้วภายในเมนูย่อยต่างๆ ได้อีกด้วย เพื่อช่วยลดขนาด image ของระบบปฏิบัติการ µClinux ให้เล็กลง แต่การที่เราจะกำหนดให้ไม่ต้องติดตั้งแอพพลิเคชั่นใดลงไปใน image ของระบบปฏิบัติการ µClinux นั้นต้องอาศัยความระมัดระวังมากพอสมควรในการพิจารณาว่า

แอพพลิเคชั่นใดที่ไม่จำเป็นต้องใช้งานจริงๆ เนื่องจากแอพพลิเคชั่นที่มีการกำหนดเอาไว้ให้อยู่แล้วโดยอัตโนมัติภายในเมนูย่อยต่างๆ ส่วนใหญ่จะใช้ในการเริ่มต้นการทำงานของระบบปฏิบัติการ μClinux หรือรองรับการใช้งานขั้นพื้นฐานจากผู้ใช้บนระบบคอมพิวเตอร์แบบฝังตัวนั้นๆ เช่น ifconfig สำหรับให้ผู้ใช้งานตรวจสอบรายละเอียดของอินเทอร์เฟสเน็ตเวิร์คต่างๆ บนระบบคอมพิวเตอร์แบบฝังตัวนั้นๆ, dhcpd เป็นเซอร์วิสที่ช่วยกำหนด IP address ให้กับอินเทอร์เฟสของ Ethernet บนระบบคอมพิวเตอร์แบบฝังตัวนั้นๆ ซึ่งเซอร์วิสดังกล่าวระบบปฏิบัติการ μClinux จะเป็นผู้เรียกให้ทำงานโดยอัตโนมัติ หลังจากที่ระบบปฏิบัติการ μClinux boot ขึ้นมาทำงานบนระบบคอมพิวเตอร์แบบฝังตัวนั้นๆ เรียบร้อยแล้ว (ตามคำสั่งที่อยู่ในไฟล์ script rc) เป็นต้น ซึ่งถ้าเรากำหนดให้ไม่ต้องติดตั้งแอพพลิเคชั่นเหล่านี้ลงไปบน image ของระบบปฏิบัติการ μClinux จะทำให้การทำงานบางอย่างของระบบปฏิบัติการ μClinux บนระบบคอมพิวเตอร์แบบฝังตัวนั้นๆ ไม่ถูกต้องและจะส่งผลให้ระบบคอมพิวเตอร์แบบฝังตัวนั้นๆ ทำงานได้ไม่เต็มประสิทธิภาพ ดังนั้นโดยปกติแล้วเราจะพยายามไม่ปรับเปลี่ยนหรือลดทอนแอพพลิเคชั่นที่มีการกำหนดเอาไว้ให้อยู่แล้วภายในเมนูย่อยต่างๆ โดยอัตโนมัติสำหรับ platform นั้นๆ (ในที่นี้หมายถึง Samsung S3C4510B ตามที่เราได้เลือกเอาไว้ในเมนู Vendor/Product Selection ก่อนหน้านี้) นอกจากมีความจำเป็นที่ไม่สามารถหลีกเลี่ยงได้ เช่น image ของระบบปฏิบัติการ μClinux ที่ build เรียบร้อยแล้วมีขนาดใหญ่เกินกว่าจะนำไปติดตั้งลงใน Flash ROM เป็นต้น ในกรณีเช่นนี้เราจำเป็นต้องลดทอนแอพพลิเคชั่นบางตัวออกจาก image ของระบบปฏิบัติการ μClinux ซึ่งสามารถทำได้โดยตรวจสอบจากไฟล์ script ที่ใช้ในการเริ่มต้นการทำงานของระบบปฏิบัติการ μClinux บน platform นั้นๆ (ในที่นี้คือ Samsung S3C4510B) ก่อน เช่น ไฟล์ rc, ไฟล์ inittab เป็นต้น เพื่อตรวจดูว่าไฟล์ script ดังกล่าวมีการเรียกใช้แอพพลิเคชั่นใดขึ้นมาทำงานบ้าง ซึ่งแอพพลิเคชั่นในกลุ่มนี้เราไม่ควรตัดออกจาก image ของระบบปฏิบัติการ μClinux หลังจากนั้นให้ตรวจสอบรายชื่อของแอพพลิเคชั่นทั้งหมดที่มีการกำหนดอยู่ภายในเมนูย่อยต่างๆ ทุกเมนู และใช้ help ตรวจดูหน้าที่การทำงานของแอพพลิเคชั่นเหล่านั้นว่าใช้สำหรับทำอะไร ถ้าตรวจดูแล้วว่าแอพพลิเคชั่นใดไม่มีความจำเป็นสำหรับใช้งานบนระบบคอมพิวเตอร์แบบฝังตัวของเรา (ในที่นี้หมายถึง NBC board) และไม่ปรากฏอยู่ในไฟล์ script สำหรับเริ่มต้นการทำงานของระบบปฏิบัติการ μClinux ตามที่ได้ตรวจสอบก่อนหน้านี้ เราก็สามารถตัดแอพพลิเคชั่นดังกล่าวออกจาก image ของระบบปฏิบัติการ μClinux ได้ทันที

หลังจากทำการกำหนดแอพพลิเคชั่นต่างๆ ที่ต้องการใช้งานบนระบบปฏิบัติการ μClinux เรียบร้อยแล้ว เราสามารถออกจากเมนูดังกล่าว (เมนูในภาพผนวกที่ ก16) ได้ทันที โดยการเลือกไป

ที่ปุ่ม Exit ตรงด้านล่างของเมนู ซึ่งเมื่อเราเลือกปุ่มดังกล่าวแล้วจะปรากฏข้อความมาถามว่าต้องการบันทึกค่าต่างๆ ของแอพพลิเคชั่นที่เราได้กำหนดให้กับเมนูก่อนหน้านี้หรือไม่ ดังรูปที่ 18




**ภาพผนวกที่ ก18** แสดงข้อความยืนยันการบันทึกค่าของแอพพลิเคชั่นที่ต้องการให้มีใช้งานอยู่บนระบบปฏิบัติการ μClinux ตามที่ได้กำหนดในเมนูก่อนหน้านี้

จากรูปที่ 18 ถ้าเราทำการกำหนดแอพพลิเคชั่นที่ต้องการนำไปใช้งานบนระบบปฏิบัติการ μClinux สำหรับ NBC board ถูกต้องครบถ้วนทั้งหมดแล้ว เราจะต้องเลือกที่ปุ่ม Yes เพื่อทำการบันทึกค่าเหล่านี้เอาไว้สำหรับใช้ในขั้นตอนของการคอมไพล์และ build image ของระบบปฏิบัติการ μClinux ต่อไป รวมทั้งถ้าก่อนหน้านี้เราทำการกำหนดค่าให้กับตัวเลือก Update Default Vendor Settings (NEW) ไว้ ค่าของแอพพลิเคชั่นต่างๆ ที่เรากำหนดให้ติดตั้งลงบนระบบปฏิบัติการ μClinux (แอพพลิเคชั่นที่ต้องการให้ build รวมอยู่ใน image ของระบบปฏิบัติการ μClinux) สำหรับนำไปทำงานบน NBC board เหล่านี้ จะได้รับการบันทึกเก็บไว้ในไฟล์ configuration config.vendor-2.4.x ของ vendor หรือ product ที่เราได้ทำการเลือกไว้ในเมนู Vendor/Product Selection ก่อนหน้านี้ (Samsung S3C4510B) โดยอัตโนมัติ เพื่อสร้างเป็นไฟล์ configuration config.vendor-2.4.x สำหรับ NBC board ขึ้นมาโดยเฉพาะ ซึ่งจะช่วยอำนวยความสะดวกในการกำหนดค่าของแอพพลิเคชั่นที่จะนำมาใช้งานบนระบบปฏิบัติการ μClinux สำหรับ NBC board ใน

คราวต่อๆ ไป แต่ถ้าค่าที่เราได้กำหนดไปยังไม่ถูกต้องก็สามารถทำการเลือกไปที่ No เพื่อออกจากการกำหนดค่าให้กับแอพพลิเคชั่นของระบบปฏิบัติการ μClinux ทันที โดยไม่มีการบันทึกค่าต่างๆ ที่ได้กำหนดไปก่อนหน้านี้เก็บไว้เลย

หลังจากที่เราทำการกำหนดค่าให้กับ kernel และแอพพลิเคชั่นต่างๆ ของระบบปฏิบัติการ μClinux สำหรับ NBC board เรียบร้อยแล้ว เราจะใช้ค่าต่างๆ เหล่านี้ในขั้นตอนของการคอมไพล์และ build image ของระบบปฏิบัติการ μClinux ขึ้นมา เพื่อให้ได้ image ของระบบปฏิบัติการ μClinux ที่มี kernel และแอพพลิเคชั่นต่างๆ เหมาะสมสำหรับการทำงานบน NBC board โดยเฉพาะ ซึ่งขั้นตอนในการคอมไพล์และ build องค์ประกอบต่างๆ เพื่อสร้างเป็น image ของระบบปฏิบัติการ μClinux ขึ้นมาจะกล่าวในหัวข้อถัดไป

3.2 ขั้นตอนในการคอมไพล์และ build ระบบปฏิบัติการ μClinux สำหรับ NBC board

ค่าที่เราได้กำหนดให้กับ kernel และแอพพลิเคชั่นต่างๆ ของระบบปฏิบัติการ μClinux สำหรับ NBC board ในขั้นตอนที่ผ่านมาจะทำหน้าที่กำหนดองค์ประกอบต่างๆ (ไฟล์ source code ของแอพพลิเคชั่น, ไฟล์ source code ของ driver) ภายใน μClinux-dist ที่จะต้องนำมาทำการคอมไพล์และ build รวมเข้าด้วยกัน เพื่อสร้างเป็น image ของระบบปฏิบัติการ μClinux ที่มี kernel และแอพพลิเคชั่นต่างๆ เหมาะสมสำหรับการทำงานบน NBC board โดยเฉพาะ

ก่อนที่จะเริ่มทำการคอมไพล์และ build image ของระบบปฏิบัติการ μClinux เราจะต้องตรวจสอบและแก้ไขไฟล์ภายใน μClinux-dist บางอย่างเพื่อให้ขั้นตอนของการคอมไพล์และ build image ของระบบปฏิบัติการ μClinux เป็นไปอย่างถูกต้อง ซึ่งจะแก้ไขและตรวจสอบรายละเอียดในไฟล์ใดของ μClinux-dist บ้างนั้นขึ้นอยู่กับ platform และความต้องการของผู้ใช้เป็นหลัก โดยผู้ใช้งานต้องตรวจสอบรายละเอียดและหน้าที่ของไฟล์ต่างๆ ภายใน μClinux-dist โดยละเอียดเพื่อจะได้ใช้ข้อมูลดังกล่าวในการเลือกไฟล์ที่จะต้องตรวจสอบและแก้ไขให้ถูกต้องเหมาะสมตาม platform และตามความต้องการของผู้ใช้งานต่อไป สำหรับ NBC board จะมีการตรวจสอบและแก้ไขไฟล์ต่างๆ ดังนี้

1) ตรวจสอบค่าที่กำหนดให้กับตัวแปร TEXTADDR ของแต่ละ platform

เราสามารถตรวจสอบค่าของตัวแปรดังกล่าวได้จาก Makefile ภายใน directory /linux-2.4.x/arch/armnommu/ ที่ต้องใช้ Makefile ภายใน directory ดังกล่าวเนื่องจากเราเลือกใช้ kernel ของ ระบบปฏิบัติการ μClinux เป็นเวอร์ชัน 2.4 (linux-2.4.x) และ NBC board ของเราใช้ microcontroller Samsung S3C4530A ซึ่งหน่วยประมวลผลเป็น ARM7TDMI ที่ไม่มี MMU อยู่ภายในนั่นเอง (/arch/armnommu) โดยภายใน Makefile ดังกล่าวจะมีการกำหนดค่า TEXTADDR สำหรับหลายๆ platform เอาไว้ ดังนั้นจึงต้องเลือก platform ให้ถูกต้อง เพื่อจะได้ทราบค่าของตัวแปร TEXTADDR ที่กำหนดให้กับ platform ของเรา ในที่นี้ให้เลือกที่ platform EVS3C4530HEI ตามที่เราได้กำหนดให้กับตัวเลือก Board Implementation ในเมนู System Type (จากหัวข้อ 3.1.1.1) ก่อนหน้านี้

เมื่อตรวจสอบภายใน Makefile ดังกล่าวแล้วพบว่าค่าของตัวแปร TEXTADDR สำหรับ platform ที่เราต้องการได้รับการกำหนดให้มีค่าเป็น 0x00020000 ซึ่งค่านี้ก็คือ address ของหน่วยความจำ (RAM) ที่ใช้เป็นจุดเริ่มต้นการทำงานของ kernel ของระบบปฏิบัติการนั่นเอง โดยในขั้นตอนของการคอมไพล์และ build สร้าง image ของระบบปฏิบัติการ μClinux ขึ้นมา เครื่องมือต่างๆ ที่ใช้ในขั้นตอนนี้จะอาศัยค่าจากตัวแปรดังกล่าวเป็นaddress เริ่มต้นในการ link คำสั่งและ symbol ต่างๆ ภายใน kernel ของระบบปฏิบัติการ μClinux เข้าด้วยกัน ซึ่งการจะทำให้ kernel ของระบบปฏิบัติการ μClinux สามารถทำงานได้อย่างถูกต้องได้นั้น เราต้องกำหนดให้ bootloader โหลด kernel ของระบบปฏิบัติการจากที่เก็บอยู่ใน ROM ไปเริ่มเขียนลงใน address ดังกล่าวภายใน RAM เพื่อให้ symbol ต่างๆ ภายใน kernel ของระบบปฏิบัติการสามารถ link ถึงกันได้อย่างถูกต้อง หลังจากนั้น bootloader จะเริ่มอ่านคำสั่งจาก address ดังกล่าวของหน่วยความจำ (RAM) ขึ้นมาทำงาน เพื่อเริ่มต้นการทำงานของระบบปฏิบัติการ μClinux บนระบบคอมพิวเตอร์แบบฝังตัว ดังนั้นเราจะต้องจดจำค่าภายในตัวแปร TEXTADDR สำหรับ platform ของเราเอาไว้สำหรับนำไปใช้กำหนดให้กับ bootloader ต่อไป

2) กำหนดค่าของ environment variable ที่จำเป็นสำหรับการคอมไพล์และ build image ของระบบปฏิบัติการ μClinux

เราสามารถตรวจสอบและกำหนดค่า environment variable สำหรับการคอมไพล์และ build image ของระบบปฏิบัติการได้จากไฟล์ Makefile ภายใน directory /linux-2.4.x/ โดย environment variable ส่วนใหญ่จะได้รับการกำหนดค่าเอาไว้ให้อยู่แล้ว แต่จะมีตัวแปรที่สำคัญอีก 2 ตัวที่เราต้องทำการกำหนดค่าด้วยตนเองเพื่อให้เหมาะสมกับลักษณะ platform ของระบบคอมพิวเตอร์แบบฝังตัวที่เรามีอยู่ สำหรับตัวแปรดังกล่าวคือ ตัวแปร ARCH และ CROSS_COMPILE ซึ่งเมื่อเราตรวจสอบใน Makefile ดังกล่าวพบว่ามีการกำหนดค่าให้กับตัวแปรทั้งสองเอาไว้หลายค่าตามแต่ละ platform และแต่ละ platform จะแยกกันอยู่คนละบรรทัดโดยเริ่มต้นทุกบรรทัดจะมีการ comment (#) เอาไว้ การจะกำหนดค่าให้กับตัวแปรทั้งสองจึงสามารถทำได้โดยตัด comment (#) ที่อยู่หน้าบรรทัดของ platform ที่เราต้องการออกไป ซึ่งหน้าที่และการกำหนดค่าให้กับตัวแปรทั้งสองสำหรับ NBC board ของเราเป็นดังนี้

- ARCH := armnommu เป็นตัวแปรที่ใช้กำหนด architecture ของระบบคอมพิวเตอร์แบบฝังตัวของเรา ซึ่งจะเป็นการกำหนด directory หลักที่เก็บไฟล์ต่างๆ ที่จำเป็นสำหรับใช้ในการคอมไพล์และ build image ของระบบปฏิบัติการ μClinux ให้ตรงกับ platform ที่เรามีอยู่ โดยในที่นี้เราจะกำหนดค่าของตัวแปรดังกล่าวให้เป็น armnommu เพื่อชี้ไปยัง directory /linux-2.4/arch/armnommu ซึ่งเก็บไฟล์ต่างๆ ที่ใช้ในการคอมไพล์และ build image ของระบบปฏิบัติการ μClinux สำหรับ platform ของระบบคอมพิวเตอร์แบบฝังตัวที่ใช้หน่วยประมวลผล ARM แบบที่ไม่มี MMU อยู่ภายใน (เรียกว่า armnommu) โดยเฉพาะ และที่เราต้องกำหนดค่าของตัวแปรดังกล่าวให้มีค่าเช่นนี้เนื่องจาก NBC board ของเราใช้ microcontroller Samsung S3C4530A ซึ่งหน่วยประมวลผลเป็น ARM7TDMI ที่ไม่มี MMU อยู่ภายในนั่นเอง

- CROSS_COMPILE = arm-elf- เป็นตัวแปรที่ใช้กำหนดชื่อเริ่มต้นของเครื่องมือต่างๆ ที่จะใช้ในการคอมไพล์และ build image ของระบบปฏิบัติการ μClinux ให้ตรงกับ platform ที่เรามีอยู่ โดยในที่นี้จะกำหนดค่าของตัวแปรดังกล่าวให้เป็น arm-elf- เพื่อให้เหมาะสมกับ NBC board ของเราที่ใช้ ARM7TDMI เป็นหน่วยประมวลผล โดยเมื่อนำค่าในตัวแปรดังกล่าวไปรวมกับชื่อเรียกเครื่องมือต่างๆ เช่น gcc, ar, และ ld เป็นต้น ก็จะได้เป็น arm-elf-gcc, arm-elf-ar, และ arm-elf-ld เป็นต้น ซึ่งเป็นชื่อของ cross-tool ที่ถูกต้องสำหรับนำมาใช้ในการคอมไพล์และ build image ของ

ระบบปฏิบัติการ μClinux ให้กับ platform ของระบบคอมพิวเตอร์แบบฝังตัวที่ใช้หน่วยประมวลผลในตระกูล ARM

3) กำหนดให้ link romfs เข้ากับ image ของระบบปฏิบัติการ μClinux

โดยทั่วไปการติดตั้งระบบปฏิบัติการ μClinux ลงไปทำงานบนระบบคอมพิวเตอร์แบบฝังตัวได้นั้นต้องอาศัยไฟล์ image ที่ build ขึ้นมา 2 ตัวด้วยกันคือ ไฟล์ image.bin ที่มีเฉพาะ kernel ของระบบปฏิบัติการ μClinux อยู่ภายในและไฟล์ romfs.img ที่เป็นไฟล์ image ของ romfs ซึ่งเราจะต้องติดตั้งไฟล์ทั้งสองนี้ลงไปในระบบคอมพิวเตอร์แบบฝังตัวของเราโดยเราจะต้องกำหนดให้ไฟล์ทั้งสอง link ถึงกันภายในระบบคอมพิวเตอร์แบบฝังตัวของเราให้ถูกต้อง ถ้าทั้งสองไฟล์ไม่สามารถ link ถึงกันได้อย่างถูกต้องจะส่งผลให้ระบบปฏิบัติการ μClinux ทำงานผิดพลาด

เพื่อความสะดวกในการติดตั้งระบบปฏิบัติการ μClinux ลงไปใช้งานบนระบบคอมพิวเตอร์แบบฝังตัวของเรา จึงได้ทำการ link romfs กับ kernel ของระบบปฏิบัติการ μClinux เข้าด้วยกัน เพื่อสร้างเป็น image ของระบบปฏิบัติการ μClinux ที่มีทั้ง kernel ของระบบปฏิบัติการ μClinux และ romfs อยู่ภายในขึ้นมา ดังนั้นเราจึงสามารถใช้แค่ไฟล์ image ดังกล่าวเพียงไฟล์เดียวเท่านั้นสำหรับการติดตั้งระบบปฏิบัติการ μClinux ลงไปทำงานบนระบบคอมพิวเตอร์แบบฝังตัว โดยวิธีการกำหนดให้ทำการ link romfs เข้ากับ kernel ของระบบปฏิบัติการ μClinux โดยอัตโนมัติมี 2 ขั้นตอนดังนี้

- เพิ่มคำสั่ง arm-elf-ld –r –o ${ROOTDIR}/${LINUXDIR}/romfs.o – b binary ${ROMFSIMG} ลงไปใต้การเรียกใช้คำสั่ง genromfs ของ target image: ในไฟล์ Makefile ภายใน directory /vendors/Samsung/4510B/ เพื่อทำการ link ไฟล์ romfs.o ไปเป็นไฟล์ romfs.img
- เพิ่ม script ลงไปตรงส่วนของ Global offset table ในไฟล์ vmlinux-armv.lds.in ภายใน directory /linux-2.4.x/arch/armnommu/ ซึ่งไฟล์ดังกล่าวเป็นไฟล์ script ที่ใช้สำหรับกำหนดการ link องค์ประกอบต่างๆ ของ kernel ของระบบปฏิบัติการ μClinux เข้าด้วยกัน โดย script ที่จะเพิ่มลงไปในไฟล์ดังกล่าวเป็นดังนี้

```
*(.got)
romfs_data = .;
romfs.o
romfs_data_end = .;
_end_romfs = .;
```

script ดังกล่าวนี้ใช้สำหรับกำหนดให้ทำการ link romfs เข้ากับ kernel ของระบบปฏิบัติการ μClinux

เมื่อตรวจสอบและแก้ไขไฟล์ภายใน μClinux-dist ตามขั้นตอนทั้งหมดข้างต้นเรียบร้อยแล้ว เราก็สามารถเริ่มต้นคอมไพล์และ build image ของระบบปฏิบัติการ μClinux ได้ทันที แต่ในกรณีของ NBC boardเราจะทำการสร้าง zImage ขึ้นมา โดย zImage ดังกล่าวก็คือ image ของระบบปฏิบัติการ μClinux ที่ได้รับการ compress ให้มีขนาดเล็กลงนั่นเอง ซึ่ง zImage นี้จะมีขนาดเล็กเพียงพอที่จะนำไปติดตั้งลงในหน่วยความจำ flash ROMภายใน NBC board ของเรา สำหรับการที่เราจะ build zImage ให้เหมาะสมกับ NBC board ของเราได้นั้น จำเป็นต้องแก้ไขไฟล์บางไฟล์ดังนี้

1) กำหนดเลขที่หน่วยความจำภายในตัวแปรต่างๆ ที่ใช้ในการ build zImage ให้เหมาะสมกับ NBC board

- ZTEXTADDR เป็นตัวแปรที่เก็บเลขที่หน่วยความจำเริ่มต้นของ zImage เอาไว้ โดยเราจะต้องนำเลขที่หน่วยความจำภายในตัวแปรดังกล่าวไปกำหนดให้กับ bootloader ภายใน NBC board เพื่อใช้กำหนดให้ bootloader นำ zImage ของระบบปฏิบัติการ μClinux ที่เราทำการติดตั้งลงไปใช้งานบน NBC board มาเริ่มต้นบันทึกลงใน flash ROM ณ ตำแหน่งดังกล่าวเสมอและเมื่อมีการ boot ระบบปฏิบัติการ μClinux ขึ้นมาทำงานบน NBC board bootloader จะเรียกให้ decompressor เริ่มทำการ decompress zImage ของระบบปฏิบัติการ μClinux ที่อยู่ภายใน flash ROM ณ ตำแหน่งนี้ด้วยเช่นกัน ซึ่งจะทำให้ระบบปฏิบัติการ μClinux สามารถเริ่มต้นการทำงานบน NBC board ได้อย่างถูกต้อง ในที่นี้จะกำหนดให้ค่า ZTEXTADDR สำหรับ NBC board ของเรามีค่าเป็น 0x00008000

- ZRELADDR เป็นตัวแปรที่ใช้เก็บเลขที่ของหน่วยความจำ RAM ซึ่ง bootloader จะใช้ในการนำเอาข้อมูลของ zImage ที่ถูก decompress แล้วมาเริ่มเขียนลงในหน่วยความจำ RAM ณ

ตำแหน่งนี้ เพื่อให้การทำงานของระบบปฏิบัติการ μClinux เป็นไปอย่างถูกต้อง และโดยส่วนใหญ่ค่าที่อยู่ภายในตัวแปร ZRELADDR สำหรับระบบคอมพิวเตอร์แบบฝังตัว platform ใดๆ จะเป็นค่าเดียวกันกับที่กำหนดอยู่ภายในตัวแปร TEXTADDR ของระบบคอมพิวเตอร์แบบฝังตัว platform นั้นๆ ดังนั้นค่าของตัวแปร ZRELADDR สำหรับ NBC board เราจะกำหนดให้มีค่าเป็น 0x00020000 เช่นเดียวกับที่กำหนดอยู่ภายในตัวแปร TEXTADDR ของ platform ดังกล่าว

สำหรับการกำหนดค่าเลขที่หน่วยความจำให้กับตัวแปรทั้งสองดังกล่าวสามารถทำได้ที่ไฟล์ Makefile ซึ่งอยู่ภายใน directory /linux-2.4.x/arch/armnommu/boot/ โดยเมื่อตรวจสอบในไฟล์ดังกล่าวพบว่ามีการกำหนดค่าของตัวแปร ZTEXTADDR และ ZRELADDR แยกกันไปตามแต่ละ platform ของระบบคอมพิวเตอร์แบบฝังตัว สำหรับ NBC board ของเราให้เพิ่ม code ที่อยู่ด้านล่างนี้ลงไปภายในไฟล์ดังกล่าว

```
ifeq ($(CONFIG_BOARD_EVS3C4530HEI),y)
ZRELADDR = 0x00020000
ZTEXTADDR = 0x00008000
endif
```

จาก code ด้านบนพบว่าเราจะใช้ค่าภายในตัวแปร CONFIG_BOARD_EVS3C4530HEI ในการตรวจสอบเงื่อนไขสำหรับ NBC board ของเรา เนื่องจากเราได้กำหนดค่าของตัวเลือก Board Implementation ภายในเมนู System Type สำหรับ NBC board ของเรา ให้เป็น S3C4530-HEI นั่นเอง (ตามหัวข้อ 3.1.1.1) และเมื่อพิจารณาจากเลขที่ของหน่วยความจำที่กำหนดให้กับตัวแปรต่างๆ สามารถสรุปได้ว่า เราต้องทำการกำหนดให้ bootloader ภายใน NBC board ของเรานำ zImage ของระบบปฏิบัติการ μClinux ที่ต้องการใช้งานมาบันทึกเก็บไว้ที่หน่วยความจำ flash ROM โดยเริ่มทำการบันทึกที่ตำแหน่ง 0x00008000 ของ flash ROM ตามค่าที่กำหนดอยู่ภายในตัวแปร ZTEXTADDR หลังจากนั้นเมื่อต้องการ boot ระบบปฏิบัติการ μClinux ขึ้นมาใช้งานบน NBC board ของเรา bootloader จะทำหน้าที่เรียกให้ decompressor เริ่มทำงานโดยทำการกำหนดให้เริ่มทำการ decompress ข้อมูลที่อยู่ภายใน flash ROM ณ ตำแหน่ง 0x00008000 ตามค่าที่กำหนดเอาไว้ในตัวแปร ZTEXTADDR โดยข้อมูลต่างๆ ที่ decompressor ทำการ decompress ออกมาได้จะถูกนำไปเขียนลงในหน่วยความจำ RAM อีกต่อหนึ่ง โดยจะเริ่มเขียนที่ตำแหน่ง 0x00020000 ของหน่วยความจำ RAM ตามค่าที่กำหนดอยู่ภายในตัวแปร ZRELADDR เพื่อเริ่มการทำงานของระบบปฏิบัติการ μClinux ต่อไป

2) กำหนดให้สร้างไฟล์ piggy.gz ขึ้นมาใช้งาน

ไฟล์ piggy.gz คือไฟล์ zImage ของระบบปฏิบัติการ μClinux ที่ถูกนำมา compress ทั้งไฟล์อีกครั้งด้วยโปรแกรม gzip ซึ่งจะทำให้ไฟล์ piggy.gz มีขนาดเล็กกว่าไฟล์ zImage อีกเล็กน้อย แต่การทำจะนำไฟล์ piggy.gz นี้ไปใช้บนระบบคอมพิวเตอร์แบบฝังตัวใดๆ ได้นั้น บนระบบคอมพิวเตอร์แบบฝังตัวดังกล่าวจะต้องมี bootloader หรือโปรแกรมที่รองรับการ decompress ไฟล์ gzip ทำงานอยู่ภายในเท่านั้น เนื่องจาก NBC board ของเราใช้ bootloader ที่มีชื่อว่า bios-lt ซึ่งภายใน bootloader ดังกล่าวมีฟังก์ชั่นที่ใช้ในการ decompress ไฟล์ gzip อยู่ด้วย ดังนั้นเราจึงสามารถนำไฟล์ piggy.gz มาใช้บน NBC board ของเราได้ โดยวิธีการที่ใช้กำหนดให้ทำการสร้างไฟล์ piggy.gz ด้วยนั้น สามารถทำได้โดยเติมเครื่องหมาย comment (#) หน้าคำสั่ง rm –f piggy piggy.gz ซึ่งคำสั่งดังกล่าวอยู่ที่ target piggy.o : ในไฟล์ Makefile ภายใน directory /linux-2.4.x/arch/armnommu/boot/compressed/

ขั้นตอนในการคอมไพล์และ build image ของระบบปฏิบัติการ μClinux มีอยู่ด้วยกันหลายขั้นตอน ซึ่งแต่ละขั้นตอนจะขึ้นต่อกันตามลำดับก่อนหลัง ดังนั้นเราจึงต้องทำแต่ละขั้นตอนเรียงลำดับกันไป เพื่อให้ขั้นตอนที่อยู่ในลำดับถัดๆ ไปสามารถทำงานได้อย่างถูกต้อง โดยลำดับและหน้าที่ของแต่ละขั้นตอนจะเป็นดังนี้

1) make clean เป็นคำสั่งที่ใช้ในการลบไฟล์ object และไฟล์ image ต่างๆ ที่เกิดจากการการคอมไพล์และ build image ของระบบปฏิบัติการ μClinux ในครั้งก่อนๆ ซึ่งเราควรจะเรียกใช้คำสั่งนี้ทุกครั้งก่อนที่จะเริ่มกระบวนการคอมไพล์และ build image ของระบบปฏิบัติการ μClinux ขึ้นมาใหม่ โดยเฉพาะถ้าเรามีการเปลี่ยนแปลงการกำหนดค่าต่างๆ ให้กับ kernel และแอพพลิเคชั่นของระบบปฏิบัติการ μClinux (ตามขั้นตอนในหัวข้อ 3.1) เพื่อป้องกันการนำไฟล์ object และไฟล์ image เดิมไปใช้สร้างเป็น image ของระบบปฏิบัติการ μClinux อันใหม่ ซึ่งจะส่งผลให้ image ของระบบปฏิบัติการ μClinux ที่ build ออกมาใหม่นี้ มี kernel และแอพพลิเคชั่นบางอย่างไม่ตรงกับที่เรากำหนดเอาไว้

2) make dep หลังจากที่ทำการกำหนดค่าให้กับ kernel และแอพพลิเคชั่นต่างๆ ของระบบปฏิบัติการ μClinux เรียบร้อยแล้ว ในกรณีที่เลือกใช้ kernel ของระบบปฏิบัติการ μClinux เป็นเวอร์ชัน 2.4 เอาไว้ เราต้องเรียกใช้คำสั่ง make dep ก่อนที่จะเริ่มทำการคอมไพล์และ build

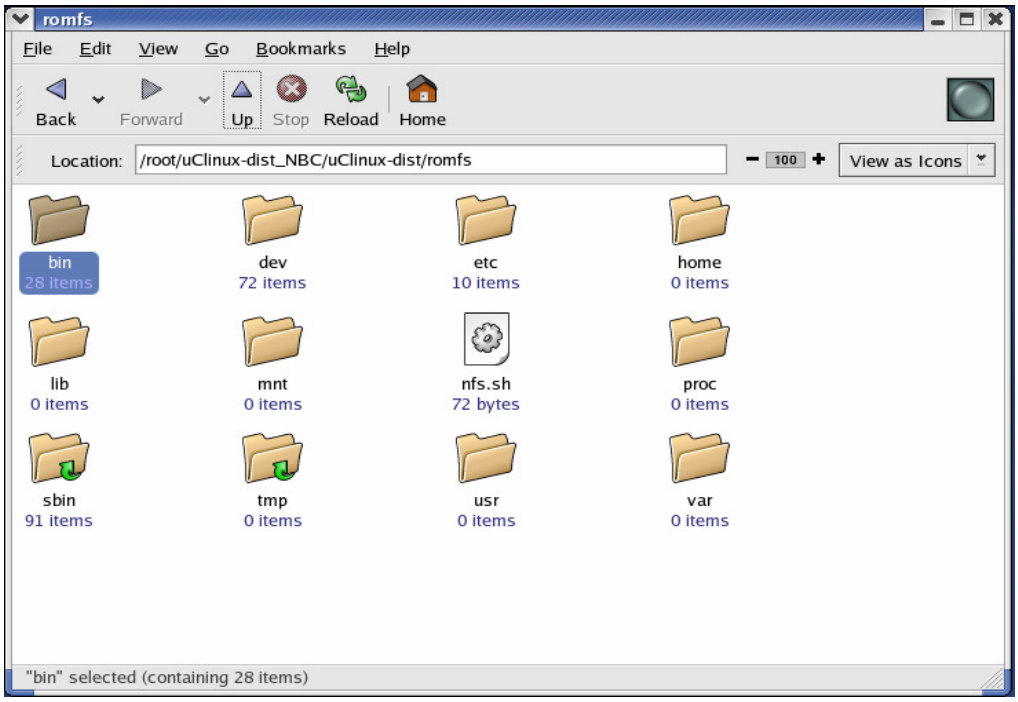
image ของระบบปฏิบัติการ μClinux เสมอ เพื่อตรวจสอบ dependencies ทั้งหมดภายใน μClinux-dist เช่นตรวจสอบไฟล์ source code และ ไฟล์ header ต่างๆ ที่จะต้องใช้ในการคอมไพล์และ build ร่วมกัน (ขึ้นต่อกัน) ว่ามีอยู่ครบถ้วนและอยู่ในตำแหน่งที่ถูกต้องสำหรับการเรียกใช้งานหรือไม่ เป็นต้น

3) make lib_only หลังจากทำคำสั่ง make clean และ make dep เรียบร้อยแล้วก็จะมาถึงขั้นตอนของการคอมไพล์และ build image ของระบบปฏิบัติการ μClinux ขึ้นมาให้เป็นตามที่เราได้ทำการกำหนดค่าเอาไว้ให้กับ kernel และแอพพลิเคชั่นต่างๆ ในขั้นตอนก่อนหน้านี้ โดยคำสั่งแรกที่จะต้องทำก็คือ make lib_only ซึ่งเป็นคำสั่งที่ใช้ในการคอมไพล์ source code ของ API ต่างๆ ทั้งหมดและนำ API เหล่านั้นมารวมกันเพื่อสร้างเป็น library ทั้งในส่วนของ uClibc และ library เสริมอื่นๆ ที่จำเป็นต้องใช้งาน เพื่อจะได้นำ library เหล่านี้ไปใช้ในการคอมไพล์และ build kernel ของระบบปฏิบัติการ μClinux รวมถึงแอพพลิเคชั่นต่างๆ ที่จะนำไปใช้งานบนระบบปฏิบัติการ μClinux ในขั้นตอนต่อๆ ไป

4) make user_only เมื่อ library ต่างๆ พร้อมใช้งานแล้ว ขั้นตอนต่อไปจะใช้คำสั่ง make user_only นี้เพื่อคอมไพล์และ build แอพพลิเคชั่นทั้งหมดตามที่เราได้กำหนดเอาไว้ในหัวข้อ 3.1.2

5) make romfs เป็นการสร้าง directory ทั้งหมดที่จะนำไปใช้งานจริงบนระบบปฏิบัติการ μClinux พร้อมทั้งนำแอพพลิเคชั่นต่างๆ ที่ได้รับการคอมไพล์และ build เอาไว้เรียบร้อยแล้วจากขั้นตอนก่อนนี้ไปติดตั้งลงใน directory ที่ถูกต้อง เพื่อให้ directory ต่างๆ ที่สร้างขึ้นในขั้นตอนนี้พร้อมสำหรับการนำไปสร้างเป็น image ของ romfs ในขั้นตอนต่อไป โดย directory ที่ได้รับการสร้างขึ้นในขั้นตอนนี้จะถูกนำไปเก็บไว้ภายใน directory /romfs ของ μClinux-dist ทั้งหมด ดังนั้นหลังจากที่คำสั่ง make romfs ทำงานเสร็จเรียบร้อยแล้วเราจึงสามารถเข้าไปตรวจสอบความถูกต้องของโครงสร้างและไฟล์ทั้งหมดที่จะนำไปใช้ในการสร้างเป็น image ของ romfs ในขั้นตอนต่อไปได้จาก directory ดังกล่าว สำหรับขั้นตอนการทำงานของคำสั่ง make romfs นั้นเราสามารถตรวจสอบได้จาก target romfs : ของไฟล์ Makefile ที่อยู่ใน directory /vendors/Samsung/4510B (เข้าไปดู Makefile ใน directory ของ vendor ตามที่เราได้เลือกเอาไว้ในตัวเลือก Vendor/Product Selection ก่อนหน้านี้) ซึ่งเราสามารถปรับเปลี่ยนและเพิ่มเติมขั้นตอนการทำงานของ make romfs ได้ตามความต้องการ โดยแก้ไขหรือเพิ่มเติม script ของการทำงานที่เราต้องการลงไปใน target romfs : ของ Makefile ดังกล่าวนั่นเอง สำหรับ directory ต่างๆ ของ romfs ที่ได้รับการสร้างขึ้นใน

ขั้นตอนนี้จะมีโครงสร้างและลักษณะใกล้เคียงกับ directory ที่มีให้ใช้งานอยู่บนระบบปฏิบัติการ Linux ดังรูปที่ 19




**ภาพผนวกที่ ก19** แสดงโครงสร้าง directory ของ romfs สำหรับ NBC board

6) make image เป็นคำสั่งที่ใช้ในการสร้างไฟล์ image ต่างๆ ของระบบปฏิบัติการ µClinux ขึ้นมาทั้ง image.bin ที่เป็นไฟล์ image ของระบบปฏิบัติการ µClinux สำหรับนำไปติดตั้งลงบนระบบคอมพิวเตอร์แบบฝังตัวและไฟล์ romfs.img ที่เป็นไฟล์ image ของ romfs โดยการเรียกใช้คำสั่ง make image ในครั้งนี้จะสามารถสร้างได้แค่ไฟล์ romfs.img เท่านั้นเนื่องจากมีการสร้าง directory ต่างๆ ของ romfs เอาไว้แล้วจากการทำงานของคำสั่ง make romfs ก่อนหน้านี้ ในขณะที่ image.bin นั้นจะยังไม่สามารถสร้างขึ้นมาได้ในขั้นตอนนี้เนื่องจากยังไม่มีการคอมไพล์องค์ประกอบต่างๆ ของ kernel ของระบบปฏิบัติการ µClinux เอาไว้ โดยไฟล์ image ทั้งหมดที่สร้างขึ้นมาโดยคำสั่ง make image นี้จะถูกนำไปเก็บไว้ใน directory /images ภายใน µClinux-dist สำหรับขั้นตอนการทำงานของ make image นั้นเราสามารถตรวจสอบได้จาก target image : ของไฟล์ Makefile ใน directory /vendors/Samsung/4510B (เข้าไปดู Makefile ใน directory ของ vendor ตามที่เราได้เลือกเอาไว้ในตัวเลือก (Vendor/Product Selection ก่อนหน้านี้) ซึ่งเราสามารถ

ปรับเปลี่ยนและเพิ่มเติมขั้นตอนการทำงานของ make image ได้ตามความต้องการ โดยแก้ไขหรือเพิ่มเติม script ของการทำงานที่เราต้องการลงไปใน target image : ของ Makefile ดังกล่าวนั่นเอง

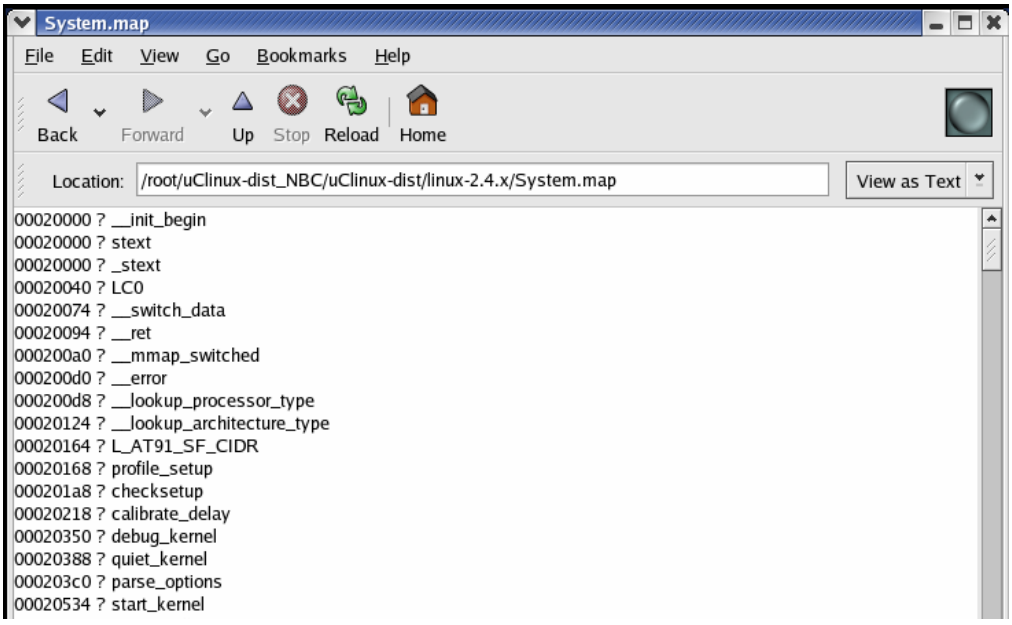
7) make เป็นการคอมไพล์องค์ประกอบต่างๆ ของ kernel ของระบบปฏิบัติการ μClinux ตามที่ได้กำหนดค่าเอาไว้ในหัวข้อ 3.1.1 ก่อนหน้านี้ เมื่อคอมไพล์องค์ประกอบต่างๆ ของ kernel เรียบร้อยแล้ว คำสั่ง make จะเริ่มต้นเรียกคำสั่ง make lib_only, make user_only, make romfs, make image ตามที่ทำไปแล้วในขั้นตอนก่อนหน้านี้ใหม่อีกครั้ง ซึ่งในขั้นตอนของการ make image ในครั้งนี้จะมีการนำองค์ประกอบของต่างๆ ของ kernel ที่ได้คอมไพล์ไว้ก่อนหน้านี้มา link รวมกันเพื่อสร้างเป็นไฟล์ linux.data และ linux.text ขึ้นมาเก็บไว้ใน directory /images หลังจากสร้างไฟล์ทั้งสองดังกล่าวเรียบร้อยแล้วคำสั่ง make image จะเรียกใช้ script ที่นำเอาไฟล์ linux.data, linux.text, และ romfs.img ที่อยู่ภายใน directory /images มา link รวมเข้าด้วยกันเพื่อสร้างเป็นไฟล์ image.bin ขึ้นมาเก็บไว้ใน directory /images โดยไฟล์ image.bin ดังกล่าวก็คือไฟล์ image ของระบบปฏิบัติการ μClinux ที่พร้อมจะนำไปติดตั้งลงบนระบบคอมพิวเตอร์แบบฝังตัวต่อไป

8) make zImage เนื่องจากไฟล์ image.bin ที่ build ขึ้นมาได้อาจมีขนาดใหญ่เกินกว่าจะนำไปเก็บไว้ใน flash ROM ของระบบคอมพิวเตอร์แบบฝังตัว ดังนั้นจึงต้องใช้คำสั่ง make zImage (เรียกใช้คำสั่งนี้ได้ก็ต่อเมื่อ command prompt อยู่ที่ directory /linux-2.4.x ภายใน μClinux-dist เท่านั้น) เพื่อสร้างไฟล์ zImage ขึ้นมา โดยไฟล์ดังกล่าวเป็น image ของระบบปฏิบัติการ μClinux ที่เกิดจากการ compress องค์ประกอบต่างๆ ของระบบปฏิบัติการ μClinux มารวมเข้าด้วยกัน ซึ่งไฟล์ zImage ที่ได้ในขั้นตอนนี้สามารถนำไปใช้งานบนระบบคอมพิวเตอร์แบบฝังตัวได้ทันทีเช่นเดียวกับไฟล์ image.bin เพียงแต่ไฟล์ zImage มีขนาดเล็กกว่าไฟล์ image.bin มาก (ในที่นี้จะได้ zImage มีขนาด 884.7 KB ในขณะที่ image.bin มีขนาด 2.6 MB) สำหรับไฟล์ zImage ที่ build ขึ้นมาได้ในขั้นตอนนี้จะเก็บอยู่ภายใน directory /linux-2.4.x/arch/armnommu/boot นอกจากไฟล์ zImage แล้วคำสั่ง make zImage ยังได้สร้างไฟล์ชื่อ piggy.gz เก็บอยู่ใน directory /linux-2.4.x/arch/armnommu/boot/compressed/ อีกด้วย ซึ่งไฟล์ดังกล่าวก็คือ zImage ที่ถูกนำไป compress ทั้งไฟล์อีกครั้งด้วยโปรแกรม gzip นั่นเอง จึงทำให้ไฟล์ piggy.gz มีขนาดเล็กกว่าไฟล์ zImage อีกเล็กน้อย และจะเหมาะกับการนำไปติดตั้งลงในระบบคอมพิวเตอร์แบบฝังตัวที่มี bootloader หรือโปรแกรมที่รองรับการ decompress ไฟล์ gzip ทำงานอยู่ภายใน

หลังจากที่เราทำตามขั้นตอนต่างๆ ข้างต้นได้เรียบร้อยทั้งหมดโดยไม่มีข้อผิดพลาดเกิดขึ้นในขั้นตอนใดขั้นตอนหนึ่ง เราก็จะได้ image ของระบบปฏิบัติการ μClinux ที่ภายในมีทั้ง kernel และแอพพลิเคชั่นต่างๆ ตรงตามที่เราได้กำหนดค่าเอาไว้ในหัวข้อ 3.1.1 และ 3.1.2 แต่เพื่อความแน่ใจเราสามารถตรวจสอบองค์ประกอบต่างๆ ภายใน image ของระบบปฏิบัติการ μClinux ที่เรา build ขึ้นมาเรียบร้อยแล้วได้อีกครั้งหนึ่งดังนี้

1) ตรวจสอบองค์ประกอบต่างๆ ของ kernel ที่อยู่ภายใน image ของระบบปฏิบัติการ μClinux

เราสามารถตรวจสอบองค์ประกอบต่างๆ ทั้งหมดของ kernel ที่อยู่ภายใน image ของระบบปฏิบัติการ μClinux ได้จากข้อมูลในไฟล์ System.map ที่อยู่ภายใน directory /linux-2.4.x (ตามเวอร์ชันของ kernel ของระบบปฏิบัติการ μClinux ที่เราเลือกใช้) โดยไฟล์ดังกล่าวจะได้รับการสร้างขึ้นโดยอัตโนมัติในระหว่างขั้นตอนของการ link องค์ประกอบต่างๆ ของ kernel ที่คอมไพล์สำเร็จแล้วเข้าด้วยกัน ข้อมูลที่อยู่ในไฟล์ System.map จะเป็นข้อมูลที่แสดง address ของหน่วยความจำ (RAM) และองค์ประกอบของ kernel (อาจจะเป็น symbol หรือ system call ต่างๆ) ที่ประจำ address นั้นๆ ดังแสดงในรูปที่ 20




**ภาพผนวกที่ ก20** แสดงข้อมูลในไฟล์ System.map ของ NBC board

จากรูปที่ 20 พบว่า address เริ่มต้นของ kernel ซึ่งอยู่ภายใน image ของระบบปฏิบัติการ μClinux ที่เรา build ออกมาจะอยู่ที่ address 0x00020000 ซึ่งเป็นไปตามค่าที่เก็บอยู่ในตัวแปร TEXTADDR ดังที่ได้กล่าวถึงก่อนหน้านี้ ดังนั้นเราต้องนำเอาเลขที่ address ดังกล่าวไปกำหนดให้กับ bootloader ภายใน NBC board เพื่อให้ bootloader ทำการ copy kernel ที่อยู่ภายใน image ของระบบปฏิบัติการ μClinux ซึ่งเก็บอยู่ในหน่วยความจำ flash ROM ไปเริ่มเขียนที่ address ดังกล่าว (0x00020000) ของหน่วยความจำ RAM เพื่อให้องค์ประกอบต่างๆ ภายใน kernel สามารถ link ถึงกันได้อย่างถูกต้องตาม address ที่กำหนดเอาไว้ (ดูได้จากไฟล์ System.map) ซึ่งจะส่งผลให้ kernel ของระบบปฏิบัติการ μClinux ที่อยู่ภายใน image ของเรา สามารถทำงานบน NBC board ได้อย่างถูกต้อง

ในกรณีที่ไม่ปรากฏองค์ประกอบของ kernel ที่เราต้องการใช้งานอยู่ภายในไฟล์ System.map ก็แสดงว่า kernel ที่อยู่ภายใน image ของระบบปฏิบัติการ μClinux ที่เราทำการ build ขึ้นมาเพื่อใช้งานในขณะนี้ไม่มีองค์ประกอบดังกล่าวด้วย ดังนั้นเราจึงจำเป็นต้องเริ่มทำการกำหนดค่าต่างๆ ของ kernel ของระบบปฏิบัติการ μClinux ใหม่อีกครั้ง (ตามหัวข้อ 3.1.1) เพิ่มเพิ่มองค์ประกอบของ kernel ที่เรายังไม่มีและจำเป็นสำหรับการทำงานลงไป หลังจากนั้นจึงเริ่มกระบวนการคอมไพล์และ build image ของระบบปฏิบัติการ μClinux ใหม่อีกครั้ง เพื่อให้ได้ image ของระบบปฏิบัติการ μClinux ที่ kernel ภายในมีองค์ประกอบต่างๆ ซึ่งจำเป็นสำหรับการทำงานอย่างครบถ้วน จากลักษณะดังกล่าวไฟล์ System.map จึงเปรียบเสมือนแผนที่ขององค์ประกอบต่างๆ ที่อยู่ภายใน kernel ของระบบปฏิบัติการ μClinux ในขณะนั้นนั่นเอง

2) ตรวจสอบแอพพลิเคชั่นทั้งหมดที่อยู่ภายใน image ของระบบปฏิบัติการ μClinux

เราสามารถอาศัยโครงสร้างภายใน directory /romfs ที่เกิดจากการทำงานของคำสั่ง make romfs มาใช้ในการตรวจสอบแอพพลิเคชั่นทั้งหมดที่มีอยู่ภายใน image ของระบบปฏิบัติการ μClinux ได้ เนื่องจาก directory /romfs ภายใน μClinux-dist เป็น directory ที่ใช้เก็บผลลัพธ์จากการทำงานทั้งหมดของคำสั่ง make romfs ดังนั้นสิ่งที่อยู่ภายใน directory ดังกล่าวจะมีทั้ง directory ต่างๆ ที่คำสั่ง make romfs สร้างขึ้นและไฟล์ของแอพพลิเคชั่นทั้งหมดตามที่เรากำหนดค่าเอาไว้ในหัวข้อ 3.1.2 อยู่ภายใน directory เหล่านั้นด้วย ซึ่งสิ่งต่างๆ ดังกล่าวภายใน directory /romfs จะถูกนำไปใช้สร้างเป็น image ของ romfs (romfs.img) สำหรับ link เข้ากับ image ของระบบปฏิบัติการ

μClinux ต่อไป ทำให้แอพพลิเคชั่นใดที่มีอยู่ภายใน directory /romfs ก็จะต้องมีอยู่ภายใน image ของระบบปฏิบัติการ μClinux ด้วยเสมอ

ถ้าตรวจสอบใน directory /romfs แล้วไม่พบแอพพลิเคชั่นที่เราต้องการใช้งาน ก็แสดงว่าภายใน image ของระบบปฏิบัติการ μClinux ก็จะไม่มีแอพพลิเคชั่นนั้นๆ อยู่ด้วย ดังนั้นเราจะต้องเริ่มกำหนดค่าของแอพพลิเคชั่นต่างๆ ที่เราต้องการใช้งานบนระบบปฏิบัติการ μClinux ใหม่อีกครั้งหนึ่ง (ตามหัวข้อ 3.1.2) เพื่อเพิ่มแอพพลิเคชั่นที่เรายังไม่มีและจำเป็นต้องใช้งานลงไป หลังจากนั้นจึงเริ่มกระบวนการคอมไพล์และ build image ของระบบปฏิบัติการ μClinux ใหม่อีกครั้ง เพื่อให้ได้ image ของระบบปฏิบัติการ μClinux ที่มีแอพพลิเคชั่นซึ่งเราต้องการใช้งานอย่างครบถ้วน

ถ้าทั้งองค์ประกอบของ kernel และแอพพลิเคชั่นต่างๆ ภายใน image ของระบบปฏิบัติการ μClinux ที่เรา build ออกมาได้ครบถ้วนตามที่เราต้องการใช้งานทั้งหมด เราก็สามารถนำ image ดังกล่าวไปติดตั้งลงบน NBC board ของเราได้ทันที หัวข้อต่อไปจะกล่าวถึงวิธีการนำเอา image ของระบบปฏิบัติการ μClinux ไปติดตั้งใช้งานจริงบน NBC board

**4 ติดตั้ง image ของระบบปฏิบัติการ μClinux ที่ build ขึ้นมาเรียบร้อยแล้วไปติดตั้งใช้งานบน NBC board**

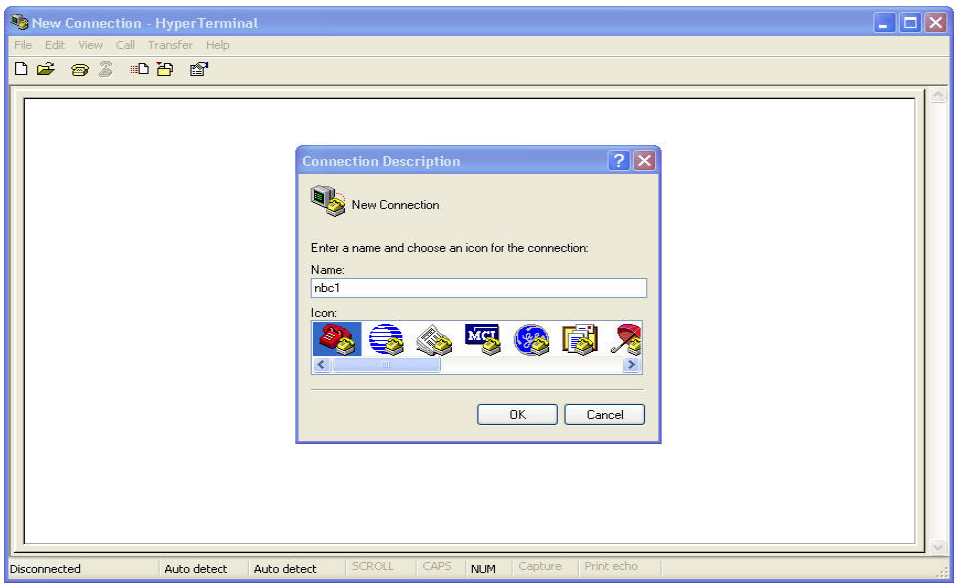
โดยปกติบริษัทผู้ผลิตระบบคอมพิวเตอร์แบบฝังตัวส่วนใหญ่จะจัดเตรียมซอฟต์แวร์ที่ใช้สำหรับโหลดแอพพลิเคชั่นจากภายนอกไปติดตั้งลงบนหน่วยความจำของระบบคอมพิวเตอร์แบบฝังตัวนั้นๆ ให้อยู่แล้ว ซึ่งซอฟต์แวร์ดังกล่าวจะได้รับการติดตั้งอยู่ใน flash ROM ของระบบคอมพิวเตอร์แบบฝังตัวและทำหน้าที่ในการนำแอพพลิเคชั่นที่ผู้ใช้งานต้องการใช้งานไปเขียนเก็บลงยัง flash ROM ในตำแหน่งที่ต้องการ แต่ซอฟต์แวร์เหล่านี้มีฟังก์ชั่นการใช้งานน้อยมาก ทำให้ไม่เหมาะสมสำหรับนำมาใช้ในการติดตั้งระบบปฏิบัติการลงไปใช้งานบนระบบคอมพิวเตอร์แบบฝังตัว ดังนั้นผู้ใช้งานส่วนใหญ่จึงหันไปใช้ bootloader ในการทำหน้าที่ดังกล่าวแทน

bootloader จะได้รับการติดตั้งอยู่ภายใน flash ROM ของระบบปฏิบัติการ โดย bootloader ที่มีใช้งานในปัจจุบันสำหรับระบบคอมพิวเตอร์แบบฝังตัวมีมากมายหลายรูปแบบ ซึ่งผู้ใช้งานจะต้องพิจารณาเลือกใช้ bootloader ให้เหมาะสมกับระบบคอมพิวเตอร์แบบฝังตัวที่ตนเองมีอยู่ เพื่อให้ bootloader สามารถทำงานร่วมกับระบบคอมพิวเตอร์แบบฝังตัวดังกล่าวได้อย่างมี

ประสิทธิภาพ สำหรับ bootloader ที่นำมาใช้กับ NBC board คือ bios-lt เนื่องจาก bootloader ดังกล่าวเป็น bootloader ที่ได้รับการออกแบบมาให้ใช้งานกับระบบคอมพิวเตอร์แบบฝังตัวที่ใช้ microcontroller Samsung S3C4510B โดยเฉพาะ ทำให้สามารถนำ bootloader ดังกล่าวมาประยุกต์ใช้กับ microcontroller Samsung S3C4530A ที่มีอยู่บน NBC board ของเราได้อย่างไม่ยากนัก เนื่องจาก microcontroller ทั้งสองมีลักษณะการทำงานและองค์ประกอบภายในใกล้เคียงกัน นอกจากนี้ bios-lt ยังได้รับการพัฒนาในลักษณะของ open source ทำให้เราสามารถนำ source code ของ bootloader ดังกล่าวมาปรับแก้ได้ตามความต้องการและในปัจจุบัน bios-lt ได้รับการพัฒนาให้มีฟังก์ชั่นการทำงานมากมายและสามารถทำงานร่วมกับระบบปฏิบัติการ μClinux ได้เป็นอย่างดี สำหรับวิธีการแก้ไข bios-lt ให้เหมาะสมกับ microcontroller Samsung S3C4530A และวิธีการติดตั้ง bios ดังกล่าวไปใช้งานบน NBC board จะไม่กล่าวถึงในที่นี้ แต่จะกล่าวถึงเฉพาะการใช้ bios-lt ในการติดตั้งระบบปฏิบัติการ μClinux ไปติดตั้งลงบน NBC board เท่านั้น (เสมือนมี bios-lt ติดตั้งอยู่ใน flash ROM ของ NBC board มาให้อยู่แล้ว)

ก่อนที่จะเริ่มติดตั้งระบบปฏิบัติการ μClinux ลงไปใช้งานบน NBC board ได้นั้น ต้องทำการ set up โปรแกรม hyperterminal บนเครื่อง PC เพื่อใช้ติดต่อกับ NBC board โดยก่อนจะเริ่ม set up โปรแกรมดังกล่าวจะต้องนำเอา NBC board มาต่อกับเครื่อง PC โดยใช้สาย RS232 เสียบเข้ากับ serial port ของทั้งสองฝั่ง และจะต้องนำสาย RJ45 มาต่อเข้ากับ interface Ethernet ของ NBC board ด้วยเพื่อใช้ในขั้นตอนของการติดตั้งระบบปฏิบัติการ μClinux ลงบน NBC board ต่อไป สำหรับวิธีการ set up โปรแกรม hyperterminal เพื่อใช้ติดต่อกับ NBC board มีขั้นตอนดังนี้

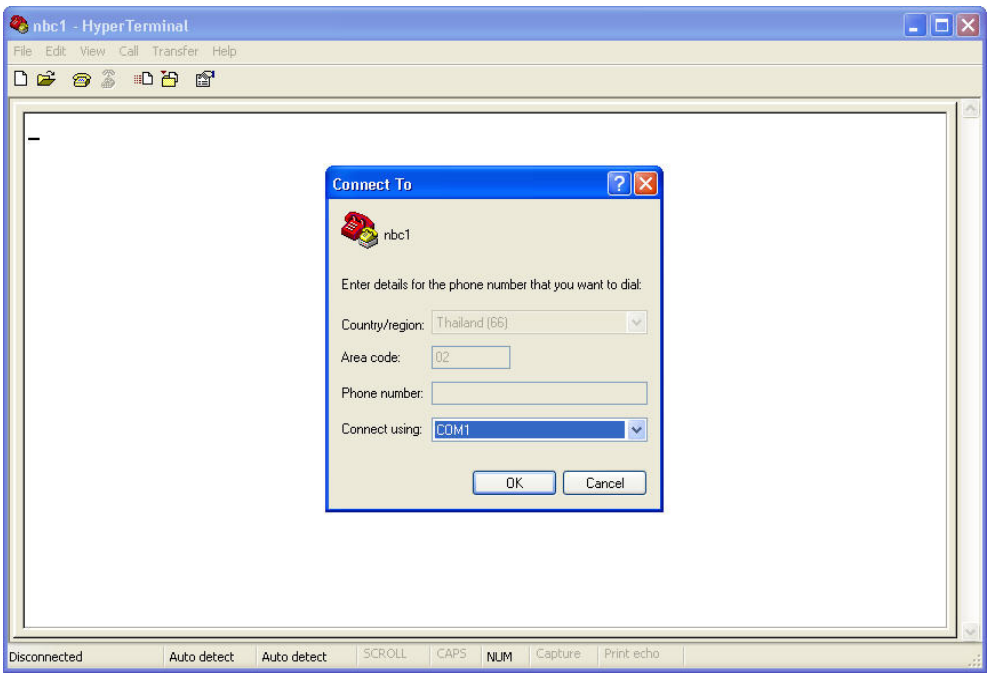
1) เปิดโปรแกรม hyperterminal ที่มีติดตั้งอยู่บนระบบปฏิบัติการ windows ขึ้นมา ซึ่งโปรแกรมดังกล่าวจะมีลักษณะเป็นดังรูปที่ 21




**ภาพผนวกที่ ก21** แสดงหน้าจอของโปรแกรม hyperterminal

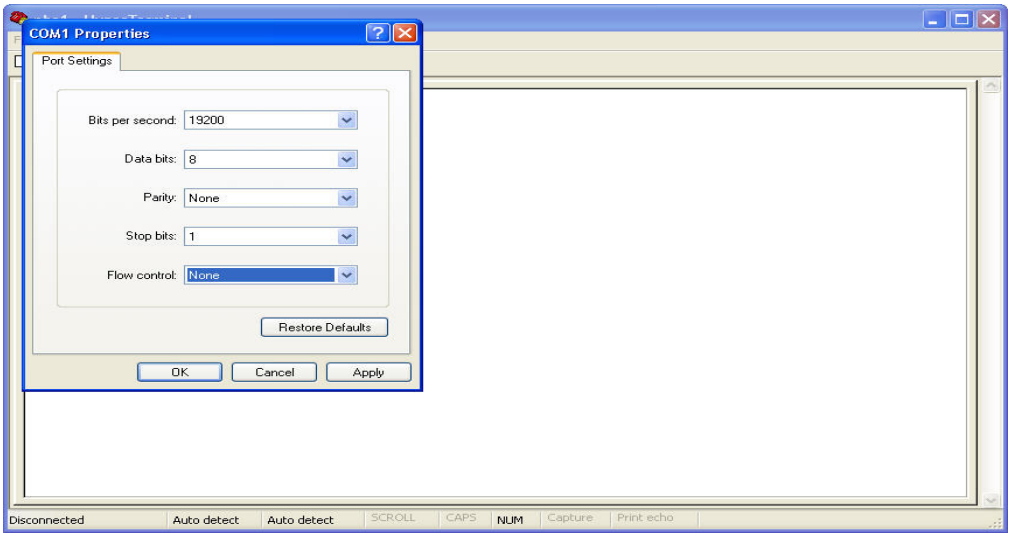
เมื่อหน้าจอของโปรแกรม hyperterminal แสดงขึ้นมาจะปรากฏ dialog box ให้กำหนดชื่อและรูปของ การเชื่อมต่อที่เราสร้างขึ้น ซึ่งเราสามารถกำหนดชื่อและเลือกรูปที่เราต้องการได้อย่างอิสระ

2) กำหนดชนิดของ port บนเครื่อง PC ที่ใช้สำหรับการเชื่อมต่อกับ NBC board ซึ่งในที่นี้เราทำการเชื่อมต่อ PC กับ NBC board ผ่านทาง serial port ดังนั้นจึงต้องกำหนดค่าของตัวเลือก Connect using: ให้เป็น COM1 ตามเลขที่ของ serial port บนเครื่อง PC ที่เราเลือกใช้ ดังรูปที่ 22



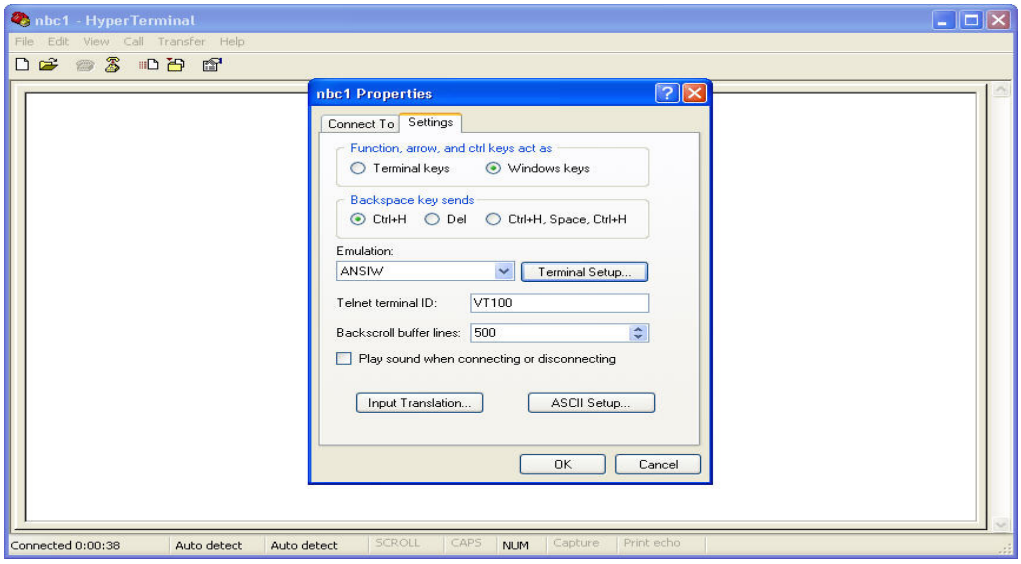

**ภาพผนวกที่ ก22** กำหนดชนิดของ port ที่เครื่อง PC ใช้สำหรับการเชื่อมกับ NBC board

3) กำหนดคุณสมบัติต่างๆ ของ port ที่เครื่อง PC เลือกใช้ในการเชื่อมต่อนี้ ซึ่งในที่นี้ก็คือการกำหนดคุณสมบัติให้กับ port COM1 บนเครื่อง PC ที่เราเลือกเอาไว้ในเมนูก่อนหน้านี้เพื่อใช้ในการเชื่อมต่อกับ NBC board นั่นเอง โดยเราจะต้องกำหนดคุณสมบัติต่างๆ ของ port ดังกล่าวให้ถูกต้อง โดยเฉพาะค่า Bits per second ที่เราจะกำหนดให้มีค่าเป็น 192000 ตามค่าของ Initial serial console speed ที่เราได้ทำการกำหนดเอาไว้ให้กับ UARTA ภายในระบบปฏิบัติการ μClinux ก่อนหน้านี้ (หัวข้อ กำหนด driver ให้กับ controller ของ UART สำหรับ serial port) ซึ่งจะทำให้โปรแกรม hyperterminal บนเครื่อง PC สามารถแสดงผลการทำงานของระบบปฏิบัติการ μClinux บน NBC board ผ่านทาง serial port ได้อย่างถูกต้อง สำหรับวิธีการกำหนดคุณสมบัติอื่นๆ ของ port COM1 เป็นดังรูปที่ 23



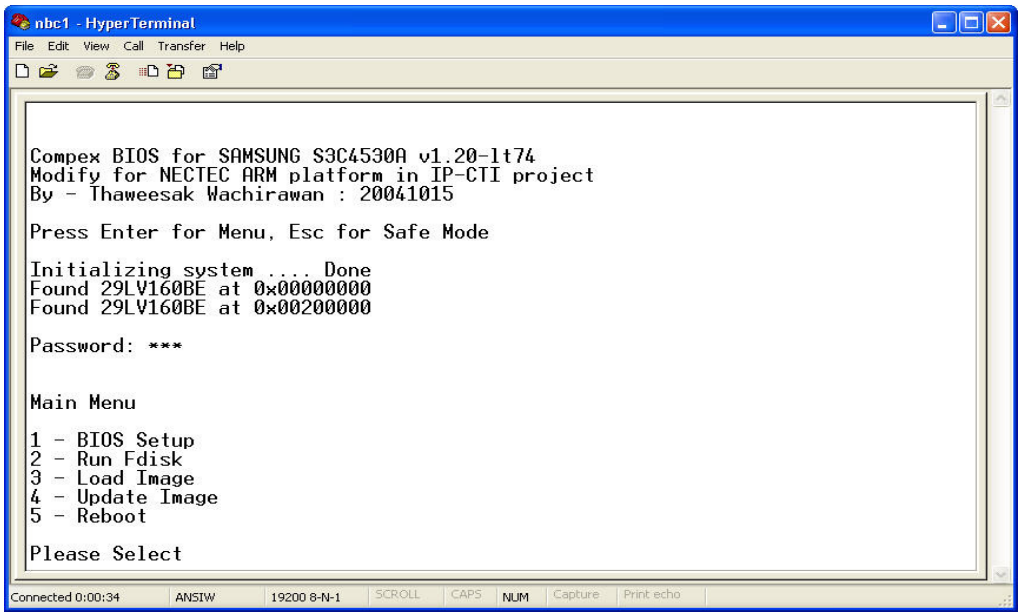

**ภาพผนวกที่ ก23** แสดงการกำหนดคุณสมบัติต่างๆ ของ port COM1 บนเครื่อง PC เพื่อให้สามารถติดต่อกับ NBC board ได้อย่างถูกต้อง

4) กำหนดคุณสมบัติต่างๆ ของการเชื่อมต่อที่สร้างขึ้นระหว่างเครื่อง PC และ NBC board เพื่อให้การเชื่อมต่อและการรับส่งข้อมูลระหว่างเครื่อง PC และ NBC board เป็นไปอย่างถูกต้อง ซึ่งการกำหนดคุณสมบัติต่างๆ ดังกล่าวเป็นดังรูปที่ 24




**ภาพผนวกที่ ก24** แสดงการกำหนดคุณสมบัติต่างๆ ให้กับการเชื่อมต่อที่สร้างขึ้นระหว่างเครื่อง PC และ NBC board

ถ้าทำการ set up โปรแกรม hyperterminal เรียบร้อยแล้ว เมื่อเปิด NBC board ที่มี bios-lt ติดตั้งอยู่ จะต้องได้ผลการทำงานดังรูปที่ 25



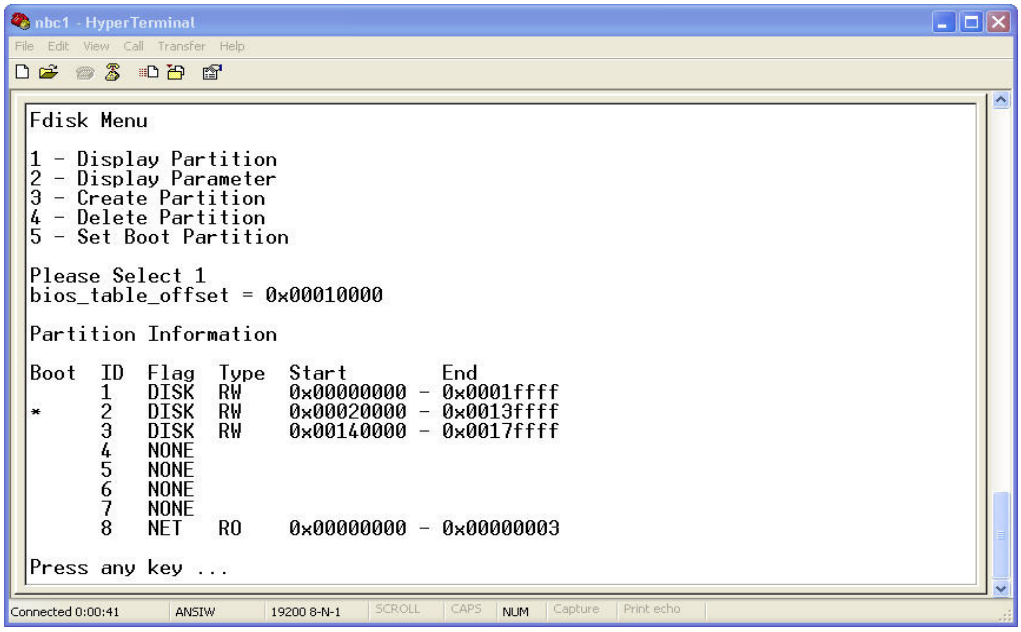

**ภาพผนวกที่ ก25** แสดงผลการทำงานของ bios-lt บน NBC board หลังจากเปิด NBC board ดังกล่าวขึ้นมาทำงาน

จากรูปที่ 25 พบว่าจะมีเมนูของ bios-lt ให้เราเลือกใช้งานอยู่ 5 เมนูด้วยกัน โดยแต่ละเมนูก็จะมีหน้าที่แตกต่างกันออกไป ซึ่งเราสามารถใช้เมนูเหล่านี้ในการตรวจสอบและจัดเตรียมสภาพแวดล้อมต่างๆ บน NBC board ให้พร้อมสำหรับการนำ image ของระบบปฏิบัติการ µClinux ไปติดตั้งได้อย่างถูกต้อง โดยการตรวจสอบและจัดเตรียมสภาพแวดล้อมต่างๆ ของ NBC board ให้พร้อมสำหรับการติดตั้ง image ของระบบปฏิบัติการ µClinux มีดังนี้

1) ตรวจสอบขนาดของ partition ภายใน flash ROM ที่จะใช้จัดเก็บ image ของระบบปฏิบัติการ µClinux

เนื่องจาก bios-lt เป็น bootloader ที่รองรับการ uncompress ไฟล์ gzip ได้ ดังนั้นรูปแบบของระบบปฏิบัติการ µClinux ที่เราจะเลือกใช้บน NBC board จึงสามารถใช้ได้ทั้งแบบ zImage และ piggy.gz แต่เพื่อประหยัดพื้นที่บนหน่วยความจำ flash ROM เราจึงเลือกใช้ไฟล์ piggy.gz เนื่องจากมีขนาดเล็กกว่า zImage อีกเล็กน้อย

ก่อนที่เราจะทำการติดตั้งไฟล์ piggy.gz ลงไปบน flash ROM ของ NBC board เราจำเป็นต้องตรวจสอบพื้นที่ของ partition ภายใน flash ROM ที่ใช้สำหรับเก็บไฟล์ดังกล่าวก่อนว่าเพียงพอหรือไม่ ถ้าพบว่ามีพื้นที่เพียงพอก็สามารถนำไฟล์ piggy.gz ไปเก็บได้ทันที แต่ถ้าไม่เพียงพอเราอาจจะต้องขยายขนาดของ partition ดังกล่าวบน flash ROM ให้มากขึ้นหรืออาจจะทำการกำหนดค่า, คอมไพล์, และ build image ของระบบปฏิบัติการ μClinux ขึ้นมาใหม่เพื่อให้ได้ไฟล์ piggy.gz ที่มีขนาดเล็กเพียงพอกับการนำไปเก็บลงใน partition นั้น สำหรับวิธีการตรวจสอบ partition ต่างๆ ภายใน flash ROM ให้เลือกไปตัวเลือก Run Fdisk ภายใน Main menu ของ bios-lt ซึ่งเมื่อเลือกที่ตัวเลือกดังกล่าวแล้ว ให้เลือก Display Partition เพื่อแสดง partition ทั้งหมดบน flash ROM ดังรูปที่ 26




**ภาพผนวกที่ ก26** แสดง partition ทั้งหมดบน flash ROM

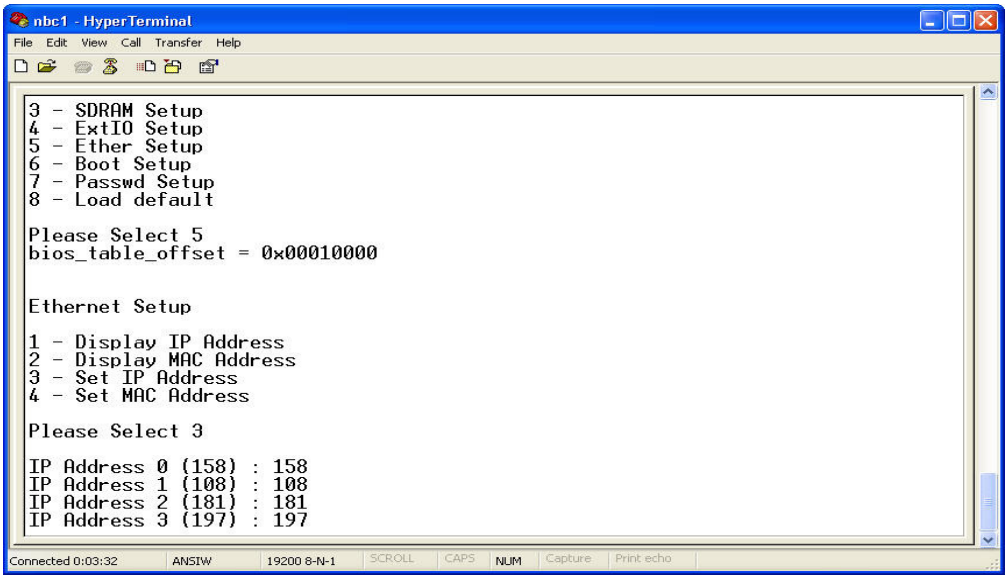
จากรูปที่ 26 partition ที่จะใช้ในการเก็บไฟล์ piggy.gz คือ partition ที่ได้รับการกำหนดให้เป็น partition สำหรับการ boot (มีเครื่องหมาย * อยู่ใน field Boot หน้า partition นั้นๆ) ซึ่งในที่นี้คือ partition ที่ 2 เมื่อเราแล้วว่า partition ใดเป็น partition ที่จะใช้เก็บไฟล์ piggy.gz ก็ให้นำค่า Start Address และ End address ของ partition ดังกล่าวมาคำนวณขนาดของหน่วยความจำสำหรับ

partition นั้นๆ แล้วนำไปเปรียบเทียบกับขนาดของ piggy.gz ที่เรา build ออกมาได้ ว่าสามารถใช้ partition ดังกล่าวสำหรับเก็บไฟล์ piggy.gz ได้หรือไม่

2) กำหนด IP address ชั่วคราวสำหรับ tftp server บน bios-lt

สำหรับกระบวนการที่ bios-lt ใช้ในการโหลดเอาไฟล์ piggy.gz ไปเก็บลงใน flash ROM ตาม partition ที่กำหนดนั้นจะอาศัยโปรแกรม tftp server คอยรอรับไฟล์ดังกล่าวจาก client ซึ่ง bios-lt จะทำการรัน tftp server ขึ้นมาทันทีเมื่อผู้ใช้งานต้องการโหลดเอาไฟล์ piggy.gz ไปเก็บลงใน flash ROM ดังนั้นเราจึงต้องกำหนด IP address ของ tftp server ดังกล่าวเอาไว้ เพื่อจะได้ใช้ในขั้นตอนของการโหลดไฟล์ piggy.gz ในขั้นตอนต่อไป

วิธีการกำหนด IP address ให้กับ tftp server บน bios-lt สามารถทำได้โดยเลือกไปที่ตัวเลือก BIOS Setup ภายใน Main menu ของ bios-lt โดยเมื่อเลือกที่ตัวเลือกดังกล่าวแล้วให้เลือกที่ตัวเลือก Ether Setup แล้วเลือกที่ตัวเลือก Set IP Address เพื่อทำการกำหนด IP address ที่ต้องการ ดังรูปที่ 27

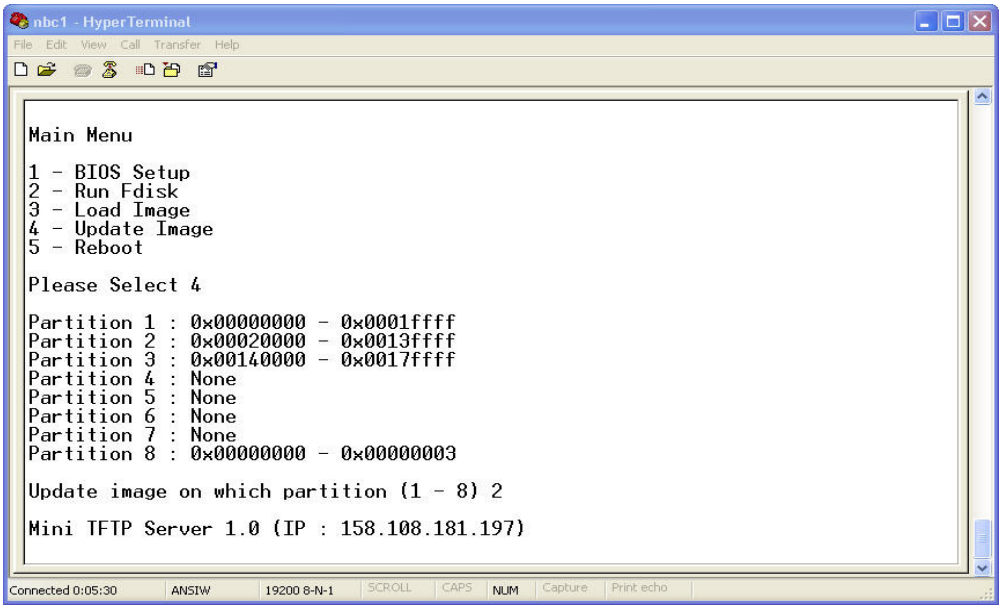



**ภาพผนวกที่ ก27** การกำหนด IP address ของ tftp server ภายใน bios-lt

หลังจากตรวจสอบขนาดของ partition ภายใน flash ROM และกำหนด IP address ของ tftp server ภายใน bios-lt แล้วเราก็สามารถทำการ load เอาไฟล์ piggy.gz ไปเก็บลงใน NBC board ได้ทันที โดยวิธีการติดตั้งไฟล์ piggy.gz และ boot ระบบปฏิบัติการ μClinux ขึ้นมาทำงานบน NBC board เป็นดังนี้

1) load ไฟล์ piggy.gz ไปเก็บลงใน flash ROM ของ NBC board

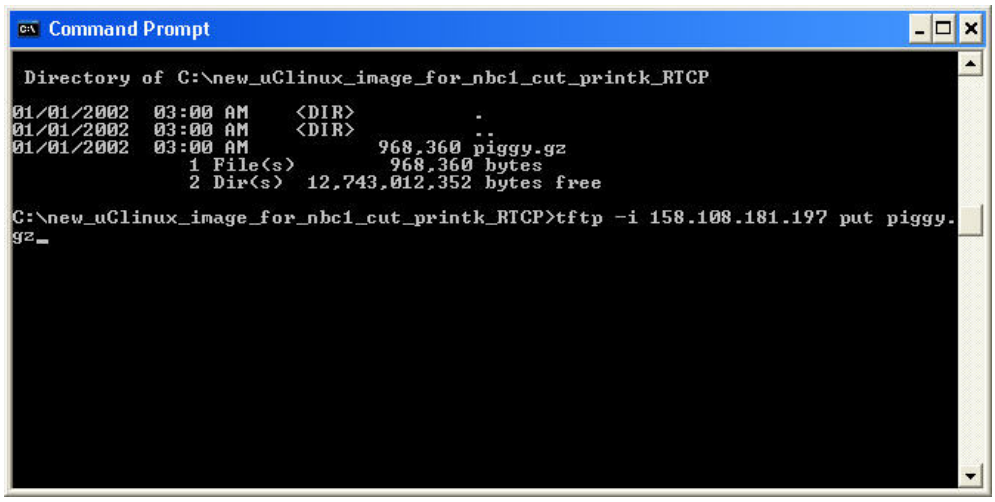
เมื่อเราต้องการ load ไฟล์ piggy.gz ไปเก็บลงใน flash ROM ของ NBC board ให้เราเลือกที่ตัวเลือก Update Image ภายใน Main menu ของ bios-lt หลังจากนั้นให้เราระบุ partition ภายใน flash ROM ที่จะนำไฟล์ดังกล่าวไปเก็บ ซึ่งในที่นี้จะเลือก partition 2 เมื่อกำหนด partition เรียบร้อยแล้ว bios-lt จะรัน tftp server ขึ้นมาทำงานโดยกำหนดให้ IP address ของ server ดังกล่าวมีค่าตามที่เรากำหนดเอาไว้ก่อนหน้านี้ ดังรูปที่ 28




**ภาพผนวกที่ ก28** แสดงการเลือกเมนูและ partition ของ flash ROM ที่ใช้ในการ load ไฟล์ piggy.gz ไปเก็บลงใน NBC board

หลังจากที่ bios-lt รัน tftp server บน NBC board เรียบร้อยแล้ว ผู้ใช้งานจะต้องเรียกใช้ tftp client บนเครื่อง PC ที่เก็บไฟล์ piggy.gz ซึ่งเราต้องการนำไปใช้งานบน NBC board เอาไว้ เพื่อจะ

ได้ load เอาไฟล์ดังกล่าวจากเครื่อง PC ไปเก็บลงใน flash ROM ของ NBC board สำหรับการเรียกใช้ tftp client บนเครื่อง PC เป็นดังรูปที่ 29

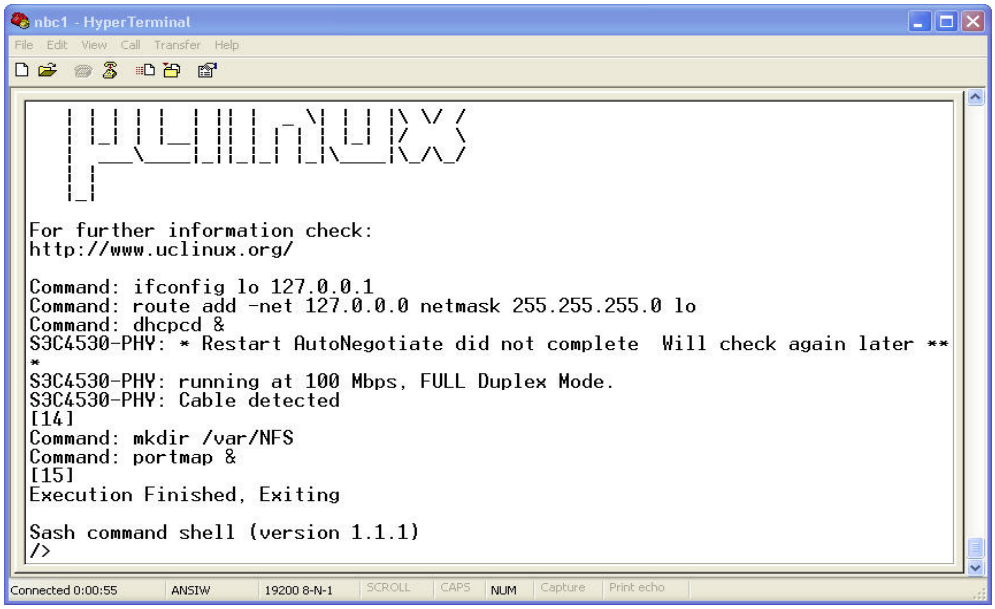



**ภาพผนวกที่ ก29** การเรียกใช้ tftp client บนเครื่อง PC ที่เก็บไฟล์ piggy.gz ที่เราต้องการใช้งานเอาไว้ เพื่อ load เอาไฟล์ดังกล่าวจากเครื่อง PC ไปเก็บลงใน flash ROM ของ NBC board

ถ้าสามารถ load ไฟล์ piggy.gz ไปเก็บไว้ใน partition 2 ของ flash ROM ได้เรียบร้อย หน้าจอของโปรแกรม hyperterminal จะกลับไปที่ Main Menu ของ bios-lt โดยอัตโนมัติ

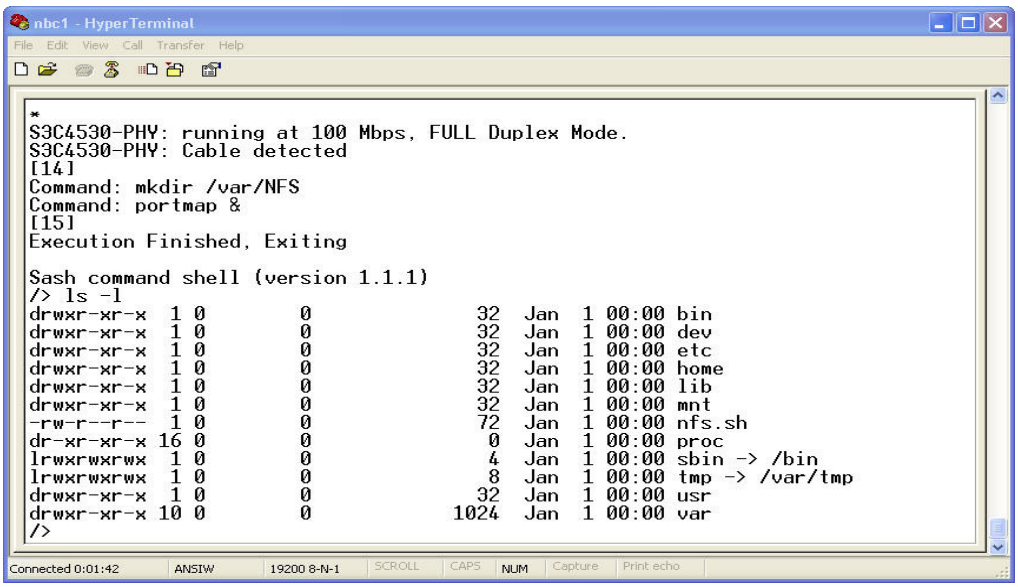
2) boot ระบบปฏิบัติการ μClinux ขึ้นมาทำงานบน NBC board

เมื่อ load ไฟล์ piggy.gz ของระบบปฏิบัติการ μClinux ไปเก็บอยู่ใน flash ROM ของ NBC board เรียบร้อยแล้ว เราสามารถสั่ง boot ระบบปฏิบัติการ μClinux จากไฟล์ดังกล่าวขึ้นมาทำงานบน NBC board ได้ทันที สำหรับการสั่ง boot ระบบปฏิบัติการ μClinux บน NBC board นั้นสามารถทำได้โดยเลือกที่ตัวเลือก Load Image ภายใน Main menu ของ bios-lt ซึ่งจะได้ผลลัพธ์ในการ boot ระบบปฏิบัติการ μClinux ดังรูปที่ 30

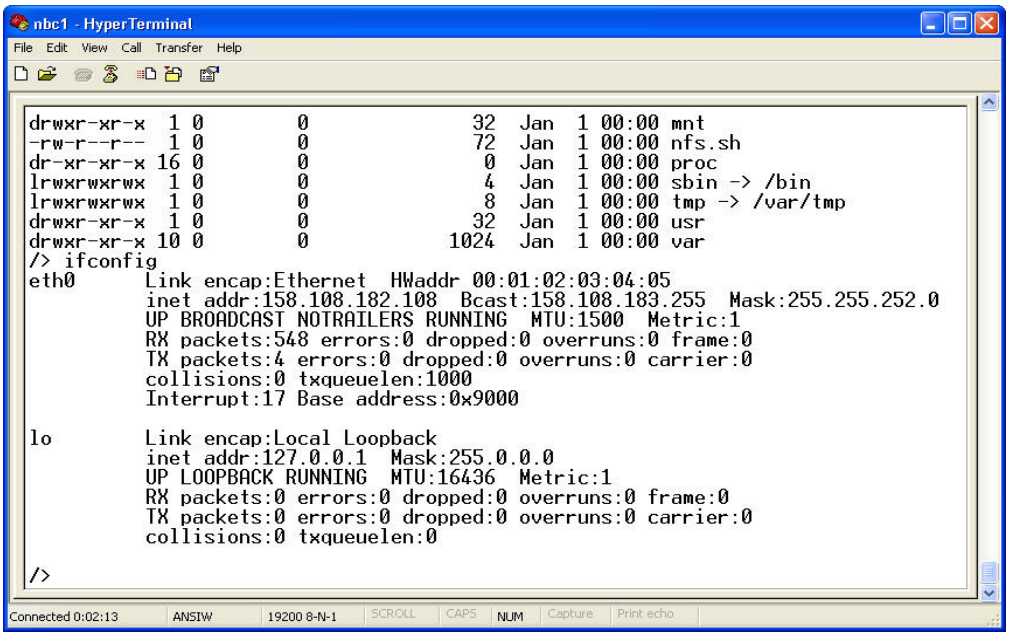



**ภาพผนวกที่ ก30** แสดงผลลัพธ์ของการ boot ระบบปฏิบัติการ μClinux จากไฟล์ piggy.gz ขึ้นมาทำงานบน NBC board

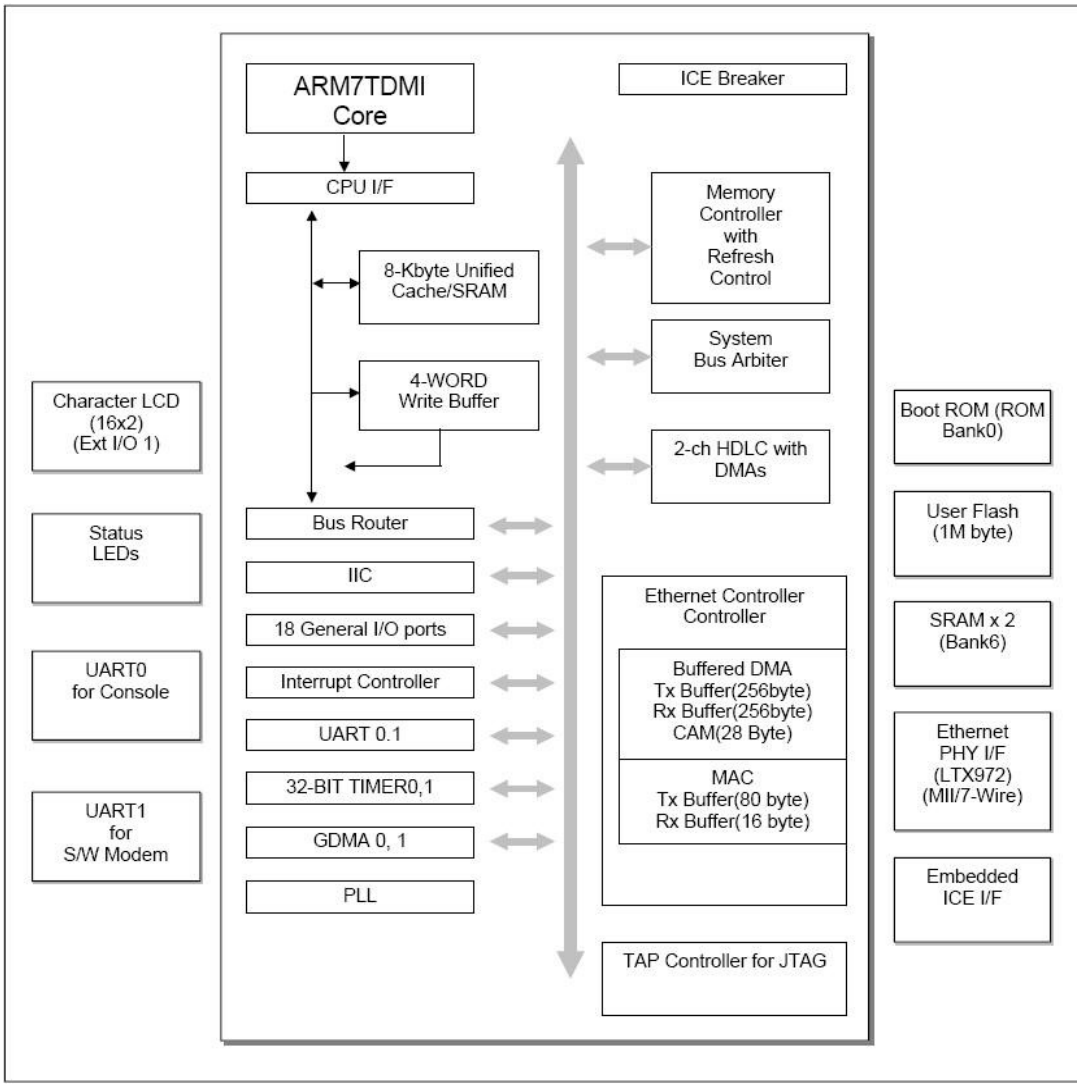
ถ้าในขั้นตอนของการ boot ระบบปฏิบัติการ μClinux ขึ้นมาทำงานบน NBC board ไม่มีข้อผิดพลาดใดๆ เกิดขึ้น และไปหยุดอยู่ที่ command prompt ก็แสดงว่าในขณะนี้ระบบปฏิบัติการ μClinux ได้ทำงานอยู่บน NBC board ของเราอย่างสมบูรณ์แล้ว และพร้อมที่จะรับคำสั่งจากผู้ใช้งานได้ทันที เช่น ls -l สำหรับ list รายชื่อของ directory และไฟล์ต่างๆ ซึ่งจะได้ผลลัพธ์ของการทำงานดังรูปที่ 31, ifconfig สำหรับตรวจสอบค่าต่างๆ ของ interface network ที่มีอยู่บน NBC board ซึ่งจะได้ผลลัพธ์ดังรูปที่ 32 เป็นต้น




**ภาพผนวกที่ ก31** แสดงผลการทำงานของคำสั่ง ls –l บนระบบปฏิบัติการ μClinux ภายใน NBC board




**ภาพผนวกที่ ก32** แสดงผลการทำงานของคำสั่ง ifconfig บนระบบปฏิบัติการ μClinux ภายใน NBC board



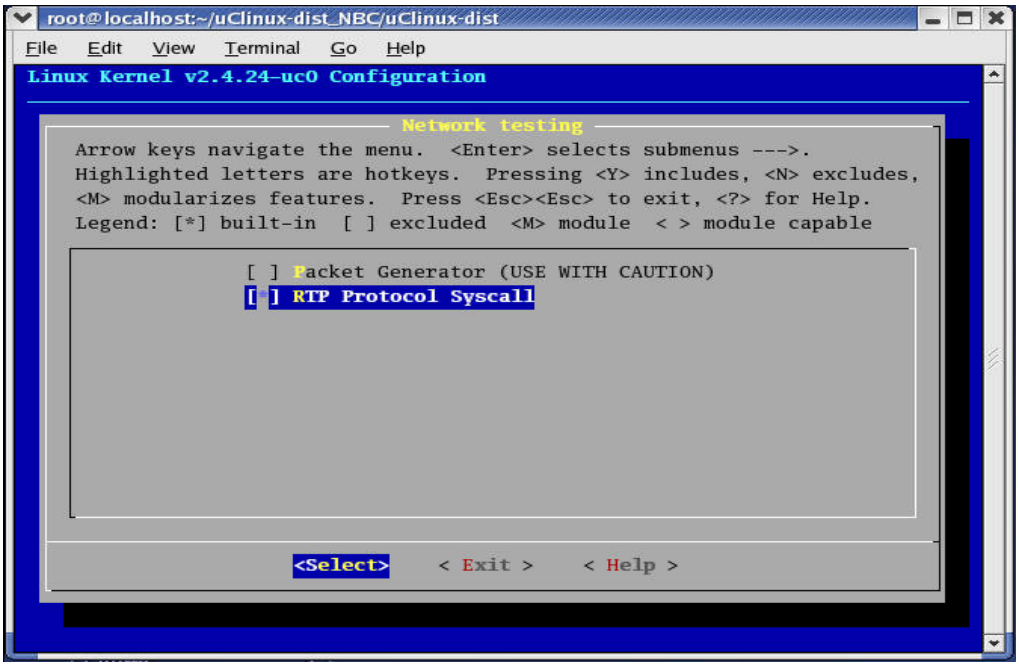

**ภาพผนวกที่ ก33** แสดง diagram การทำงานของ microcontroller Samsung S3C4530A

# ภาคผนวก ข

การติดตั้งและใช้งาน system call ของโปรโตคอล RTP และ RTCP

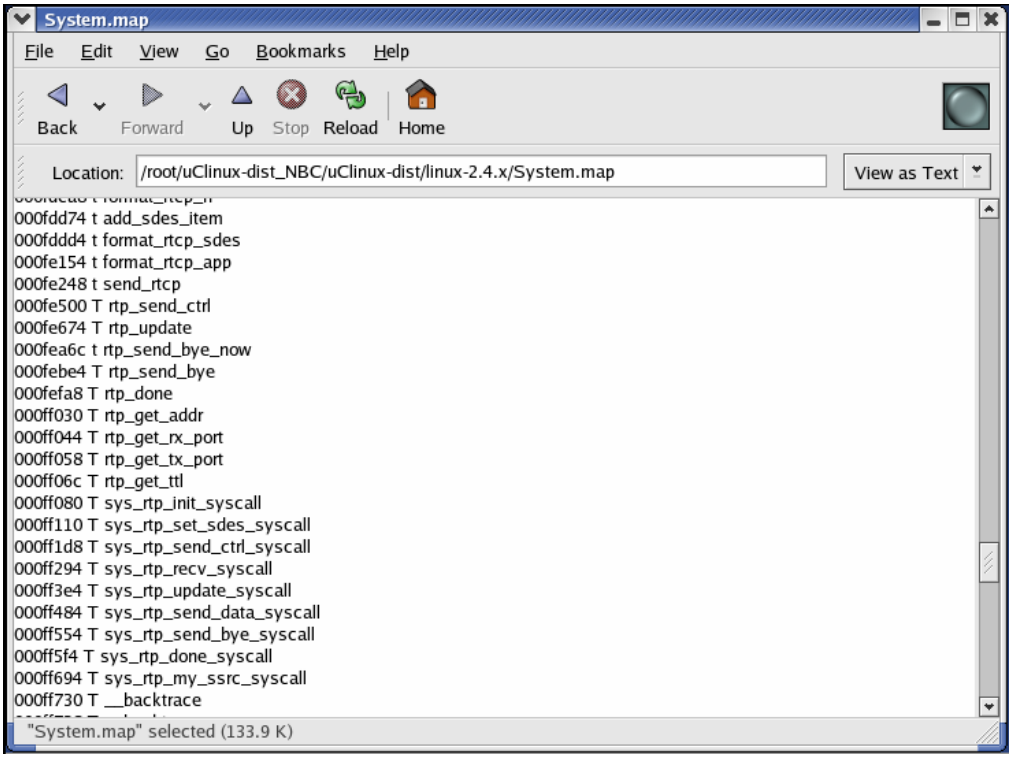
**การติดตั้ง system call ลงบน kernel ของระบบปฏิบัติการ μClinux**

การติดตั้ง system call ลงบน kernel ของระบบปฏิบัติการ μClinux สามารถทำได้ในขั้นตอนของการคอมไพล์และ build image ของระบบปฏิบัติการดังกล่าว ดังนั้นผู้ใช้งานจึงต้องกำหนดให้ติดตั้ง system call ที่ต้องการใช้งานทั้งหมดลงบน kernel ของระบบปฏิบัติการ μClinux ให้เรียบร้อยก่อนเริ่มทำการคอมไพล์และ build kernel ดังกล่าวขึ้นมาใช้งาน ซึ่งงานวิจัยชิ้นนี้ได้ผนวกเอา system call ของโปรโตคอล RTP และ RTCP ที่พัฒนาขึ้นไปเก็บอยู่ภายใน μClinux-dist และได้เพิ่มตัวเลือกที่ใช้สำหรับการกำหนดค่าให้กับ system call ดังกล่าวลงไปในเมนูหลักที่ใช้ในการกำหนดค่าให้กับ kernel ของระบบปฏิบัติการ μClinux เพื่อให้ผู้ใช้งานสามารถเลือกตัวเลือกดังกล่าวสำหรับกำหนดให้ติดตั้ง system call ของโปรโตคอล RTP และ RTCP ลงไปบน kernel ของระบบปฏิบัติการ μClinux โดยอัตโนมัติ สำหรับตัวเลือกของ system call ดังกล่าวจะอยู่ภายในเมนู Main menu (เมนูหลักในการกำหนดค่าต่างๆ ให้กับ kernel ของระบบปฏิบัติการ μClinux) -> Networking options -> Network testing ---> และมีลักษณะดังรูปที่ 1




**ภาพผนวกที่ ข1** ลักษณะของตัวเลือกสำหรับติดตั้ง system ของโปรโตคอล RTP และ RTCP ลงบน kernel ของระบบปฏิบัติการ μClinux

เมื่อเรากำหนดค่าให้กับตัวเลือก RTP Protocol Syscall ดังในรูปที่ 1 แล้วก็จะเป็นการกำหนดให้ติดตั้ง system call ของโปรโตคอล RTP และ RTCP ลงบน kernel ของระบบปฏิบัติการ μClinux โดยอัตโนมัตินั่นเอง ซึ่งหลังจากทำการคอมไพล์และ build image ของระบบปฏิบัติการ μClinux ที่เรากำหนดให้ติดตั้ง system call ดังกล่าวเอาไว้เรียบร้อยแล้ว เมื่อทำการตรวจสอบ symbol ต่างๆ ของ kernel ที่อยู่ภายใน image ดังกล่าวจากไฟล์ System.map จะพบว่ามี system call สำหรับโปรโตคอล RTP และ RTCP ปรากฏอยู่ด้วย ดังรูปที่ 2




**ภาพผนวกที่ ข2** แสดง system call ของโปรโตคอล RTP และ RTCP ที่ปรากฏอยู่ในไฟล์ System.map

ถ้ามี system call ของโปรโตคอล RTP และ RTCP ปรากฏอยู่ภายในไฟล์ System.map ดังในรูปที่ 2 ก็แสดงว่า system call ดังกล่าวได้รับการติดตั้งอยู่ภายใน kernel ของระบบปฏิบัติการ μClinux เรียบร้อยแล้ว ดังนั้นเราจึงสามารถพัฒนาแอพพลิเคชั่นต่างๆ จาก system call เหล่านี้ได้ทันที โดยเมื่อนำแอพพลิเคชั่นที่พัฒนาขึ้นมาจาก system call ดังกล่าวไปติดตั้งใช้งานกับระบบปฏิบัติการ μClinux ที่ภายใน kernel มี system call ของโปรโตคอล RTP และ RTCP ติดตั้งอยู่ แอพพลิเคชั่นเหล่านั้นก็จะสามารถทำงานได้อย่างถูกต้องโดยไม่มีข้อผิดพลาดใดๆ เกิดขึ้น

**หน้าที่และการใช้งาน system call ต่างๆ ของโปรโตคอล RTP และ RTCP รวมทั้งโครงสร้างของข้อมูลที่เกี่ยวข้องกับ system call ดังกล่าว**

**โครงสร้างของข้อมูล**

1) struct socket_udp

```
typedef struct _socket_udp socket_udp;
struct _socket_udp{
        int mode; // IPv4
        char *addr; //เป็นค่า IP address ของผู้ที่สร้าง session นี้ขึ้นมาใช้งาน มีชนิดข้อมูลเป็น string
        uint16_t rx_port; // เลขที่ของ port ที่ใช้ในการรับข้อมูล
        uint16_t tx_port; // เลขที่ของ port ที่ใช้ในการส่งข้อมูล
        struct socket *udp_sock; //โครงสร้าง socket สำหรับ session
        uint32_t addr4; // เป็นค่า IP address ที่ผ่านการแปลงด้วย function in_aton() เรียบร้อยแล้ว
        int sock_fd; // เลขประจำตัวของโครงสร้าง socket udp_sock ของ session นี้
};
```

เป็นโครงสร้างข้อมูลของ UDP Session ที่เป็นพื้นฐานในการสร้างเป็น RTP Session

2) struct rtp_packet

```
typedef struct{
        uint32_t *csrc;
        char *data; // payload ที่บรรจุอยู่ใน RTP packet
        int data_len; // ขนาดของ payload
        unsigned char *extn;
        uint16_t extn_len; /* Size of the extension in 32 bit words minus one */
```

```
        uint16_t extn_type; /* Extension type field in the RTP packet header   */
        /* field ที่อยู่ด้านล่างนี้จับคู่โดยตรงกับ header ต่างๆ ภายใน RTP packet */
#ifdef WORDS_BIGENDIAN
        unsigned short   v:2;           /* packet type          */
        unsigned short   p:1;           /* padding flag         */
        unsigned short   x:1;           /* header extension flag    */
        unsigned short   cc:4;          /* CSRC count           */
        unsigned short   m:1;           /* marker bit           */
        unsigned short   pt:7;          /* payload type         */
#else
        unsigned short   cc:4;          /* CSRC count           */
        unsigned short   x:1;           /* header extension flag    */
        unsigned short   p:1;           /* padding flag         */
        unsigned short   v:2;           /* packet type          */
        unsigned short   pt:7;          /* payload type         */
        unsigned short   m:1;           /* marker bit           */
#endif
        uint16_t        seq;            /* sequence number      */
        uint32_t        ts;         /* timestamp            */
        uint32_t        ssrc;           /* synchronization source   */
}rtp_packet;
```

เป็นโครงสร้างของ RTP packet มีทั้งแบบที่เป็น Big Endian และ Little Endian

3) struct rtcp_sr

```
typedef struct{
        uint32_t        ssrc; // เลข ssrc ของผู้ที่ส่ง RTCP SR packet นี้ออกมา
        uint32_t        ntp_sec; // NTP timestamp most significant word
        uint32_t        ntp_frac; // NTP timestamp least significant word
```

```
        uint32_t        rtp_ts; // ค่า RTP timestamp ที่ตรงกับ NTP timestamp
        uint32_t        sender_pcount; // จำนวนของ packet ที่ฝ่ายส่งส่งออกมา
        uint32_t        sender_bcount; // จำนวน byteของข้อมูลทั้งหมดที่ฝ่ายส่งส่งออกมา
}rtcp_sr;
```

เป็นโครงสร้างของ RTCP SR packet

4) struct rtcp_rr

```
typedef struct {
        uint32_tssrc;                   // เลข ssrc ของผู้ที่ส่ง RTCP RR packet นี้ออกมา
#ifdef WORDS_BIGENDIAN
        uint32_tfract_lost:8;   // อัตราส่วนในการสูญหายของข้อมูล
        uint32_ttotal_lost:24; // จำนวนของข้อมูลที่สูญหายทั้งหมด
#else
        uint32_ttotal_lost:24;
        uint32_tfract_lost:8;
#endif
        uint32_tlast_seq; //sequence number ของ RTP packet ล่าสุดที่รับได้
        uint32_tjitter; // ค่า interarrival jitter
        uint32_tlsr; // เวลาที่ได้รับ RTCP SR ล่าสุดมาจากฝ่ายส่ง
        uint32_tdlsr; // ค่าเวลาที่ delay ระหว่างเวลาที่ได้รับ RTCP SR ล่าสุดมาจากฝ่ายส่งและ
เวลาที่ทำการส่ง RTCP RR packet นี้ออกไป
} rtcp_rr;
```

เป็นโครงสร้างของ RTCP RR packet

5) enum rtcp_sdes_type

```
typedef enum  {
      RTCP_SDES_END   = 0,
      RTCP_SDES_CNAME = 1,
      RTCP_SDES_NAME  = 2,
      RTCP_SDES_EMAIL = 3,
      RTCP_SDES_PHONE = 4,
      RTCP_SDES_LOC   = 5,
      RTCP_SDES_TOOL  = 6,
      RTCP_SDES_NOTE  = 7,
      RTCP_SDES_PRIV  = 8
} rtcp_sdes_type;
```

เป็นโครงสร้างข้อมูลที่เก็บชนิดของ RTCP SDES เอาไว้

6) struct init_rtp_params

```
typedef struct {
        const char *addr; // IP address ปลายทางของ session นี้
        uint16_t rx_port; // เลขที่ของ port ที่ใช้ bind เข้ากับ UDP socket
        uint16_t tx_port; // เลขที่ของ port ที่ใช้ในการส่ง UDP packet
        int ttl; // ค่าของ TTL เมื่อมีการส่งข้อมูลแบบ multicast
        double rtcp_bw; // ค่า bandwidth ทั้งหมด (bytes ต่อ วินาที) ที่กำหนดให้กับ RTCP
        uint8_t *userdata; // ข้อมูลพิเศษบางอย่างที่กำหนดให้กับ session นี้
} init_rtp_params;
```

เป็นโครงสร้างของข้อมูลที่ใช้สำหรับการสร้าง RTP session

7) struct send_data_params

```
typedef struct {
        uint32_t rtp_ts; // ค่า timestamp ที่จะนำไปใช้ในการสร้าง RTP packet
        char pt; // ชนิดของ payload ที่อยู่ภายใน RTP packet
        int m; // maker bit ที่จะกำหนดให้กับ RTP packet
        int cc; // จำนวนของ contributing source ทั้งหมดที่อยู่ภายใน RTP packet
        uint32_t *csrc; //array ของ SSRC สำหรับ contributing source
        char *data; // ข้อมูลที่จะนำไปเก็บใน RTP packet เพื่อส่งออกไป
        int data_len; // ขนาดของข้อมูล
        char *extn; // Extension data (ถ้ามี)
        uint16_t extn_len; // ขนาดของ Extension data
        uint16_t extn_type; // ชนิดของ Extension data
} send_data_params;
```

เป็นโครงสร้างของข้อมูลที่ใช้ในการสร้าง RTP packet สำหรับส่งออกไป ซึ่งจะสังเกตได้ว่าลักษณะของข้อมูลที่อยู่ภายในโครงสร้างนี้มีลักษณะเหมือนกับโครงสร้างของ RTP packet ที่กล่าวถึงไปก่อนหน้านี้

8) struct set_sdes_params

```
typedef struct{
        uint32_t ssrc; // ค่า SSRC ของสมาชิกนั้นๆ ภายใน session
        rtcp_sdes_type type; // ชนิดของ RTCP SDES packet
        const char *value; // ข้อมูลที่จะใช้กำหนดให้กับ source
        int length; // ขนาดของข้อมูล
}set_sdes_params;
```

เป็นโครงสร้างของข้อมูลที่ใช้สำหรับกำหนดค่าให้กับ RTCP SDES packet เพื่อใช้ packet ดังกล่าวในการกำหนดค่าที่ต้องการบางอย่างให้กับ source ของ session ซึ่งจะพบว่าข้อมูลที่อยู่ภายในโครงสร้างนี้จะมีลักษณะเหมือนกับโครงสร้างของ RTCP SDES packet

9) struct options

```
typedef struct {
        int     promiscuous_mode;
        int     wait_for_rtcp;
        int     filter_my_packets;
        int     reuse_bufs;
} options;
```

เป็นโครงสร้างของข้อมูลที่ใช้ในการกำหนด option ต่างๆ ให้กับ session เพื่อให้ session ดำเนินการบางอย่าง

10) struct rtcp_rr_wrapper

```
typedef struct _rtcp_rr_wrapper {
        struct _rtcp_rr_wrapper   *next;
        struct _rtcp_rr_wrapper   *prev;
        uint32_t            reporter_ssrc; // ค่า SSRC ของผู้ที่สร้าง RTCP RR packet นี้ขึ้นมา
        rtcp_rr                 *rr; // โครงสร้างของ RTCP RR packet
        struct timeval          *ts; // เวลาที่ RTCP RR packet นี้เดินทางมาถึง
} rtcp_rr_wrapper;
```

เป็นโครงสร้างที่ใช้จัดเก็บ RTCP RR packet ทั้งหมดที่ได้รับเอาไว้ภายใน session

11) struct source

```
typedef struct _source {
        struct _source    *next;
        struct _source    *prev;
        uint32_t ssrc;
        char              *cname;
        char              *name;
        char              *email;
        char              *phone;
        char              *loc;
        char              *tool;
        char              *note;
        char              *priv;
        rtcp_sr           *sr;
        struct timeval     last_sr;
        struct timeval     last_active;
        int       should_advertise_sdes;
        int       sender; // flag ที่ใช้บ่งบอกว่า source นี้คือผู้ส่งหรือไม่
        int       got_bye; //flag ที่ใช้บ่งบอกว่าได้รับ RTCP BYE packet จาก source นี้หรือไม่
        uint32_t base_seq;
        uint16_t max_seq;
        uint32_t bad_seq;
        uint32_t cycles;
        int       received;
        int       received_prior;
        int       expected_prior;
        int       probation;
        uint32_t jitter;
        uint32_t transit;
```

```
} source;
```

เป็นโครงสร้างข้อมูลที่ใช้เก็บรายละเอียดต่างๆ ของ source ซึ่งต่ออยู่กับ RTP session ใดๆ

12) struct rtp

```
struct rtp {
        socket_udp      *rtp_socket; // UDP socket สำหรับโปรโตคอล RTP
        socket_udp      *rtcp_socket; // UDP socket สำหรับโปรโตคอล RTCP
        char            *addr; // IP address ปลายทางของ session นี้
        uint16_t rx_port; // เลขที่ของ port ที่ใช้ bind เข้ากับ UDP socket
        uint16_t tx_port; // เลขที่ของ port ที่ใช้ในการส่ง UDP packet
        int    ttl; // ค่าของ TTL เมื่อ session นี้ มีการส่งข้อมูลแบบ multicast
        uint32_t my_ssrc; ค่า SSRC ของ source ที่ต่ออยู่กับ session นี้
        int   last_advertised_csrc;
        source   *db[RTP_DB_SIZE]; // ข้อมูลของ source อื่นๆ ทั้งหมดที่ต้องอยู่กับ session นี้
ในที่นี้จะเก็บข้อมูลของสมาชิกที่ต่ออยู่กับ session ดังกล่าวได้ถึง 11 สมาชิก
        rtcp_rr_wrapper  rr[RTP_DB_SIZE][RTP_DB_SIZE];
        options         *opt; //option ของ session นี้
        uint8_t         *userdata;
        int       invalid_rtp_count; // จำนวน RTP packet ที่รับเข้ามาแล้ว format ไม่ถูกต้อง
        int       invalid_rtcp_count; // จำนวนของ RTCP packet ที่รับเข้ามาแล้ว format ไม่
ถูกต้อง
        int       bye_count; //จำนวนของ source ที่ส่ง RTCP BYE packet ออกมา
        int       csrc_count; //จำนวนของ CSRC ทั้งหมดภายใน session นี้
        int       ssrc_count; // จำนวนของ SSRC ทั้งหมดภายใน session นี้
        int       ssrc_count_prev; // ค่า ssrc_count ที่คำนวณได้ก่อนหน้านี้
        int       sender_count; // จำนวนของ source ที่หน้าที่เป็นผู้ส่งภายใน session
        int       initial_rtcp;
        int       sending_bye; // flag ที่ใช้บ่งบอกว่าขณะนี้ source ดังกล่าวกำลังส่ง RTCP BYE
```

```
packet หรือไม่
        double  avg_rtcp_size;
        int     we_sent;
        double          rtcp_bw; // ค่า bandwidth ของ RTCP
        struct timeval  last_update;
        struct timeval  last_rtp_send_time;
        struct timeval  last_rtcp_send_time;
        struct timeval  next_rtcp_send_time;
        double          rtcp_interval; //ค่าที่กำหนดช่วงเวลาในการส่ง RTCP compound packet
ถัดไป
        int             sdes_count_pri;
        int             sdes_count_sec;
        int             sdes_count_ter;
        uint16_t rtp_seq;
        uint32_t rtp_pcount;
        uint32_t rtp_bcount;
        int firsttime_rr; // ใช้ตรวจสอบว่า RTCP RR packet ที่ได้รับเข้ามาจาก source ใดๆ เป็น
RTCP RR packet แรกสำหรับ source ดังกล่าวหรือไม่
        rtp_packet *current_rtp_packet; // RTP packet ล่าสุดที่ได้รับ
};
```

เป็นโครงสร้างข้อมูลของ RTP session ที่ใช้ในการขนส่ง RTP packet และ RTCP compound packet ระหว่าง source ใดๆ

**System call ของโปรโตคอล RTP และ RTCP**

1) rtp_init_syscall

```
int rtp_init_syscall(struct rtp * session, init_rtp_params * init_params)
```

เป็น system call ที่ใช้ในการสร้างและกำหนดค่าเริ่มต้นต่างๆ ให้กับ RTP session โดย argument ที่ต้องส่งให้กับ system call ดังกล่าวได้แก่

- struct rtp * session เป็น pointer ของ RTP session ที่ใช้สำหรับเก็บ RTP session ซึ่ง system call นี้สร้างขึ้น
- init_rtp_params * init_params เป็น pointer ของโครงสร้างข้อมูลที่เก็บค่าต่างๆ ซึ่งจำเป็นสำหรับให้system call ดังกล่าวนำไปใช้ในการสร้าง RTP session ที่ถูกต้อง

ถ้า system call นี้สามารถสร้าง RTP session ได้สำเร็จจะ return ค่า 1 ออกมา

2) rtp_set_sdes_syscall

```
int rtp_set_sdes_syscall(struct rtp * session, set_sdes_params * sdes_params)
```

เป็น system call ที่ใช้ในการกำหนดค่าบางอย่างให้กับ source โดย argument ที่ต้องส่งให้กับ system call ดังกล่าวได้แก่

- struct rtp * session เป็น pointer RTP session ที่ source ดังกล่าวเชื่อมต่ออยู่
- set_sdes_params * sdes_params เป็น pointer ของโครงสร้างข้อมูลที่เก็บค่าต่างๆ ซึ่งจำเป็นสำหรับให้ system call ดังกล่าวนำไปสร้าง RTCP SDES packet เพื่อใช้กำหนดค่าที่ต้องการให้กับ source นั้นๆ ได้อย่างถูกต้อง

ถ้า system call นี้สามารถกำหนดค่าที่ต้องการให้กับ source ได้สำเร็จจะ return ค่า 1 ออกมา

3) rtp_send_ctrl_syscall

```
int  rtp_send_ctrl_syscall(struct rtp * session, uint32_t  rtp_ts)
```

เป็น system call ที่ใช้สำหรับตรวจสอบช่วงเวลาในการส่ง RTCP compound packet และจะทำการส่ง packet ดังกล่าวทันทีเมื่อถึงช่วงเวลาที่กำหนด โดย argument ที่ต้องส่งให้กับsystem call ดังกล่าวได้แก่

- struct rtp * session เป็น pointer ของ RTP session
- uint32_t rtp_ts เวลาปัจจุบันที่อยู่ในหน่วยของ media timestamp

ถ้า system call นี้ทำงานได้สำเร็จจะ return ค่า 1 ออกมา

4) rtp_recv_syscall

| |
|---|
| int rtp_recv_syscall( struct rtp * session, struct timeval * timeout, uint32_t curr_rtp_ts, rtp_packet * p, char * data) |

เป็น system call ที่ใช้ในการรับ RTP packet โดย argument ที่ต้องส่งให้กับsystem callดังกล่าวได้แก่

- struct rtp * session เป็น pointer ของ session ที่จะใช้ในการรับ RTP packet
- struct timeval * timeout เป็นค่าที่กำหนดเวลาซึ่งอนุญาตให้ system call ดังกล่าวรอ RTP packet ที่ส่งมาจากฝ่ายส่งข้อมูล ถ้าsystem callนี้รอ RTP packet ที่ต้องการนานเกินกว่าค่าเวลา timeout ที่กำหนด system call ดังกล่าวจะหยุดการทำงานของตนเองในรอบนั้นทันที
- uint32_t curr_rtp_ts เวลาปัจจุบันที่อยู่ในหน่วยของ media timestamp
- rtp_packet * p คือ RTP packet ที่ได้รับจากการทำงานของ system call ดังกล่าวในรอบนี้
- char * data คือข้อมูลที่อยู่ภายใน RTP packet ซึ่งได้รับเข้ามา

ถ้า system call ดังกล่าวสามารถรับ RTP packet ได้ จะ return ค่า 1 ออกมา แต่ถ้าเกิด timeout เสียก่อนก็จะ return เป็นค่า 0 แทน

5) rtp_update_syscall

```
int rtp_update_syscall( struct rtp * session)
```

เป็น system call ที่ใช้ในการ update ค่าต่างๆ ภายใน session โดย argument ที่ต้องส่งให้กับ system call ดังกล่าวได้แก่

- struct rtp * session เป็น pointer ของ session ที่ต้องการให้ทำการ update ค่า

ถ้า system call นี้ทำงานได้สำเร็จจะ return ค่า 1 ออกมา

6) rtp_send_data_syscall

```
int rtp_send_data_syscall(struct rtp * session, send_data_params * send_params)
```

เป็น system call ที่ใช้สำหรับส่ง RTP packet ออกไปยัง RTP session โดย argument ที่ต้องส่งให้กับ system call ดังกล่าวได้แก่

- struct rtp * session เป็น pointer ของ session ที่จะใช้ในการส่ง RTP packet ออกไป
- send_data_params * send_params เป็น pointer ของโครงสร้างข้อมูลที่ใช้สำหรับสร้าง RTP packet ของข้อมูลที่ต้องการส่งออกไป

ถ้า system call นี้สามารถส่ง RTP packet ออกไปได้สำเร็จจะ return ค่า 1 ออกมา

7) rtp_send_bye_syscall

```
int rtp_send_bye_syscall( struct rtp * session)
```

เป็น system call ที่ใช้ในการส่ง RTCP BYE packet ออกมา โดย source ใดที่ต้องการออกจาก RTP Session ที่ตนเองเชื่อมต่ออยู่ในขณะนั้น จะต้องเรียกใช้ system call นี้เพื่อทำการส่ง RTCP BYE

packet ออกไปให้กับ source อื่นๆ ที่ต่ออยู่กับ RTP session ดังกล่าวก่อนเสมอ โดย argument ที่ต้องส่งให้กับ system call นี้ได้แก่

- struct rtp * session เป็น pointer ของ RTP session ที่ source ดังกล่าวต้องการยกเลิกการเชื่อมต่อ

ถ้า system call นี้ทำงานได้สำเร็จจะ return ค่า 1 ออกมา

8) rtp_done_syscall

```
int rtp_done_syscall( struct rtp * session)
```

เป็น system call ที่ใช้ในการเคลียร์ค่าต่างๆ ของ RTP session ที่ source นั้นๆ ยกเลิกการเชื่อมต่อ โดยจะมีการเรียกใช้ system call นี้ก็ต่อเมื่อ source ดังกล่าวเรียกใช้ system call rtp_send_bye_syscall เรียบร้อยแล้วเท่านั้น ดังนั้น system call นี้จึงทำหน้าที่เปรียบเสมือนการออกจาก RTP session อย่างถาวร ซึ่ง argument ที่จะต้องกำหนดให้กับ system call ดังกล่าวคือ

- struct rtp * session เป็น pointer ของ RTP session ที่ source ดังกล่าวต้องการยกเลิกการเชื่อมต่อ

ถ้า system call นี้ทำงานได้สำเร็จจะ return ค่า 1 ออกมา

9) rtp_my_ssrc_syscall

```
int rtp_my_ssrc_syscall( struct rtp * session, uint32_t ssrc_value)
```

เป็น system call ที่ใช้ดึงค่า SSRC ประจำ source นั้นๆ ออกมา โดย argument ที่จะต้องกำหนดให้กับ system call ดังกล่าวคือ

- struct rtp * session เป็น RTP session ที่ source ดังกล่าวเชื่อมต่ออยู่
- uint32_t ssrc_value เป็นตัวแปรที่ใช้เก็บค่า SSRC ของ source นั้นๆ ออกมา ซึ่งค่า SSRC ที่เก็บอยู่ภายในตัวแปรดังกล่าวก็จะได้จากการทำงานของ system call นี้นั่นเอง

ถ้า system call นี้ทำงานได้สำเร็จจะ return ค่า 1 ออกมา

# ประวัติการศึกษา และการทำงาน

| | |
|---|---|
| ชื่อ –นามสกุล | นายศศิน จันทร์พวงทอง |
| วัน เดือน ปี ที่เกิด | วันที่ 22 ธันวาคม 2522 |
| สถานที่เกิด | สงขลา |
| ประวัติการศึกษา | ระดับปริญญาตรี คณะวิทยาศาสตร์<br>ภาควิชาวิทยาการคอมพิวเตอร์<br>มหาวิทยาลัยสงขลานครินทร์ |
| ตำแหน่งหน้าที่การงานปัจจุบัน | วิศวกรอาวุโส |
| สถานที่ทำงานปัจจุบัน | DTAC |
| ผลงานดีเด่นและรางวัลทางวิชาการ | |
| ทุนการศึกษาที่ได้รับ | |