การพัฒนาโปรแกรม SubManager สำหรับการจัดการเครือข่าย

Development of SubManager for Network Management

โดย

นางสาว นิรมล โกลากุล

รหัส 43067008

อาจารย์ที่ปรึกษา

อาจารย์ อัครินทร์ คุณกิตติ

รายงานนี้เป็นส่วนหนึ่งของวิชาโครงการพัฒนาระบบงาน

หลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ

ภาคเรียนที่ 1 ปีการศึกษา 2545

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ชื่อหัวข้อ	การพัฒนาโปรแกรม SubManager สำหรับการจัดการเครือข่าย
นักศึกษา	นางสาวนิรมล โกลากุล
อาจารย์ที่ปรึกษา	อาจารย์อัครินทร์ คุณกิตติ
ระดับการศึกษา	วิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ
แขนงวิชา	วิทยาการสารสนเทศ
ปีการศึกษา	2545

บทคัดย่อ

การที่เครือข่ายมีขนาดใหญ่ขึ้นทำให้งานด้านการจัดการทำได้ยาก และมีปริมาณที่เพิ่มขึ้น ซึ่งในขณะที่เครือข่ายเหล่านี้มีการจัดโครงสร้างเป็นแบบลำดับชั้น แต่โครงสร้างของการจัดการเครือข่ายยังคงอยู่ในระดับเดียวและถูกควบคุมจากศูนย์กลาง ส่งผลให้สถานีการจัดการต้องทำงานหนักและประสิทธิภาพในการจัดการลดน้อยลง ดังนั้นโครงงานนี้จึงได้ทำการศึกษาวิธีการแก้ไขปัญหาดังกล่าว และพบว่ามีงานวิจัยที่ชื่อว่า Hierarchical Network Management A Concept and its Prototype in SNMPv2 ได้นำเสนอการพัฒนาโปรแกรม SubManager เพื่อเป็นตัวกลางในการติดต่อสื่อสารระหว่าง manager กับ agent ซึ่งช่วยลดภาระในการคำนวณและการติดต่อไปยัง agent แต่สำหรับกรณีที่มีการส่ง traps มาจาก agents เพื่อรายงานเหตุการณ์ต่างๆ โปรแกรมทำได้เพียงแค่รับ traps มาเก็บไว้เท่านั้น โครงงานนี้จึงได้นำโปรแกรมดังกล่าวมาพัฒนาเพิ่มเติมในส่วนที่เกี่ยวกับการจัดการ SNMPv1 trap ให้มีความสามารถในการกรองซ้ำ ตรวจสอบ timeout และตรวจสอบเงื่อนไข กล่าวคือเมื่อพบว่าจำนวนครั้งที่ได้รับ trap เดิมซ้ำครบตามที่กำหนด หรือเกิด timeout หรือพบว่าเมื่อนำ trap ต่างๆ ที่ได้รับมาตรวจสอบเงื่อนไขร่วมกันแล้ว ทำให้เกิดเหตุการณ์ตามที่ได้ระบุไว้ในคอนฟิกไฟล์ จะทำการสร้าง trap เพื่อแจ้งไปยัง manager ซึ่งจากผลการทดลองแสดงให้เห็นว่า การทำงานดังกล่าวช่วยลดปริมาณข้อมูลที่จะส่งไปยัง manager และช่วยให้การจัดการเครือข่ายทำได้ง่ายขึ้น

Title	Development of SubManager for network management
Student	Miss Niramol Kolakul
Advisor	Mr. Akharin Khunkitti
Level of Study	Master of Science in Information Technology
Major	Information Science
Academic Year	2002

Abstract

According to network expansion, the management is hindered and also its number has risen. While large networks are already structured hierarchically, network management has not yet moved from flat to hierarchical structures and controlled by centralize, which has resulted in overloaded processes in management station and degraded management efficiency. One of the appealing alternatives is additional mid-level of network management structure. It has been discovered that there was a research - Hierarchical Network Management A Concept and its Prototype in SNMPv2- that presented a program named SubManager to act as communication entity between management and agent. This program can reduce the number of calculation at the manager and the number of data request to agent. However, it can only receive and store traps. So, this work will develop this program further in part of SNMPv1 trap management including duplication screening, timeout control, and conditional trap-generating. When the same traps which repetitively received from agents, have reached its limited number or excess its timeout, these trap will be sent to manager. Also a kind of trap correlation have met situation which specified in the configuration file, the report trap will be generated and sent to inform the manager. The experiment results for these new abilities shows that it can reduce data sent to the manager and hence the network is managed more easily.

กิตติกรรมประกาศ

ผู้จัดทำขอขอบพระคุณ อาจารย์อัครินทร์ คุณกิตติ ซึ่งได้ให้คำปรึกษา ข้อเสนอแนะต่างๆ จนกระทั่งโครงการพัฒนาระบบงานนี้เสร็จสมบูรณ์ และขอขอบคุณ Mr.Georg Trausmuth ที่กรุณาส่งไฟล์ source code ของโปรแกรม SubManager มาให้ได้ทำการศึกษา และพัฒนาเพิ่มเติมเกี่ยวกับการจัดการ trap ในโครงงานนี้ รวมทั้งขอขอบคุณเพื่อนๆ ทุกท่านที่ได้ให้ความช่วยเหลือมาโดยตลอด

สารบัญ

หน้า

บทคัดย่อภาษาไทย..I
บทคัดย่อภาษาอังกฤษ..II
กิตติกรรมประกาศ..III
สารบัญ..IV
สารบัญตาราง..VI
สารบัญภาพ..VII
บทที่
1. บทนำ..1
1.1 ความสำคัญและที่มา..1
1.2 เป้าหมายของการพัฒนาระบบงาน..2
1.3 วัตถุประสงค์ของการพัฒนาระบบงาน..2
1.4 ขอบเขตของการพัฒนาโปรแกรม..3
1.5 ประโยชน์ที่คาดว่าจะได้รับ..3
1.6 ขั้นตอนการพัฒนาระบบงาน..3
1.7 รายละเอียดของแต่ละบท..4
2. การจัดการระบบเครือข่าย..5
2.1 ระบบการจัดการเครือข่าย..5
2.2 ขอบเขตการจัดการเครือข่าย..6
2.3 สถาปัตยกรรมการจัดการเครือข่าย..6
2.4 โพรโตคอล SNMP..8
2.5 โพรโตคอล SNMPv2..11
2.6 โพรโตคอล SNMPv3..14

2.7 ฐานข้อมูลการจัดการ 14
2.8 Abstract Syntax Notation one 17
2.9 Basic Encoding Rule 18
2.10 การทำงานของโปรแกรม SubManager ต้นแบบ 19
2.11 CMU-SNMP Toolkit Distribution 24
3. กรอบการพัฒนาระบบงาน 37
3.1 เครื่องมือที่ใช้พัฒนาระบบงาน 37
3.2 การทำงานเกี่ยวกับ trap ของโปรแกรม SubManager ต้นแบบ 37
3.3 การพัฒนาฟังก์ชันเพิ่มเติมในส่วนการจัดการ trap 39
4. การออกแบบระบบงาน 45
4.1 รูปแบบคอนฟิกไฟล์ 45
4.2 ลำดับการทำงานของ submgrtrapd..47
4.3 การทำงานของ Controller 49
4.4 การทำงานของ Worker 50
4.5 การทำงานของ Generate TrapOut 56
5. การทดสอบระบบงาน...60
5.1 การทดสอบระบบงาน 60
5.2 สถานการณ์ที่ใช้ทดสอบระบบงาน..62
5.3 TrapIn ที่ใช้ในการทดสอบระบบงาน..65
5.4 ผลการทดสอบระบบงาน..65
6. สรุปผลและข้อเสนอแนะ..69
6.1 สรุปผลการทดสอบระบบงาน..69
6.2 ข้อเสนอแนะ...70
บรรณานุกรม 71
ภาคผนวก ก. การติดตั้งและใช้งานโปรแกรม SubManager ในส่วนการจัดการ SNMPv1 Trap............72
ภาคผนวก ข. ตัวอย่างโปรแกรม SubManager ในส่วนการจัดการ SNMPv1 Trap.............................. 74

สารบัญตาราง

หน้า

ตารางที่

2.1 Error Status ใน SNMP 10
2.2 ชนิดของข้อมูล SNMPv1 Trap 11
2.3 กลุ่มย่อยภายใต้ mib-2 16
2.4 SubMgrEntry 20
2.5 SubMgrOps 21
2.6 SubMgrValue 21
2.7 ตัวอย่าง procedure ที่ใช้ในการทำงาน 23
3.1 ชนิดข้อมูล trap สำหรับ submgrtrapd 43

สารบัญภาพ

หน้า

ภาพที่

2.1 สถาปัตยกรรมการจัดการเครือข่าย 7
2.2 SNMPv1 Message 9
2.3 โครงสร้าง PDU ของ get, get-next และ get-response 9
2.4 โครงสร้าง PDU ของคำสั่ง trap 10
2.5 รูปแบบโครงสร้างการจัดการของ SNMPv2 13
2.6 MIB Tree 15
2.7 โครงสร้าง TLV 19
2.8 การนำ SubManager เข้ามาช่วยจัดการเครือข่าย 19
2.9 ตัวอย่างการใช้งานคำสั่ง snmpget ของ SNMPv1 28
2.10 ตัวอย่างการใช้งานคำสั่ง snmpwalk ของ SNMPv1 29
2.11 ตัวอย่างการใช้งานคำสั่ง snmpset ของ SNMPv1 29
2.12 snmptrapd และคำสั่ง snmptrap ของ SNMPv1 30
2.13 ตัวอย่างการใช้งานคำสั่ง snmpget ของ SNMPv2 33
2.14 ตัวอย่างการใช้ snmptest เพื่อส่ง SNMPv2-TRAP 35
2.15 ผลลัพธ์ที่ได้จาก SNMPv2-TRAP 36
3.1 ภาพรวมการทำงานเกี่ยวกับ traps ของโปรแกรม SubManager ต้นแบบ 38
3.2 ภาพรวมการทำงานเกี่ยวกับ traps ที่พัฒนาเพิ่มเติม 40
4.1 ตัวอย่างข้อกำหนดใน filter.conf 45
4.2 ลำดับการทำงานของ submgrtrapd 48
4.3 การทำงานภายใน Controller 49
4.4 การทำงานของการกรองซ้ำ 51
4.5 การทำงานของการตรวจสอบเงื่อนไข 53

4.6 การทำงานของการตรวจสอบ timeout 55
4.7 การทำงานของ Generate TrapOut ในส่วน Filter..57
4.8 การทำงานของ Generate TrapOut ในส่วน Condition..58
5.1 ตัวอย่างการติดตั้งโปรแกรมด้วยคำสั่ง make all install 60
5.2 ตัวอย่าง filter rules และ condition rules ในไฟล์ filter.conf ที่ใช้ทดลอง 61
5.3 การทดสอบฟังก์ชันการกรองซ้ำ..62
5.4 การทดสอบฟังก์ชันการตรวจสอบ Timeout..63
5.5 การ่ทดสอบฟังก์ชันการตรวจสอบเงื่อนไข..64
5.6 ตัวอย่างการส่ง TrapIn จนครบตามจำนวน *counter* 66
5.7 การทำงานของ submgrtrapd เมื่อได้รับ TrapIn ครบตามจำนวน *counter* 66
5.8 การทำงานของ snmptrapd เมื่อได้รับ TrapOut จากการกรองซ้ำ 66
5.9 ตัวอย่างการส่ง TrapIn ที่จะทำให้เกิด timeout 67
5.10 การทำงานของ submgrtrapd เมื่อได้รับ TrapIn แล้วเกิด timeout 67
5.11 การทำงานของ snmptrapd เมื่อได้รับ TrapOut จากกรณี timeout 67
5.12 ตัวอย่างการส่ง TrapIn ที่จะทำให้เกิด *situation* 68
5.13 การทำงานของ submgrtrapd เมื่อได้รับ TrapIn แล้วเกิด *situation* 68
5.14 การทำงานของ snmptrapd เมื่อได้รับ TrapOut จากการตรวจสอบเงื่อนไขร่วมกัน 68

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ในปัจจุบันการจัดการเครือข่ายได้กลายเป็นสิ่งสำคัญที่ผู้ดูแลระบบจะต้องให้ความสนใจ เนื่องจากการเติบโตอย่างต่อเนื่องของเครือข่ายคอมพิวเตอร์ ที่ประกอบไปด้วยอุปกรณ์คอมพิวเตอร์ต่างๆจากผู้ผลิตหลายรายที่มีมาตรฐานทางการผลิตแตกต่างกัน มาเชื่อมต่อเพื่อแลกเปลี่ยนข้อมูลข่าวสารซึ่งกันและกันนั้น ส่งผลให้การจัดการเครือข่ายทำได้ยากขึ้น จึงจำเป็นต้องใช้คอมพิวเตอร์เข้ามาช่วยในการบริหารและจัดการตัวระบบ โดยในเครือข่ายจะมีคอมพิวเตอร์อย่างน้อยหนึ่งเครื่องทำหน้าที่เป็นสถานีจัดการเครือข่าย หรือเรียกว่า manager มีหน้าที่ตรวจตราและควบคุม agents ซึ่งคืออุปกรณ์เครือข่ายต่างๆ ที่มีฟังก์ชันให้ตรวจสอบและปรับเปลี่ยนการทำงานได้ และเพื่อที่จะให้การจัดการเครือข่ายเป็นไปอย่างมีประสิทธิภาพ จึงได้มีการกำหนดมาตรฐานการจัดการระบบเครือข่ายต่างๆขึ้นมา โดยในปัจจุบันมาตรฐานที่ได้รับความนิยมมากคือโพรโตคอล SNMP (Simple Network Management Protocol) เพราะไม่ซับซ้อน มีขนาดเล็กทำให้ไม่เปลืองทรัพยากร และการจัดการเครือข่ายใน TCP/IP ก็อาศัยรูปแบบการจัดการมาตรฐานตามข้อกำหนดของโพรโตคอลนี้

อย่างไรก็ตามโครงสร้างการจัดการเครือข่ายยังคงมีลักษณะแบบไม่เป็นลำดับชั้น เมื่อมีจำนวน agent มากขึ้น ปริมาณงานในการจัดการเครือข่ายก็จะเพิ่มมากขึ้น ซึ่งเป็นการเพิ่มภาระให้กับ manager นอกจากนี้ยังเกิดปัญหาคอขวดในบริเวณ SNMP interface อีกด้วย

วิธีแก้ปัญหาทางหนึ่งคือการเพิ่ม mid-level ในโครงสร้างการจัดการเครือข่าย ซึ่งได้มีการนำเสนอไว้ในบทความที่ชื่อว่า Hierarchical Network Management A Concept and its Prototype in SNMPv2 (Manfred R.Siegl และ Georg Trausmuth : 1996) โดยบทความนี้นำเสนอแนวคิดในการพัฒนาโปรแกรม SubManager ซึ่งเป็น tool ที่มีการกระจายงานด้าน monitoring แบบง่ายๆ เพื่อทำหน้าที่เป็นตัวกลางระหว่าง manager กับ agent โดยติดตั้งไว้ใกล้กับ agent ที่จะต้องควบคุมดูแล ซึ่งวิธีดังกล่าวนี้สามารถนำไปรวมเข้ากับระบบการจัดการเครือข่ายที่มีอยู่แล้วได้ง่าย และมีความยืดหยุ่นมากกว่าระบบอื่นๆ ที่มีอยู่

หน้าที่ของโปรแกรม SubManager คือคำนวณข้อมูลตามฟังก์ชันที่ถูกกำหนดไว้แทน manager จึงช่วยลดภาระในการคำนวณ และการติดต่อไปยัง agent เพื่อทำการร้องขอข้อมูล

แต่ยังขาดความสามารถในการจัดการเกี่ยวกับ trap ที่ได้รับมาจาก agent เนื่องจากโปรแกรม SubManager ทำได้เพียงแค่รับ trap มาเก็บไว้เท่านั้น ทำให้ manager ไม่สามารถรับรู้เหตุการณ์ต่างๆ ที่ agent แจ้งมา หรือถ้าหากจะให้ agent ทำการส่ง trap ไปยัง manager โดยตรงอาจเกิดปัญหาคอขวดที่บริเวณ SNMP interface ของ manager ได้เมื่อมีการส่ง trap จำนวนมาก ดังนั้นโครงงานนี้จึงได้นำโปรแกรม SubManager ดังกล่าว มาพัฒนาเพิ่มเติมในส่วนที่เกี่ยวกับการจัดการ SNMPv1 trap ให้มีความสามารถในการกรองซ้ำ ตรวจสอบ timeout และตรวจสอบเงื่อนไข กล่าวคือเมื่อพบว่าจำนวนครั้งที่ได้รับ trap เดิมซ้ำครบตามที่กำหนด หรือเกิด timeout หรือพบว่า trap ที่ได้รับเข้ามาทำให้เกิดเหตุการณ์ตามเงื่อนไขที่ได้ระบุไว้ในคอนฟิกไฟล์ จะทำการสร้าง trap เพื่อแจ้งไปยัง manager ซึ่งการทำงานดังกล่าวจะช่วยลดปริมาณข้อมูลที่จะส่งไปยัง manager และช่วยให้การจัดการเครือข่ายทำได้ง่ายขึ้น

1.2 เป้าหมายของการพัฒนาระบบงาน

พัฒนาโปรแกรม SubManager ในส่วนของการจัดการเกี่ยวกับ SNMPv1 trap ที่ได้รับมาจาก agent เพื่อช่วยลดปริมาณข้อมูลที่จะส่งไปยัง manager และช่วยให้การดูแลจัดการเครือข่ายทำได้ง่ายขึ้นโดยการนำ traps ต่างๆ ที่ได้รับนั้นมาตรวจสอบเงื่อนไขร่วมกัน ถ้าตรงตามที่กำหนดไว้ในคอนฟิกไฟล์จะส่งสรุปเป็นเหตุการณ์แจ้งไปยัง manager

โปรแกรมจะมีความสามารถในการทำงานเกี่ยวกับ traps ดังนี้

- ตรวจสอบ community name ของ trap ที่ได้รับจาก agents
- กรองซ้ำ
- ตรวจสอบ timeout
- ตรวจสอบเงื่อนไขที่ระบุไว้ในคอนฟิกไฟล์
- สร้าง trap เพื่อส่งรายงานไปยัง manager

1.3 วัตถุประสงค์ของการพัฒนาระบบงาน

- เพื่อศึกษารูปแบบการจัดการเครือข่าย รวมทั้งมาตรฐานต่างๆ และทฤษฎีที่เกี่ยวข้อง
- เพื่อนำหลักการดังกล่าว มาพัฒนาโปรแกรม SubManager ในส่วนที่เกี่ยวข้องกับการจัดการ traps ซึ่งจะช่วยลดภาระงานด้านการจัดการเครือข่าย

1.4 ขอบเขตของการพัฒนาโปรแกรม

นำโปรแกรม SubManager มาพัฒนาเพิ่มเติมในส่วนของการจัดการเกี่ยวกับ SNMPv1 trap ที่ได้รับมาจาก agent ซึ่งแต่เดิมโปรแกรมมีความสามารถเพียงแค่รับ trap มาเก็บไว้ ดังนั้นโครงงานนี้จึงทำการเพิ่มเติมฟังก์ชันต่างๆ ได้แก่ ฟังก์ชันการตรวจสอบ community name ของ trap ที่ได้รับจาก agents ฟังก์ชันการกรอง traps ฟังก์ชันการตรวจสอบ timeout ฟังก์ชันการตรวจสอบเงื่อนไข และฟังก์ชันในการสร้าง trap เพื่อส่งรายงานไปยัง manager ซึ่งจะทำให้โปรแกรม SubManager สามารถช่วยลดภาระงานของ manager ได้มากยิ่งขึ้น

การพัฒนาโปรแกรม SubManager ในส่วนของการจัดการ traps ดังกล่าว จะสนับสนุนเฉพาะ trap ของ SNMPv1 เท่านั้น โดยจะทำการพัฒนาด้วยภาษาซี ทำงานบนระบบปฏิบัติการ Linux Redhat 7.2 และใช้ library บางส่วนจาก cmu-snmp toolkit distribution เป็นขอบเขตในการพัฒนา

1.5 ประโยชน์ที่คาดว่าจะได้รับ

สถานีจัดการเครือข่ายหรือ manager จะสามารถดูแลจัดการเครือข่ายได้ง่าย และมีความคล่องตัวมากขึ้น เนื่องจากโปรแกรม SubManager ที่พัฒนาเพิ่มเติมในส่วนของการจัดการเกี่ยวกับ SNMPv1 trap นี้จะช่วยสรุปเหตุการณ์จาก trap ต่างๆ ที่ได้รับมาจาก agent รวมทั้งช่วยกรอง trap ซึ่งเป็นการช่วยลดปัญหาคอขวดที่ manager

1.6 ขั้นตอนในการพัฒนาระบบงาน

- ศึกษารูปแบบการจัดการเครือข่าย
- ศึกษาปัญหาที่เกิดขึ้นจากลักษณะโครงสร้างของการจัดการเครือข่าย
- ศึกษาแนวทางในการแก้ไขปัญหา และทฤษฎีต่างๆ ที่เกี่ยวข้อง
- ทำการวิเคราะห์ และออกแบบโปรแกรม SubManager
- ศึกษาเครื่องมือที่จะนำมาใช้ในการพัฒนาระบบงาน
- พัฒนาโปรแกรม SubManager
- ทดสอบการใช้งาน และปรับปรุงแก้ไขโปรแกรมที่พัฒนาแล้ว
- สรุปผลการทดสอบจากการใช้งานที่เกิดขึ้น
- จัดทำเอกสารประกอบโครงงาน

1.7 รายละเอียดของแต่ละบท

- บทที่ 2 : รวบรวมทฤษฎีต่างๆ ที่เกี่ยวข้องกับการจัดการเครือข่าย โดยจะกล่าวถึงลักษณะงานในการจัดการเครือข่าย รูปแบบโครงสร้างการจัดการเครือข่าย โพรโตคอล SNMP และฐานข้อมูลการจัดการ (Management Information Base : MIB) รวมทั้งทฤษฎีเกี่ยวกับ SubManager ด้วย
- บทที่ 3 : กล่าวถึงลักษณะการทำงาน ฟังก์ชันและองค์ประกอบต่างๆ ของโปรแกรม SubManager ทั้งในส่วนที่เป็นต้นแบบ และส่วนของการกรอง traps ที่จะทำการพัฒนาเพิ่มเติม
- บทที่ 4 : การออกแบบโปรแกรมและรูปแบบของคอนฟิกไฟล์ รวมทั้งแสดงความสัมพันธ์ขององค์ประกอบต่างๆ ภายในโปรแกรม
- บทที่ 5 : แสดงผลที่ได้จากการทดลอง สรุปผลการพัฒนาโปรแกรม และข้อเสนอแนะ

บทที่ 2
การจัดการระบบเครือข่าย

การเชื่อมโยงคอมพิวเตอร์เข้าเป็นเครือข่ายก็เพื่อต้องการให้คอมพิวเตอร์สามารถสื่อสารและแลกเปลี่ยนข้อมูลระหว่างกันได้ ช่วยให้ประหยัดเวลาและค่าใช้จ่าย เครือข่ายที่ประกอบด้วยคอมพิวเตอร์และอุปกรณ์จำนวนมาก จะทำงานได้อย่างมีประสิทธิภาพจำเป็นต้องมีการบริหาร และจัดการเครือข่ายที่ดี ซึ่งการจัดการเครือข่ายก็คือการตรวจสอบ ควบคุม จัดการระบบ และวางแผนการใช้ทรัพยากร โดยมีเป้าหมายเพื่อควบคุมการดำเนินงาน การใช้ทรัพยากรต่างๆ และปรับปรุงประสิทธิภาพในการให้บริการ รวมทั้งสามารถเพิ่มการบริการในการเข้าใช้ข้อมูล และทรัพยากรต่างๆ ได้มากขึ้น ถ้าไม่มีการจัดการเครือข่ายก็จะทำให้เกิดความยุ่งยากในการจัดการ ซึ่งผู้ใช้ย่อมไม่อยากพบกับปัญหาที่จะเกิดขึ้นตามมาอย่างแน่นอน

การที่เครือข่ายประกอบไปด้วยอุปกรณ์ต่างๆ เป็นจำนวนมากนั้น ทำให้การจัดการเครือข่ายไม่ใช่งานง่าย และไม่สามารถที่จะจัดการให้มีประสิทธิภาพได้โดยใช้ manual เทคนิคเพียงอย่างเดียว เครือข่ายคอมพิวเตอร์นั้นช่วยให้เราประหยัดเวลาและเงิน แต่ก็ต้องใช้เวลาและเงินในการจัดการดูแลเช่นกัน การจัดการเครือข่ายจึงได้มุ่งเน้นที่จะใช้ความสามารถจากระบบคอมพิวเตอร์และเครือข่ายที่มีอยู่จัดการตัวมันเอง เนื่องจากคอมพิวเตอร์สามารถทำหลายสิ่งหลายอย่างได้ดีกว่ามนุษย์ หนึ่งในนั้นก็คือการจัดการกับปัญหาฉุกเฉินที่เกิดกับทรัพยากรของเครือข่ายได้โดยอัตโนมัติ ดังนั้นการจัดการเครือข่ายจึงต้องใช้เครื่องมือหลายๆ อย่างเข้ามาช่วย

2.1 ระบบการจัดการเครือข่าย (Network Management System)

ระบบการจัดการเครือข่ายคือชุดของเครื่องมือสำหรับการตรวจตราและควบคุมเครือข่าย ซึ่งจะประกอบด้วยฮาร์ดแวร์และซอฟท์แวร์ที่เพิ่มขึ้นจากส่วนประกอบเครือข่ายที่มีอยู่เดิม ซอฟท์แวร์ที่ใช้ในงานจัดการเครือข่ายจะต้องถูกติดตั้งอยู่ใน host computer และ communication processor เช่น front –end processor, terminal cluster controller ระบบจัดการเครือข่ายถูกออกแบบมาเพื่อมองเครือข่ายทั้งหมดเสมือนเป็นโครงสร้างเดียวกัน โดยมีการกำหนดที่อยู่และชื่อให้กับแต่ละจุด ระบุคุณสมบัติของแต่ละส่วน แล้วทำให้ระบบรู้จักสิ่งเหล่านี้ โดยส่วนประกอบต่างๆ ที่ทำงานอยู่ในเครือข่ายจะรายงานสถานะที่กำหนดไว้ต่อศูนย์กลางควบคุมเครือข่าย

2.2 ขอบเขตงานการจัดการเครือข่าย

การจัดการระบบเครือข่ายจำเป็นอย่างยิ่งที่จะต้องมีมาตรฐานการจัดการระบบเครือข่ายออกมารองรับ เพื่อให้การจัดการระบบเครือข่ายเป็นไปตามมาตรฐานเดียวกัน รวมถึงจะส่งผลให้การจัดการระบบเครือข่ายเป็นไปอย่างมีประสิทธิภาพ และง่ายต่อการจัดการ ในส่วนของงานด้านการจัดการนั้น องค์กรที่ชื่อว่า International Organization for Standardization (ISO)ได้ทำการเสนอ OSI Management Framework ซึ่งกำหนดขอบเขตงานของการจัดการเครือข่าย โดยแบ่งออกเป็น 5 ส่วนคือ

1. Fault management จัดการเกี่ยวกับการตรวจจับ รายงานข้อผิดพลาด ติดตาม แยกแยะหาสาเหตุของความผิดพลาดเพื่อนำมาวิเคราะห์และแก้ไขข้อผิดพลาดนั้น
2. Configuration management จัดการเกี่ยวกับการบันทึกองค์ประกอบต่างๆ ของระบบปัจจุบันและเมื่อมีการเปลี่ยนแปลงเกิดขึ้น กำหนดการเริ่มต้นและสิ้นสุดของระบบ รวมทั้งการตั้งชื่อให้กับ object และกลุ่มของ object ในระบบด้วย
3. Accounting management กำหนดขอบเขตในการใช้ทรัพยากรเครือข่าย ดูแลการใช้งานต่างๆ และประเมินค่าใช้จ่ายในอนาคต
4. Performance management เป็นการจัดการเพื่อให้เครือข่ายมีประสิทธิภาพมากที่สุด โดยเก็บบันทึกข้อมูลต่างๆ ลง performance logs สำหรับวิเคราะห์หาความผิดพลาดที่เกิดขึ้น และเป็นข้อมูลในการตัดสินใจเมื่อต้องมีการเปลี่ยนแปลงองค์ประกอบต่างๆ ในระบบ
5. Security management จัดการด้านความปลอดภัยของระบบ กำหนดสิทธิ์ในการใช้ทรัพยากรเครือข่าย การเข้ารหัส การ authenticate และดูแลระบบ firewalls รวมถึงการจัดทำ security logs

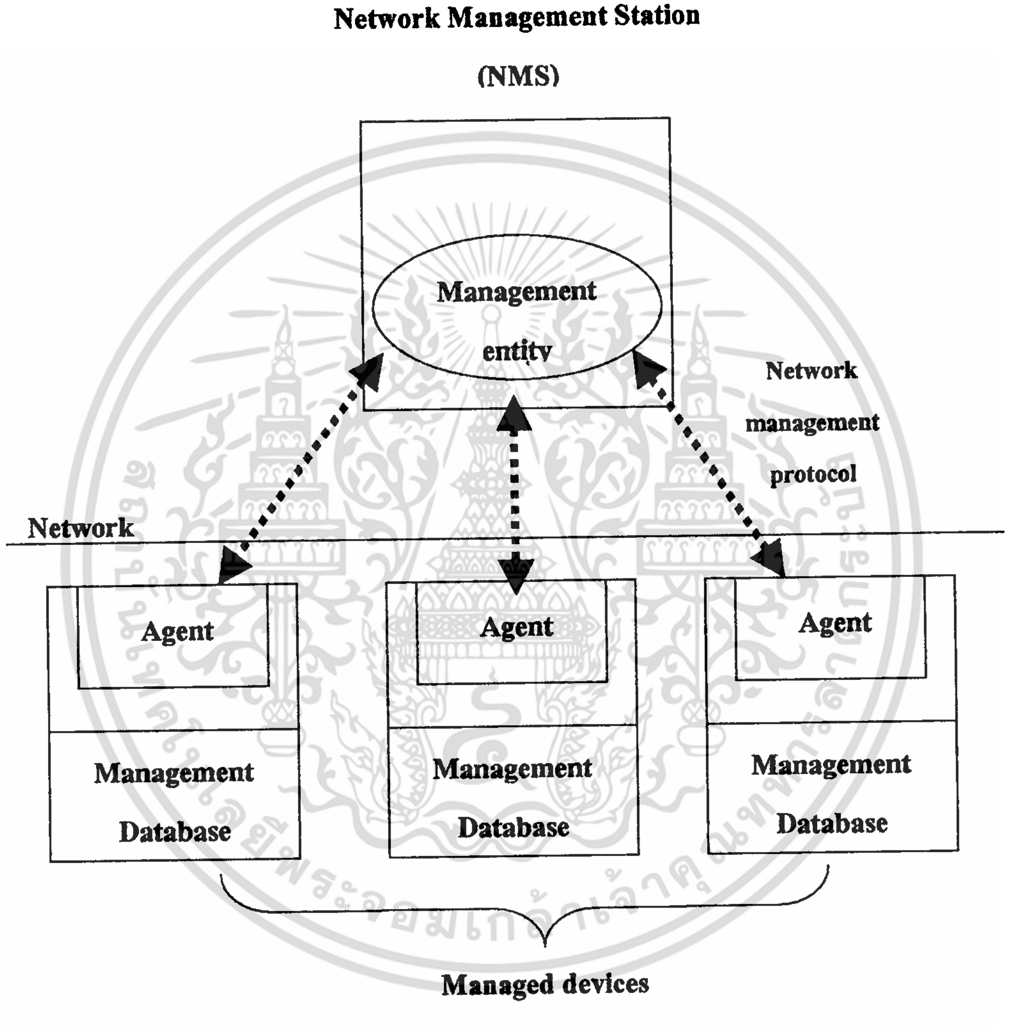
2.3 สถาปัตยกรรมการจัดการเครือข่าย (Network Management Architecture)

สถาปัตยกรรมสำหรับการจัดการเครือข่ายได้ถูกพัฒนาขึ้นโดยหลายๆ องค์กร ที่สำคัญคือ

- The International Organization for Standardization (ISO) มาตรฐานเกี่ยวกับการจัดการเครือข่ายที่เป็นที่รู้จักกันแพร่หลายคือ OSI Management Framework, OSI Systems Management Overview และ Common Management Information Protocol (CMIP)
- The Committee Consultative International on Telegraphy and Telephony (CCITT) ซึ่งปัจจุบันคือ International Telecommunication Union of the Telecommunication Standardization Sector (ITU-T) พัฒนามาตรฐานสำหรับเครือข่ายโทรคมนาคม ที่รู้จักกันดีได้แก่ Telecommunications Management Network (TMN)

- The Internet Engineering Task Force (IETF) พัฒนาโพรโตคอล SNMP

อย่างไรก็ตามสถาปัตยกรรมการจัดการเครือข่ายส่วนใหญ่จะมีโครงสร้างพื้นฐานที่เหมือนกัน คือประกอบด้วยส่วนต่างๆ ดังรูปที่ 2.1



รูปที่ 2.1 สถาปัตยกรรมการจัดการเครือข่าย

- สถานีจัดการเครือข่าย (Network Management Station : NMS) เป็นเครื่องที่ใช้ในการบริหารระบบเครือข่าย ประกอบด้วย
    - โปรแกรมจัดการ (Management Application) สำหรับใช้วิเคราะห์ข้อมูลหรือใช้ตรวจสอบระบบเครือข่าย
    - อินเตอร์เฟส (Interface) ทั้งส่วนที่ใช้ในการแสดงผล และควบคุมระบบเครือข่าย

- ส่วนที่ใช้ดึงข้อมูลจากฐานข้อมูลการจัดการ (MIB) บนอุปกรณ์ต่างๆในระบบเครือข่าย หรือเปลี่ยนแปลงค่าได้

- เอเจนต์ (Agent) คืออุปกรณ์เครือข่าย เช่น พีซี โมเด็ม ฮับ หรือเราท์เตอร์ ที่มีซอฟแวร์เอเจนต์ฝังอยู่ และจะถูกมองว่าเป็น object หนึ่งของระบบจัดการ ซึ่งแต่ละ object จะมีฟังก์ชันให้ตรวจสอบ และปรับเปลี่ยนการทำงานโดย NMS ได้ เอเจนต์ประกอบด้วยส่วนสำคัญสองส่วนคือ
  - โปรโตคอลเอ็นจิน ทำหน้าที่ประมวลคำสั่งที่มาจาก NMS ซึ่งได้แก่ รับคำสั่ง ถอดคำสั่ง ทำงานตามคำสั่งและส่งผลตอบกลับ
  - ฐานข้อมูลการจัดการ (Management Information Base : MIB) เป็นส่วนที่เก็บตัวแปรและค่ากำหนดการทำงานประจำอุปกรณ์ เมื่อ manager ต้องการข้อมูลก็จะระบุตำแหน่งข้อมูลที่ต้องการใน MIB มายัง agent หลังจากนั้น agent ก็จะส่งข้อมูลใน Management Database ที่เหมาะสม กลับไปยัง manager

- โพรโตคอลการจัดการเครือข่าย(Network Management Protocol) ทำหน้าที่กำหนดรูปแบบเมเซจสำหรับใช้จัดการเมเนจอ๊อปเจกต์ (Managed object) รวมไปถึงการตรวจสอบ และกำหนดสิทธิเพื่อการติดต่อสื่อสาร

2.4 โพรโตคอล SNMP (Simple Network Management Protocol)

การที่สถานีจัดการสามารถแลกเปลี่ยนข้อมูลข่าวสารกับเอเจนต์ได้นั้นต้องอาศัยโพรโตคอลในการจัดการเครือข่ายดังที่ได้กล่าวไปแล้ว โดยโพรโตคอลการจัดการเครือข่ายที่ได้รับความนิยมคือ SNMP เพราะไม่ซับซ้อน มีขนาดเล็กทำให้ไม่เปลืองทรัพยากร และที่สำคัญคือการจัดการเครือข่ายใน TCP/IP ก็อาศัยรูปแบบการจัดการมาตรฐานตามข้อกำหนดของโพรโตคอลนี้ โดย SNMP จะอยู่ในชั้น application ทำหน้าที่กำหนดรูปแบบ SNMP message ที่ใช้ในการติดต่อสื่อสารกันระหว่างสถานีจัดการ และ SNMP agent

การติดต่อระหว่างสถานีจัดการกับเอเจนต์มีรูปแบบในการติดต่อหลายรูปแบบขึ้นอยู่กับวัตถุประสงค์ โดยในโพรโตคอล SNMP รุ่น 1 มี 5 แบบคือ

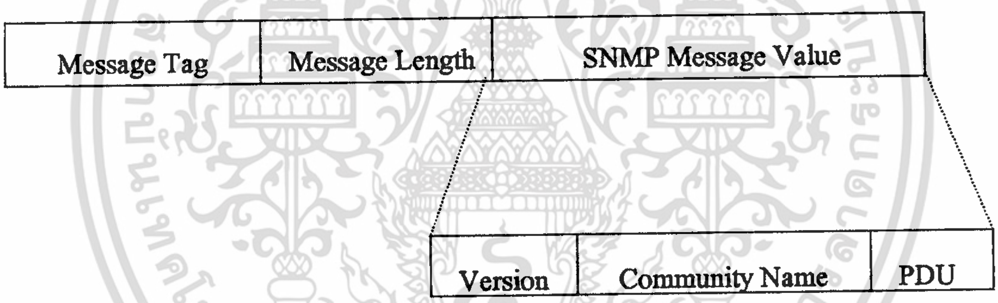
1. get–request : ใช้สอบถามข้อมูลจากตัวเอเจนต์ที่อยู่บนอุปกรณ์ที่ต้องการตรวจสอบในระบบเครือข่าย
2. get–next–request : ใช้สอบถามข้อมูลที่เรียงเป็นลำดับ เช่นข้อมูลที่เก็บอยู่ในรูปตาราง หรือในกรณีที่ไม่ทราบชื่อตัวแปรที่แน่ชัด

3. get–response : ใช้สำหรับเอเจนต์ส่งคำตอบกลับมายังผู้สอบถาม
4. set–request : ใช้เปลี่ยนแปลงค่าตัวแปรที่เอเจนต์รับผิดชอบอยู่
5. trap : ใช้แจ้งเหตุการณ์ที่เกิดขึ้นในระบบเครือข่าย เช่นการเริ่มต้นทำงานใหม่ของอุปกรณ์หรือเส้นทางขัดข้อง

SNMP อาศัยโพรโตคอล UDP โดยใช้ Port หมายเลข 161 สำหรับการติดต่อแบบที่ 1 ถึง 4 และใช้ Port หมายเลข 162 สำหรับการติดต่อแบบที่ 5

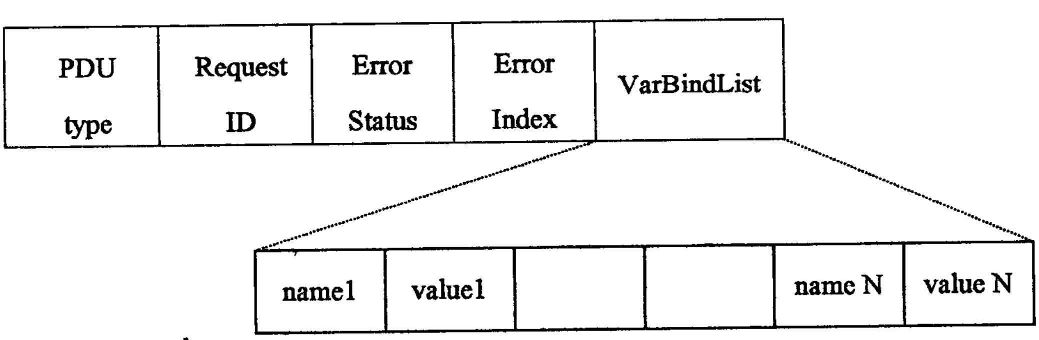
ในส่วนฟอร์แมตของ SNMP ประกอบด้วย header และ PDU โดยในส่วนของ header ประกอบด้วยฟิลด์ย่อย 2 ฟิลด์ ได้แก่

1. version : คือรุ่นของโพรโตคอลที่ใช้ ถ้าเป็นรุ่น 1 จะมีค่า 0 หากเป็นรุ่น 2 จะมีค่า 1
2. community : คือรหัสผ่านในรูป string เพื่อให้เอเจนต์ใช้ตรวจสอบว่าข้อความที่ส่งมามีสิทธิ์ในการสอบถามหรือเปลี่ยนแปลงข้อมูลหรือไม่



รูปที่ 2.2 SNMPv1 Message

ในส่วนของ PDU จะประกอบด้วยฟิลด์ย่อยตามชนิดของข้อความ เช่นหากเป็นข้อความ get, get-next และ get-response จะมีโครงสร้างเดียวกัน โดยแต่ละฟิลด์มีความหมายดังนี้



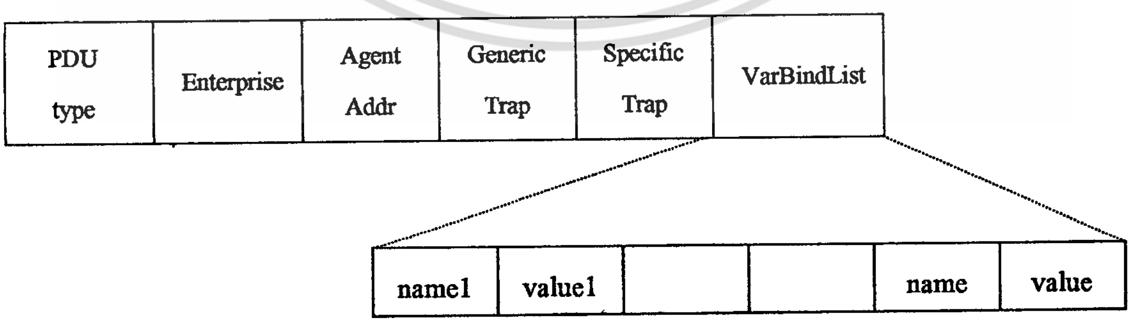
รูปที่ 2.3 โครงสร้าง PDU ของ get, get-next และ get-response

- PDU type : รูปแบบการติดต่อ
- Request ID : กำหนดบอกหมายเลขข้อความเพื่อใช้จับคู่เมื่อรับคำตอบกลับมา
- Error Status : สถานะความผิดพลาดที่เกิดขึ้น แสดงในตารางที่ 2.1
- Error Index : ดรรชนีชี้ค่าผิดพลาดที่เกิดขึ้น ว่าเกิดจากตัวแปรลำดับที่เท่าไหร่ของตัวแปรทั้งหมด ที่ทำการสอบถามไป
- VarBindList : ค่าผูกพันตัวแปร (variable binding) แสดงอยู่ในรูปของตัวแปรและค่าของตัวแปรต่อเนื่องกันไปเป็นรายการ

ตารางที่ 2.1 Error Status ใน SNMP

รหัส	ชื่อ	คำอธิบาย
0	NoError	ไม่มีข้อผิดพลาด
1	TooBig	เอเจนต์ไม่สามารถส่งคำตอบได้ในเฟรมเดียว
2	noSuchName	ไม่มีตัวแปรที่ต้องการสอบถามอยู่ในฐานข้อมูล
3	BadValue	ค่าที่กำหนดให้ตัวแปรไม่ถูกต้อง
4	ReadOnly	เปลี่ยนค่าตัวแปรไม่ได้เพราะอ่านค่าได้เพียงอย่างเดียว
5	GenErr	มีข้อผิดพลาดอื่นๆ เกิดขึ้น

สำหรับข้อความ trap จะมีลักษณะดังรูปที่ 2.4 โดยขนาดความยาวของแต่ละฟิลด์จะไม่ได้นำมากล่าวถึง เนื่องจากทุกฟิลด์ใน SNMP จะต้องเข้ารหัสและจะได้ขนาดของแต่ละฟิลด์ที่มีความยาวแตกต่างกันไปตามชนิดข้อมูล ซึ่งแต่ละฟิลด์มีความหมายดังนี้



รูปที่ 2.4 โครงสร้าง PDU ของคำสั่ง trap

- PDU type : รูปแบบการติดต่อ
- Enterprise : คือ Object ID ของเครื่องที่ทำการส่ง trap
- Agent Addr : คือหมายเลข IP ของเครื่องที่ทำการส่ง trap
- Generic Trap : คือชนิดของ trap ซึ่งใน SNMPv1 แบ่งออกเป็น 7 ประเภท (RFC 1157) ดังตารางที่ 2.2
- Specific Trap : คือชนิดของ trap พิเศษที่สามารถกำหนดขึ้นมาใช้งานเพิ่มเติมได้
- VarBindList : ค่าผูกพันตัวแปร (variable binding) แสดงอยู่ในรูปของตัวแปรและค่าของตัวแปรต่อเนื่องกันไปเป็นรายการ

ตารางที่ 2.2 ชนิดของข้อมูล SNMPv1 trap

trap-type	meaning
0	ColdStart
1	WarmStart
2	LinkDown
3	LinkUp
4	AuthenticationFailure
5	EgpNeighborLoss
6	EnterpriseSpecific

2.5 โพรโตคอล SNMPv2 (RFC 1902 ถึง RFC 1907)

จากหัวข้อ 2.4 จะเห็นได้ว่า SNMPv1 มีข้อดีคือโครงสร้างไม่ซับซ้อน มีรูปแบบการจัดการที่เข้าใจได้ง่าย (manager / agent) และมี community name เพื่อใช้ควบคุมการเข้าถึงข้อมูลใน MIB แต่จะมีข้อเสียคือเมื่อเครือข่ายมีขนาดใหญ่ขึ้น รูปแบบการจัดการดังกล่าวจะไม่มีประสิทธิภาพเพียงพอ ทั้งในด้านการทำงาน และความปลอดภัย

ดังนั้นจึงมีการพัฒนาเป็นเวอร์ชันที่ 2 โดยได้เพิ่มเติมคำสั่ง get-bulk-request เพื่อสอบถามค่าโดยกำหนดจำนวน object ที่ต้องการได้แทนที่จะต้องใช้ get-next-request ซ้ำหลายครั้ง และทำการขยาย MIB เพิ่มเติมด้วย

อีกส่วนหนึ่งที่ได้ทำการปรับปรุงคือ ส่วนของรูปแบบการจัดการ โดย SNMPv2 นั้นจะยอมให้ manager สามารถทำการแลกเปลี่ยนข้อมูลกันได้ เรียกว่าการสื่อสารแบบ M2M (Manager to Manager) ซึ่งทำให้ผู้บริหารเครือข่ายสามารถสร้างสถานีการจัดการให้มีโครงสร้างแบบลำดับชั้นได้ ยกตัวอย่างเช่นบริษัทแห่งหนึ่งมี 20 สาขาเชื่อมต่อกัน โดยแต่ละสาขามีเครือข่ายย่อยซึ่งประกอบด้วย 500 เครื่อง ดังนั้นถ้าใช้ SNMPv1 สถานีจัดการจะต้องดูแลเครื่องทั้งหมดถึง 10,000 เครื่อง ซึ่งถือได้ว่าเป็นภาระงานที่หนักมาก แต่ภายใต้ SNMPv2 เราสามารถจัดให้แต่ละเครือข่ายย่อยสามารถมีสถานีจัดการเป็นของตัวเอง แล้วจึงให้สถานีเหล่านี้ติดต่อสื่อสารกับสถานีการจัดการส่วนกลางอีกทีหนึ่ง

นอกจากนี้ยังมีการปรับปรุงด้านความปลอดภัยให้สมบูรณ์ยิ่งขึ้น คือได้มีการนำแนวคิดที่ชื่อว่า context เข้ามาใช้งานแทน community name ซึ่งแต่เดิมนั้นถูกใช้ในการควบคุมหลายส่วนมากเกินไป จึงทำให้เกิดความไม่น่าเชื่อถือ โดยจะต้องดูแลในส่วนเหล่านี้ทั้งหมด

- The identity of the manager
- The identity of the agent
- The location of the MIB information
- Authentication information
- Access control information
- MIB view information

ดังนั้น SNMPv2 จึงได้แบ่งหน้าที่ในการดูแลดังที่กล่าวข้างต้นออกเป็นส่วนๆ เพื่อให้มีความยืดหยุ่นในการควบคุมการเข้าถึงข้อมูลต่างๆ ใน MIB และทำให้มีความปลอดภัยมากยิ่งขึ้น

- Context

context เป็นเสมือนแหล่งที่เก็บรวบรวมข้อมูลเกี่ยวกับการจัดการ (MIB Information) ซึ่งใน SNMPv2 ได้แบ่ง context ออกเป็น 2 ประเภท กล่าวคือถ้าข้อมูลถูกเก็บอยู่ในเครื่องที่เป็น agent จะเรียก context ว่า MIB view และหากข้อมูลถูกเก็บอยู่ในเครื่องที่ไม่ได้สนับสนุน SNMP แต่ทำหน้าที่เป็นเพียง proxy จะเรียก context ว่า proxy relationship

- Party

SNMPv2 ได้เสนอแนวคิดเกี่ยวกับ party ขึ้น เพื่อให้เห็นสิทธิที่ชัดเจนว่าใครสามารถทำอะไร กับข้อมูลส่วนใดได้บ้าง โดย party ก็คือ SNMP entity (manager, agent หรือ proxy machine) ซึ่งแนวคิดนี้จะมุ่งเน้นไปที่บทบาทหน้าที่ของ party ยกตัวอย่างเช่น agent หนึ่งเครื่องอาจมีการกำหนดสิทธิในการเข้าถึงข้อมูล(read หรือ write)ใน MIB tree สำหรับ manager A และ manager B แตกต่างกัน

รูปแบบโครงสร้างการจัดการของ SNMPv2 จะเป็นดังรูปที่ 2.5 มี root คือ ACL(Access Control List) และใน ACL ประกอบด้วยส่วนย่อยๆ ดังนี้

- Source Party : ระบุว่าใครเป็นผู้ร้องขอ
- Destination Party : ระบุว่าใครเป็นผู้รับการร้องขอนั้น
- Context : ระบุว่าเป็นข้อมูลจากส่วนใด
- Privileges : ระบุว่า Destination Party สามารถที่จะดำเนินการอะไรให้กับ Source Party ได้บ้าง

นอกจากนี้เพื่อให้การติดต่อสื่อสารระหว่าง party เป็นความลับ ดังนั้น SNMPv2 จึงได้ออกแบบให้มีการใช้ DES encryption ในการตรวจสอบ privacy และใช้ MD5 digest algorithm ในการตรวจสอบ authentication อีกด้วย

```
                                                  |- - > Get
        |- - > Privileges - - - - - - - - - - - - - - - - - - - - - - >|
        |                                         |- - > Set
        |
        |                                         |- - > IP Address
        |- - > Destination Party - - - - - - - - - - - - - - - - - >|- - > Privacy Keys
        |                                         |- - > Authentication Keys
ACL - - >|
        |                                         |- - > IP Address
        |- - > Source Party - - - - - - - - - - - - - - - - - - - - >|- - > Privacy Keys
        |                                         |- - > Authentication Keys
        |
        |                                         |- - > MIB View
        |- - > Context - - - - - - - - - - - - - - - - - - - - - - - ->|
                                                  |- - > Proxy Relationship
```

รูปที่ 2.5 รูปแบบโครงสร้างการจัดการของ SNMPv2

2.6 โพรโตคอล SNMPv3 (RFC 2273 ถึง RFC 2275)

นำ SNMPv2 มาพัฒนาเพิ่มเติมโดยมุ่งเน้นด้านความปลอดภัยของการดูแลเครือข่าย มีการให้สิทธิและการดูแลในด้านความเป็นส่วนตัวมากขึ้น มีการตรวจสิทธิและควบคุมการเข้าถึงข้อมูล นอกจากนี้ยังเพิ่มความสามารถในด้านการบริหารเครือข่าย ทำการให้นโยบายแก่เครือข่าย ดูแลจัดการเรื่องชื่อและรหัสต่างๆของผู้ใช้ มีความสัมพันธ์และการทำงานกับ proxy ที่ดีขึ้น รวมทั้งสนับสนุนความสามารถในการกำหนดองค์ประกอบต่างๆ จากระยะไกลผ่านกระบวนการของ SNMP

2.7 ฐานข้อมูลการจัดการ (Management Information Base : MIB)

ฐานข้อมูลการจัดการเป็นฐานข้อมูลที่ใช้เก็บค่าข้อมูลต่าง ๆ ของ managed object เพื่อใช้สำหรับการตั้งค่าและรายงานผลของตัวอุปกรณ์ ซึ่งในเครือข่ายหนึ่งๆ ประกอบด้วยอุปกรณ์หลายชิ้น และอุปกรณ์แต่ละชิ้นมีข้อมูลได้มากมาย อีกทั้งอุปกรณ์ต่างประเภทกันย่อมมีข้อมูลประจำอุปกรณ์แตกต่างกัน ดังนั้นการสอบถาม หรือปรับเปลี่ยนค่าฐานข้อมูลจึงต้องมีรูปแบบมาตรฐานให้กับอุปกรณ์ทุกประเภท โครงสร้างต้นไม้แบบลำดับชั้นเป็นโครงสร้างที่เหมาะสมสำหรับใช้เป็นฐานข้อมูลเพื่อจัดเก็บตัวแปรเหล่านี้

2.7.1 โครงสร้าง MIB

ข้อมูลการจัดการเครือข่ายที่ agent จัดเตรียมให้กับ manager นั้น ถูกจัดเก็บอยู่ในส่วนที่เรียกว่า MIB tree ดังรูปที่ 2.6 ซึ่งมีลักษณะโครงสร้างเป็นแบบต้นไม้ที่แสดงข้อมูล หรือ object ของ SNMP โดยแต่ละโหนดจะแทนแต่ละ object ซึ่งมีชื่อพร้อมทั้งเลขฐานสิบประจำโหนดเพื่อใช้อ้างอิง ยกเว้น root จะไม่มีชื่อกำกับ ยกตัวอย่างโหนดที่ชื่อว่า system จะประกอบไปด้วยหลาย object เช่น sysDescr sysContact sysName และ sysLocation โดย objects เหล่านี้จะเป็นตัวบอกชนิดของข้อมูล และในแต่ละ object ก็จะมีส่วนที่เรียกว่า instances ซึ่งเก็บค่าของข้อมูลไว้

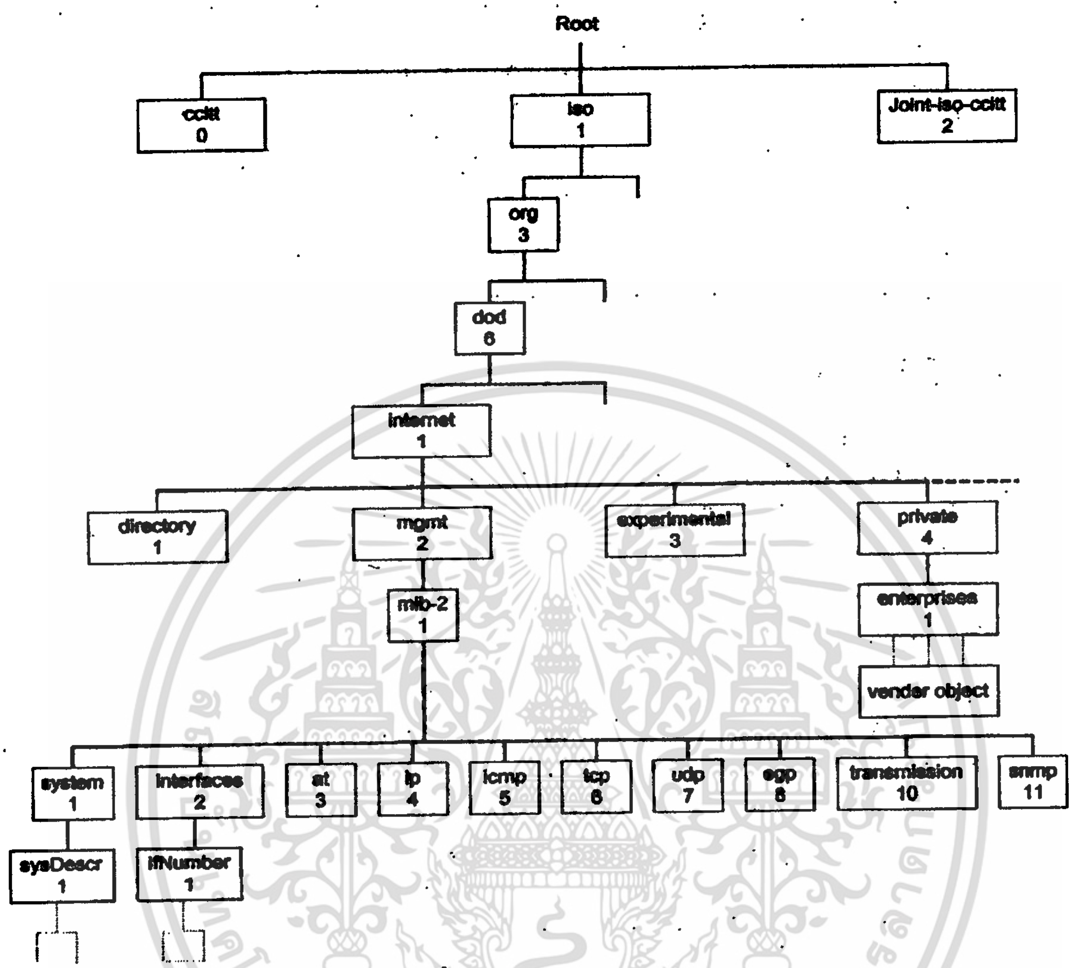
เพื่อให้เห็นความแตกต่างระหว่าง objects และ instances ให้พิจารณาการอ้างถึงข้อมูลใน MIB ดังต่อไปนี้

- ถ้าจะอ้างถึง sysContact object เราจะใช้สัญลักษณ์

.iso.org.dod.internet.mgmt.mib-2.system.sysContact

- แต่ถ้าจะระบุถึงค่าของ sysContact ที่อยู่บนเครื่อง agent จะต้องใช้สัญลักษณ์ดังนี้

.iso.org.dod.internet.mgmt.mib-2.system.sysContact.0



รูปที่ 2.6 MIB Tree

จากตัวอย่างจะสังเกตเห็นได้ว่าโหนดแรก(iso)นั้น จะต้องนำหน้าด้วยจุดทศนิยมเสมอ เพื่อเป็นการระบุเส้นทางว่าเริ่มต้นจาก root ของ MIB tree

นอกจากการอ้างถึงข้อมูลด้วยชื่อโหนดแล้ว เรายังสามารถอ้างถึงโดยใช้เลขที่อยู่ถัดจากชื่อโหนดได้อีกด้วย โดยเขียนหมายเลขจาก root ไปตามเส้นทางถึงโหนดนั้นและคั่นด้วยจุด ลำดับตัวเลขนี้เรียกว่า object identifier (OID) เช่นในการระบุถึงโหนด iso.org.dod.internet.mgmt.mib-2.system เราจะอ้างถึงโดยระบุเลข .1.3.6.1.2.1.1

ลำดับชั้นแรกของ MIB tree จะมีโหนดหลักสามโหนด ซึ่งกำหนดองค์กรสามกลุ่มคือ ITU-T ISO และ Joint-ISO-ITU-T ภายใต้โหนด ISO มีโหนดลำดับที่สามคือ org กำหนดองค์กรนานาชาติ และส่วนหนึ่งขององค์กรนี้คือ dod หรือ Department of Defense และมีโหนด internet

เพื่อกำหนดกลุ่มการจัดการเครือข่ายในอินเตอร์เน็ต โดย MIB ภายใต้ internet มีกลุ่มย่อยทั้งหมด 6 กลุ่มคือ

- directory สงวนไว้สำหรับใช้งานในอนาคต
- mgmt กลุ่ม MIB ที่ใช้ในการจัดการภายใต้ SNMPv1
- experimental ใช้สำหรับการทดลอง
- private สำหรับให้ผู้ผลิตกำหนดตัวแปรเฉพาะอุปกรณ์
- security ใช้ในระบบรักษาความปลอดภัย
- SNMPv2 ใช้ใน SNMPv2

สำหรับกลุ่มย่อยที่อยู่ภายใต้กลุ่ม mib-2 จะใช้ใน SNMP โดยแต่ละกลุ่มจะประกอบด้วยตัวแปรซึ่งมีรูปแบบต่างๆ กันไปดังตารางที่ 2.3

ตารางที่ 2.3 กลุ่มย่อยภายใต้ mib-2

ลำดับ	ชื่อ	ความหมาย
1	system	ข้อมูลระบบ
2	interfaces	ข้อมูลอินเตอร์เฟสที่ใช้เชื่อมต่อ
3	at	ข้อมูลการแปลง address
4	ip	ข้อมูลไอพี
5	icmp	ข้อมูล ไอซีเอ็มพี
6	tcp	ข้อมูลทีซีพี
7	udp	ข้อมูลยูดีพี
8	egp	ข้อมูลโปรโตคอลเกตเวย์ภายนอก
10	transmission	ข้อมูลสายสื่อสาร
11	snmp	ข้อมูลเอสเอ็นเอ็มพี

2.7.2 ชนิดของตัวแปร MIB

- integer : จำนวนเต็ม มีค่าได้ตั้งแต่ 0 ถึง 65535 เช่นหมายเลข port ของ TCP หรือ UDP
- OctetString : ชนิดข้อมูลขนาดตั้งแต่ 0 octet แต่ละ octect มีค่าตั้งแต่ 0 ถึง 255 หนึ่ง octect จะใช้แทนตัวอักษรหนึ่งตัว เช่นรหัสผ่าน

- DisplayString : ชนิดข้อมูลขนาดตั้งแต่ 0 octet แต่ละ octect ต้องเป็นรหัส ASCII NVT ข้อมูลประเภทนี้มีความยาวตั้งแต่ 0 ถึง 255
- Null : ใช้บอกว่าตัวแปรนั้นไม่มีค่าข้อมูลใดอยู่
- Objectidentifier : ชื่อตัวแปรในรูปของการอ้างถึงแบบตัวเลขตามโครงสร้าง MIB
- IpAddress : ชนิดข้อมูลที่ประกอบด้วย 4 octect แต่ละ octect แทนไอพีแอดเดรสแต่ละตำแหน่ง
- PhysicalAddress : ชนิดข้อมูลที่กำหนดฮาร์แวร์แอดเดรส เช่น Ethernet ใช้ 6 octect
- Counter : เลขจำนวนเต็มไม่คิดเครื่องหมาย มีค่าตั้งแต่ 0 ถึง 2^{23}-1 การใช้ข้อมูลชนิดนี้เป็นแบบเพิ่มค่าขึ้นอย่างเดียวและเมื่อถึงค่ามากสุดจะกลับเป็น 0
- Gauge : เลขจำนวนเต็มไม่คิดเครื่องหมายมีค่าตั้งแต่ 0 ถึง 2^{23}-1 โดยสามารถเพิ่มหรือลดค่าได้ แต่เมื่อเพิ่มไปสูงสุดแล้วจะคงค่าไว้ (Latches) จนกว่าจะถูกปรับค่ากลับมาเป็น 0 อีกครั้ง
- TimeTicks : เลขจำนวนเต็มใช้นับเวลามีหน่วยเป็น 1/100 วินาที
- Sequence : โครงสร้างแบบ record ใช้สำหรับการกำหนด manage object เป็นแบบ list
- Sequence of : โครงสร้างแบบ array เช่นตารางเลือกเส้นทางของไอพี

2.8 Abstract Syntax Notation one (ASN.1)

เนื่องจากบนเครือข่ายมีอุปกรณ์มากมายที่แตกต่างกันทั้งในเรื่องสถาปัตยกรรม และข้อมูลที่ใช้ในการประมวลผล ดังนั้นจึงต้องมีการกำหนดมาตรฐานและรูปแบบของข้อมูลที่ใช้สื่อสารผ่านเครือข่าย สำหรับการติดต่อสื่อสารระหว่างสถานีจัดการ และ SNMP เอเจนต์นั้น ได้ใช้ภาษา ASN.1 ที่กำหนดขึ้นโดยองค์กร ISO มาใช้ในการกำหนดรูปแบบของ SNMP Message โดยโครงสร้างของภาษาแบ่งออกเป็น 3 ส่วนคือ

- Module Name : เป็นการตั้งชื่อโมดูล มีรูปแบบคือ *module name* DEFINITIONS ::= BEGIN เช่นถ้าต้องการตั้งชื่อเป็น RFC1157-SNMP สามารถเขียนได้ดังนี้

  RFC1157-SNMP DEFINITIONS ::= BEGIN

- Linkage Statement : ส่วนนี้เป็นการอนุญาตให้โมดูลสามารถ Import และ Export ข้อมูลกับโมดูลอื่นๆ ได้ เช่น

```
IMPORTS
    ObjectName, ObjectSyntax, NetworkAddress, IpAddress, TimeTicks
      FROM RFC1155-SMI;
```

- Declaration Statement : เป็นส่วนในการนิยามอ็อบเจ็ก เช่น นิยามของอ็อบเจ็ก sysContact สามารถเขียนได้ดังนี้

```
sysContact    OBJECT-TYPE
              SYNTAX        DisplayString (SIZE (0..255))
              ACCESS        read-write
              STATUS        mandatory
              DESCRIPTION
                            " The textual identification of the contact person
                            for this managed node, together with information
                            on how to contact this person. "
              : : = {system 4}
```

นิยามข้างต้นมีความหมายดังนี้

- SYNTAX : กำหนดรูปแบบข้อมูลของตัวแปร
- ACCESS : กำหนดรูปแบบการเข้าใช้ เช่น อ่านอย่างเดียว(read-only), อ่านเขียนได้(read-write) หรือห้ามเข้าถึง(not-accessible)
- STATUS : กำหนดสถานะของตัวแปรว่าต้องจัดให้มีตัวแปรนี้หรือไม่ เช่น จำเป็น(mandatory), มีหรือไม่มีก็ได้(optional), ไม่จำเป็นเนื่องจากยกเลิกไม่ใช้แล้ว(obsoleted)
- DESCRIPTION : ข้อความอธิบายตัวแปร
- บรรทัดสุดท้ายกำหนดว่าตัวแปร sysContact นี้จะเชื่อมกับโครงสร้างต้นไม้ต่อจากโหนด system และมีค่าเท่ากับ 4 ซึ่งแสดงให้เห็นว่าตัวแปรนี้มี identifier คือ iso.org.dod.internet.mgmt.mib-2.system.4 หรือ 1.3.6.1.2.1.1.4

2.9 Basic Encoding Rule (BER)

จากที่กล่าวข้างต้นว่าในการติดต่อสื่อสารกันระหว่างสถานีจัดการกับ SNMP เอเจนต์นั้น จะใช้ SNMP Message ที่ถูกกำหนดโครงสร้างด้วยภาษา ASN.1 ในการแลกเปลี่ยนข้อมูลข่าวสาร อย่างไรก็ตามการส่งข้อมูลใน SNMP ไม่ได้ส่งในรูป octets ของตัวเลขโดยตรง แต่ใช้การเข้ารหัสแบบ BER ก่อนแล้วจึงนำข้อมูลส่งผ่านเครือข่าย โดยข้อมูลที่ถูกส่งเป็นอันดับแรกคือข้อมูลบิตที่ 8 เป็น

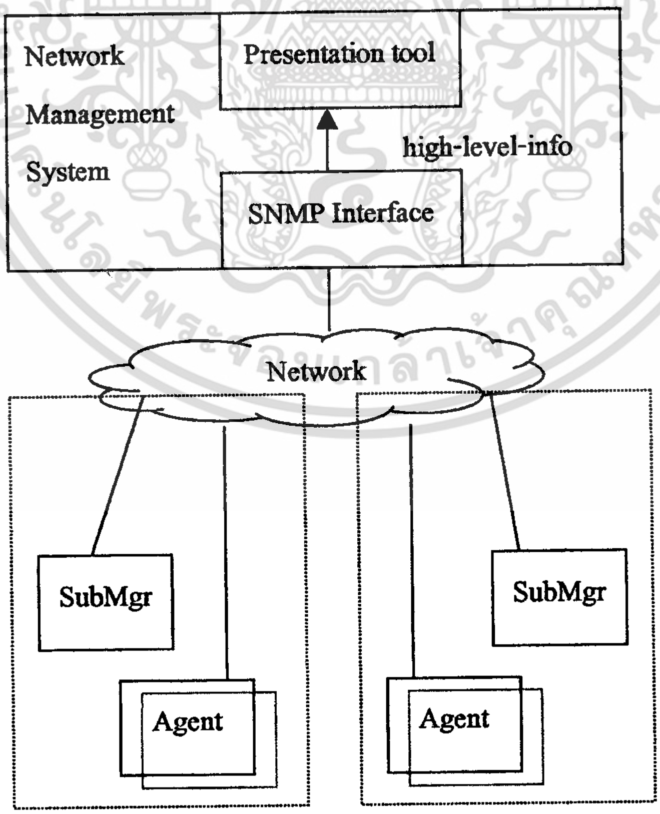
บิตที่มีนัยสำคัญสูงสุด(MSB) และบิตที่ส่งเป็นอันดับสุดท้ายคือบิตที่ 1 เป็นบิตที่มีนัยสำคัญต่ำสุด(LSB) SNMP Message ที่ผ่านการเข้ารหัสแล้วจะประกอบไปด้วย 3 ส่วน คือ ชนิดข้อมูล(tag หรือ type), ความยาวข้อมูล(length) และตัวข้อมูล(value) ซึ่งโครงสร้างรหัสนี้เรียกว่า โครงสร้าง TLV ดังรูปที่ 2.7

Type	Length of contents	Value

รูปที่ 2.7 โครงสร้าง TLV

2.10 การทำงานของโปรแกรม SubManager ต้นแบบ

โปรแกรมนี้มีแนวคิดคล้ายกับ manager to manager (M2M) แต่ใช้การเพิ่ม manager อีกตัวหนึ่งเข้าไประหว่าง manager และ agents โดยเรียกว่า SubManager เพื่อเข้ามาช่วยงานด้านการจัดการเครือข่าย ดังรูปที่ 2.8



รูปที่2.8 การนำ SubManager เข้ามาช่วยจัดการเครือข่าย

SubManager จะทำงานตามคำสั่งที่ได้กำหนดไว้ โดยอาจต้องมีการดึงข้อมูลมาจาก agent เพื่อนำมาประมวลผล และเก็บผลลัพธ์ที่ได้ไว้ในฐานข้อมูลของตน เมื่อ manager ต้องการข้อมูลเหล่านี้ก็สามารถมาดึงค่าไปใช้งานได้โดยใช้ SNMP เช่นคำสั่ง get-request แต่ถ้าในส่วนสถานะระบุว่าเกิดข้อผิดพลาด ข้อมูลนั้นจะถือว่าเป็นค่าที่ใช้ไม่ได้ ส่วนเวลาที่ทำการคำนวณจะดูได้จากค่า timestamp ซึ่งการทำงานดังกล่าวจะช่วยลดปัญหาที่คอขวดในบริเวณ SNMP interface เนื่องจากได้ย้ายส่วนของการคำนวณมาไว้ที่ SubManager นอกจากนี้เมื่อจำนวน agent เพิ่มมากขึ้นจะช่วยให้การจัดการเครือข่ายทำได้สะดวกขึ้นด้วย

2.10.1 ฟังก์ชันและองค์ประกอบของ SubManager

ภายใน SubManager จะประกอบด้วยฟังก์ชันหลักในการทำงาน คือฟังก์ชันที่ใช้ในการตีความคำสั่งแล้วประมวลผล และประกอบด้วยฐานข้อมูลซึ่งแสดงในรูปของตาราง มีรายละเอียดดังนี้

- ฟังก์ชันที่ใช้ในการคำนวณผลลัพธ์

SubManager จะเริ่มทำงานตามช่วงเวลาที่กำหนดไว้ ซึ่งรายละเอียดของ procedure เหล่านี้จัดเก็บอยู่ในตาราง SubMgrEntry และ SubMgrOps โดยในการตีความคำสั่ง และทำการประมวลผลนั้นจะมีการส่ง get_request ไปยัง agent เพื่อสอบถาม arguments ต่างๆ เมื่อคำนวณผลลัพธ์แล้วจะทำการเก็บลงในตาราง SubMgrValue เพื่อรอให้ manager มานำค่าเหล่านี้ไปใช้งาน

- ฐานข้อมูลของ SubManager

ฐานข้อมูลหรือ MIB นั้นจะแสดงให้เห็นรายละเอียดข้อมูลที่จัดเก็บในรูปของตาราง โดยตารางที่ 2.4 และ2 .5 ได้แก่ SubMgrEntry SubMgrOps ใช้ในการเก็บข้อมูลเกี่ยวกับ procedure ส่วนตารางที่ 2.6 คือ SubMgrValue ใช้เก็บผลลัพธ์ที่ได้จากการคำนวณ

ตารางที่ 2.4 : SubMgrEntry

Element	Description
SubMgrValues	Number of history values
SubMgrPolling	Polling interval
SubMgrLines	Number of statements
SubMgrStatus	Status

ตารางที่ 2.5 : SubMgrOps

Element	Description
subMgrOpsCode	Operation code
subMgrOpsOpnd	Operand
subMgrOpsStatus	Status

ตารางที่ 2.6 : SubMgrValue

Element	Description
subMgrValValue	Value of computation
subMgrValTime	Timestamp
subMgrValStatus	Status

2.10.2 หลักการคำนวณ

ฟังก์ชันหรือสูตรที่ใช้ในการคำนวณที่กำหนดนั้นจะอาศัยหลักการของ stack โดยคำสั่งที่ใช้ในการใส่ค่าลงใน stack และทำการดึงค่าเหล่านั้นขึ้นมา หรือที่เรียกว่า Storage commands ประกอบด้วย

- pushConst : ใส่ค่าที่เป็น 32 bit integer ลงบน stack
- getValV1 : อ่านค่าโดยใช้ SNMPv1 แล้วนำไปใส่ไว้ใน stack โดยต้องทำการระบุ Object Identifier ของค่าการจัดการเหล่านี้รวมทั้งต้องระบุ IP Address ด้วย
- getValV2 : อ่านค่าผ่านทาง SNMPv2 get-request แล้วนำค่าเหล่านี้ใส่ลงไปใน stack โดยต้องมีการระบุ Object Identifier, source and target party และ context
- pull : ทำการ destroy ค่าที่อยู่ด้านบนสุดของ stack ซึ่งจะสามารถทำการ destroy ได้มากกว่า 1 ค่า
- stoReg : การนำค่าที่อยู่บนสุดของ stack ไปให้ specified register
- rclReg : ใส่ค่าที่ได้มาจาก specified register ลงไปใน stack
- ping : ใส่ค่าลงไปบน stack โดยใส่ 0 ถ้า IP address ที่อยู่ใน parameter สามารถเข้าถึงได้ แต่ถ้าเข้าถึงไม่ได้ให้ใส่ 1

สำหรับส่วนของการเปลี่ยน control flow ใน procedure ของชุดคำสั่ง 2 กลุ่มนั้น จะใช้คำสั่งดังนี้

- jmp : การข้ามแบบไม่มีเงื่อนไข ไปยังบรรทัดที่ระบุไว้ใน argument
- jmpCD : control flow จะถูกย้ายไปยังบรรทัดที่ระบุใน argument ถ้า elements ที่อยู่บนสุดตรงตามเงื่อนไข
- ret : ใส่ argument value และทำการ return
- retCD : ใส่ argument value และ return ถ้าเงื่อนไขเป็นจริง

เมื่อเงื่อนไขเป็นจริง ส่วนของ return operation จะใส่ argument ลงบน stack และทำการสิ้นสุด procedure นั้น เงื่อนไขต่างๆได้แก่

- Equal (EQ)
- Not Equal (NE)
- Lower Than (LT)
- Lower or Equal (LE)
- Greater Than (GT)
- Greater or Equal (GE)

2.10.3 ตัวอย่างการทำงานบน SubManager

ตัวอย่างต่อไปนี้แสดงวิธีการคำนวณค่า error rate ของ packet ที่รับมาโดย interface ผลลัพธ์ที่ต้องการคือ error rate ที่ต่ำกว่าค่า threshold นั่นคือผลลัพธ์ที่เป็นศูนย์ ซึ่งคำนวณได้จากสูตรดังนี้

$$\text{ifInPkts} = \text{ifInUcastPkts} + \text{ifInNUcastPkts}$$

$$\text{errorRate} = \frac{(\text{ifInErrs}_{now} - \text{ifInErrs}_{old}) * 100}{\text{ifInPkts}_{now} - \text{ifInPkts}_{old}}$$

$$\text{result} = \begin{cases} 0 : \text{errorRate} \le \text{threshold} \\ \text{errorRate} : \text{errorRate} > \text{threshold} \end{cases}$$

จากสูตรจะเห็นว่าผลลัพธ์จะถูกคำนวณได้ก็ต่อเมื่อได้มีการคำนวณค่า ifInErrors_{old} และ ifInPkts_{old} ในอดีตมาก่อนแล้ว ขั้นตอนการดำเนินการนั้นเริ่มแรกจะเป็นการตรวจสอบสถานะการ execute ครั้งสุดท้าย ถ้าไม่ OK ค่า ifInErrors และ ifInPkts ก็จะถูกคำนวณและเก็บไว้ใช้ในการ

execute ครั้งต่อไป แต่ถ้า OK ก็จะทำการคำนวณค่า error rate และนำไปเปรียบเทียบกับค่า threshold ดังบรรทัดที่ 26 ในตารางที่ 2.7 ซึ่งแสดงตัวอย่างของ procedure ที่ใช้ในการทำงาน

ตารางที่ 2.7 ตัวอย่างของ procedure ที่ใช้ในการทำงาน

```
# Check status of last calculation
1    pushConst   1
2    getValV2    OID-ifStatus.23.1 Src Dst Cnt
3    jmpEQ       +8
# last calculation was NOT OK ; read inErrors and InPkts for next calculation
4    getValV1    OID-ifInErrors IP-addr
5    stoReg
6    getValV1    OID-ifInUcastPkts IP-addr
7    getValV1    OID-ifInNUcastPkts IP-addr
8    add
9    stoReg      2
10   ret         -1
# last calculation was OK ; calculate delta of inErrors
11   pushConst   Threshold[%]
12   recReg      1
13   getValV1    OID-ifInErrors IP-addr
14   stoReg      1
# if no error occurred, exit with 0 ; read inErrors and InPkts for next calculation
15   retEQ       0
# calculate delta inPkts
16   sub
17   pushConst   100
18   mul
19   recReg      2
20   getValV1    OID-ifInUcastPkts IP-addr
21   getValV1    OID-ifInNUcastPkts IP-addr
22   add
23   stoReg      2
24   sub
25   div
# if error rate is less than Threshold ; exit with 0, otherwise exit with error rate
26   retGE       0
```

2.11 CMU-SNMP toolkit distribution

CMU-SNMP toolkit distribution เป็น software toolkit ที่พัฒนาขึ้นด้วยภาษาซี สามารถนำมาใช้ได้โดยไม่เสียค่าใช้จ่าย ถึงแม้ว่าจะยังคงมี bugs และข้อจำกัดอยู่บ้าง แต่ก็จัดว่าเป็น tool ที่มีประสิทธิภาพตัวหนึ่ง

2.11.1 CMU-SNMP library

CMU-SNMP library มีหลายฟังก์ชันให้เรียกใช้งานโดย application และมีหลายฟังก์ชันที่ถูกเรียกใช้โดยฟังก์ชันด้วยกันเอง ซึ่งสามารถแบ่งได้ตามลักษณะหน้าที่การใช้งานได้ดังนี้

- ASN.1 Functions

ทำหน้าที่เข้ารหัส / ถอดรหัส packets ของ MIB และ SNMP โดยฟังก์ชันเหล่านี้จะรวมการเข้าและถอดรหัส ASN.1 ทุกชนิด เช่น INTEGER, DisplayString, OCTET_STRING, COUNTER เป็นต้น

ฟังก์ชันต่างๆดังกล่าวจะอยู่ในไฟล์ที่ชื่อ asn1.c ในการสร้าง SubManager นี้ผู้พัฒนาไม่ต้องทำการเรียกใช้ฟังก์ชันเหล่านี้โดยตรงเลย ทำให้ช่วยลดเวลาในการศึกษาเกี่ยวกับรูปแบบโครงสร้างทั้งหมดของ ASN.1 เพราะ code ดังกล่าวจะจัดการให้เอง

- Socket Functions

ฟังก์ชันกลุ่มนี้จะพบในใน ไฟล์ snmp_client.c, snmp_api.c และ snmp.c ทำหน้าที่เกี่ยวกับการเปิด socket สำหรับ SNMP, ส่ง request, รับ response และปิด session ซึ่งการพัฒนาโปรแกรม SubManager จะใช้ฟังก์ชันเหล่านี้เป็นส่วนมาก โดยฟังก์ชันเหล่านี้จะช่วยจัดการเกี่ยวกับ socket ในระดับล่าง ทำให้ผู้พัฒนาไม่ต้องไปยุ่งเกี่ยวกับโครงสร้างภายในของ SNMP

เนื่องจากฟังก์ชันเหล่านี้ถูกเรียกใช้โดย applications บ่อยมาก จึงจะนำเสนอโครงสร้าง และการนำไปใช้งานบางส่วน โดยฟังก์ชันที่จะนำเสนอต่อไปนี้จะต้องมีการ include ไฟล์ header ดังนี้

```
#include "snmp.h"
#include "snmp_impl.h"
#include "asn1.h"
#include "snmp_api.h"
```

```
void snmp_synch_setup (session)
struct snmp_session *session;
```

ฟังก์ชันนี้จะพบใน snmp_client.c เพื่อ setup callback ที่ใช้ในการตอบสนองต่อ SNMP request message

```
struct snmp_session  *snmp_open (session)
struct snmp_session  *session;
```

return pointer ให้กับ object ที่มีโครงสร้างแบบ snmp_session โดยฟังก์ชันนี้จะถูกเรียกใช้หลังจากที่มีการกำหนดค่าพารามิเตอร์ เช่น hostname, community ถูกต้องเรียบร้อยแล้ว เนื่องจากมันต้องใช้ข้อมูลเหล่านี้ในการสร้าง session ซึ่งก็คือการเปิด และเชื่อมต่อกับ UDP port ที่จำเป็น ถ้าเกิดข้อผิดพลาดขึ้นจะ return null และ snmp_errno จะจัดการกับ error code ตามความเหมาะสม ฟังก์ชันนี้จะถูกเรียกใช้ในไฟล์ snmp_api.c

```
struct snmp_pdu *snmp_pdu_create (command)
int command;
```

ถูกเรียกใช้ใน snmp_client.c เพื่อสร้าง SNMP protocol data unit ซึ่งใช้ในการส่ง SNMP message ออกไปในเครือข่าย โดยต้องมีการระบุชนิดของ pdu เช่น get, getnext ให้กับพารามิเตอร์ command ด้วย

```
int snmp_add_null_var (pdu, name, name_length)
struct snmp_pdu *pdu;
oid                 *name;
int                 name_length;
```

ถูกเรียกใน snmp_client.c ใช้สร้างตัวแปรที่เป็น null ให้กับชื่อที่ถูก request ลงในท้ายลิสต์ของ SNMP pdu เพื่อที่จะเพิ่มชื่อนั้นลงในตัวแปร สามารถเรียกใช้ฟังก์ชันนี้ได้หลายๆ ครั้ง เพื่อสร้าง SNMP request สำหรับหลายตัวแปร

```
int snmp_synch_response (ss, pdu, response)
struct snmp_session      *ss;
struct snmp_pdu *pdu;
struct snmp_pdu **response;
```

ฟังก์ชันนี้ใช้ข้อมูลจากพารามิเตอร์ ss ในการส่งข้อมูลออกไปยัง SNMP socket และรอ response เมื่อมีการตอบกลับมาก็จะทำการอ่านข้อมูลนั้นจาก socket มาจัดเก็บลงใน pdu response ที่ถูกเตรียมเอาไว้ แล้วทำการ return กลับไปพร้อมกับสถานะ แต่ถ้าพบว่ามีข้อผิดพลาดเกิดขึ้นก็จะ return error สำหรับข้อมูลที่บรรจุใน pdu ที่ส่งกลับไปนั้นอาจจะเป็น SNMP packet ทั้งหมดที่จะนำไปแสดงผล, ข้อมูลที่เกี่ยวกับการจัดการ หรืออื่นๆ ฟังก์ชันนี้จะพบในไฟล์ snmp_client.c

void snmp_free_pdu (pdu)

struct snmp_pdu *pdu;

ฟังก์ชันนี้ใช้ใน snmp_api.c เพื่อทำการคืนพื้นที่ที่จองไว้สำหรับข้อมูลใน pdu ที่ถูกส่งมาเป็น argument ยังฟังก์ชันนี้

int snmp_close (session)

struct snmp_session *session;

ฟังก์ชันนี้ใช้ใน snmp_api.c เพื่อปิด input session โดยจะทำการคืนพื้นที่ของข้อมูล ยกเลิกคำร้องต่างๆ ที่ยังคงค้างอยู่ และปิด socket ทั้งหมดที่จองไว้สำหรับการจัดการ session แล้วทำการ return ค่า 1 แต่ถ้ามีข้อผิดพลาดเกิดขึ้นจะ return ค่า 0

- MIB Functions

CMU-SNMP library จะทำงานกับตัวแปร MIB ที่ได้นิยามไว้ใน MIB ไฟล์เท่านั้น ถ้าไม่อย่างนั้นค่าที่ส่งกลับจากคำร้องขอต่างๆ จะไม่ถูกแปลงให้อยู่ในรูป ASN.1 ซึ่งหลักการทำงานคือ code จะทำการตีความไฟล์ที่นิยามเกี่ยวกับ ASN.1 MIB แล้วสร้าง leaf objects ใน linked list tree เพื่อเก็บไว้ใช้ซึ่ง tree นี้จะถูกจัดเก็บอยู่ใน memory และหากต้องการทำงานกับ MIBs หลายๆ ตัวในเวลาเดียวกัน จะต้องทำการรวมให้อยู่ในไฟล์เดียวกันก่อนที่จะเริ่มต้นการทำงานของ application

Library นี้มีฟังก์ชันเกี่ยวกับตัวแปร MIB เป็นจำนวนมากในไฟล์ mib.c ที่สำคัญคือฟังก์ชันที่ใช้แปลงค่า MIB object ซึ่งอยู่ในรูปของ array of integer ให้อยู่ในรูปแบบที่ใกล้เคียงกับภาษาที่คนอ่านแล้วเข้าใจได้ง่าย

2.11.2 CMU-SNMP implementation

CMU-SNMP toolkit distribution พัฒนาโดยนำโพรโตคอล SNMP มา implement อาศัยหลักการของ client/server กล่าวคือเครื่องที่เป็น agent (server) จะทำการรัน daemon ที่ชื่อว่า

snmpd เพื่อเอาไว้รอฟังคำสั่ง get-request และ set-request จาก manager ส่วนเครื่องที่เป็น manager (client) จะรัน application สำหรับติดต่อกับ agent โดยใน manager จะมีการเตรียมคำสั่งที่ใช้สำหรับ execute get-request หรือ set-request ใน shell-level ไว้ให้ user ซึ่งใน CMU-SNMP implementaion คำสั่งเหล่านี้ได้แก่

- snmpget ใช้ get request ในการดึงค่า MIB object ของ entity ที่ต้องการ โดยสามารถส่งค่า object identifiers เป็น argument ได้มากกว่า 1 ตัว
- snmptest ใช้ get, getnext หรือ set request ในการติดต่อ หรือจัดการกับ agent
- snmpwalk ใช้ getnext request ในการท่องผ่านเข้าไปใน MIB ของ agent โดยเริ่มจากค่าใดค่าหนึ่งแล้วไล่ไปตามลำดับ
- snmpstatus ดึงค่าเกี่ยวกับสถิติที่สำคัญจาก agent เช่น IP address, text description, uptime, ผลรวมของ packets ที่ผ่านเข้าออกบริเวณ interface, IP packets in and out
- snmpnetstat ทำงานคล้ายกับ netstat แต่จะดึงค่าต่างๆ ผ่านทาง MIB ของ agent นั้นๆ
- snmptrap ใช้ในการส่ง SNMP trap message ไปยัง host
- snmptrapd ทำการรับ และบันทึก SNMP trap message ลง logs

2.11.3 CMU-SNMP toolkit : SNMPv1 Configuration

การใช้งาน SNMPv1 เกือบจะไม่ต้องมีการปรับแต่งใดๆ เลย เนื่องจากไฟล์จำพวก *.conf (acl.conf, context.conf, party.conf, snmpd.conf, view.conf) ที่เก็บอยู่ในไดเร็กทอรี /etc นั้น มีเพียงแต่ไฟล์ snmpd.conf ที่จะถูกอ่านไปใช้ในการทำงานสำหรับ SNMPv1

ถ้าหากต้องการที่จะเปลี่ยนแปลงเกี่ยวกับ community name, system contact, system location, system name หรือต้องการลบ interfaces ที่ไม่ได้ใช้งานทิ้งไป สามารถทำได้โดยดูรายละเอียดจาก snmpd.conf และสำหรับ community name ที่อนุญาตให้ใช้ได้มีเพียงสองตัวเท่านั้นคือ public และ private (หรืออาจเป็นชื่อใดๆ ก็ได้ที่เราตั้งให้สำหรับ private community)

2.11.3.1 ตัวอย่างการใช้งานคำสั่งสำหรับ SNMPv1

ก่อนเริ่มทดลองใช้งานจะต้องทำการปรับแต่งเครื่องให้เป็นทั้ง agent (run snmpd) และ manager (*snmpget*, etc) เสียก่อน

- คำสั่ง *snmpget*

คำสั่งนี้ช่วยให้เราสามารถดึงค่าที่เป็นเดี่ยวของตัวแปร MIB มาจาก agent ได้ เมื่อเราทราบ MIB object instance ที่แน่นอน โดยมีโครงสร้างของคำสั่งดังนี้

snmpget -v 1 hostname community_name MIB_object_instance

```
Shell - root@meaw:~ - Konsole
File Sessions Settings Help
[root@meaw root]# snmpget -v 1 localhost public system.syscontact.0
system.sysContact.0 = "s3067008@kmitl.ac.th"
[root@meaw root]# █
```

รูปที่ 2.9 ตัวอย่างการใช้งานคำสั่ง snmpget ของ SNMPv1

ยกตัวอย่างดังรูปที่ 2.9 คำสั่ง *'snmpget -v 1 localhost public system.syscontact.0'* จะทำการดึงข้อมูลเกี่ยวกับ system contact จากเครื่องที่กำลังใช้งานอยู่โดยใช้ public community name แต่ถ้าหากเรามี private community name ที่ชื่อ neon (กำหนดไว้ใน /etc/snmpd) เราก็สามารถดึงข้อมูลดังกล่าวได้ด้วยคำสั่งนี้

snmpget -v 1 localhost neon system.syscontact.0

- คำสั่ง *snmpwalk*

คำสั่งนี้ช่วยให้เราสามารถดึงค่าของตัวแปร MIB มาจาก agent ได้โดยไม่ต้องมีการระบุ instance ที่แน่นอน มีโครงสร้างของคำสั่งดังนี้

snmpwalk -v 1 hostname community_name MIB_object_type

ยกตัวอย่างดังรูปที่ 2.10 คำสั่ง *'snmpwalk -v 1 localhost public system.syscontact'* จะทำการดึงข้อมูลเกี่ยวกับ system contact จากเครื่องที่กำลังใช้งานอยู่โดยใช้ public community name แต่ถ้าหากเรามี private community name ที่ชื่อ neon (กำหนดไว้ใน /etc/snmpd) เราก็สามารถดึงข้อมูลดังกล่าวได้ด้วยคำสั่งนี้

snmpwalk -v 1 localhost neon system.syscontact

สังเกตได้ว่าแตกต่างกับการใช้ *snmpget* คือไม่ต้องมีการระบุ instance identifier ('.0') นอกจากนี้เรายังสามารถใช้ *snmpwalk* ในการดึงค่าของตัวแปรได้มากกว่า 1 ค่า เช่น *'snmpwalk -v 1 localhost neon system'* คำสั่งดังกล่าวจะให้ผลลัพธ์เป็นค่าของตัวแปรทุกตัวใน MIB ที่อยู่ภายใต้โหนด system

```
File Sessions Settings Help
[root@meaw root]# snmpwalk -v 1 localhost public system.syscontact
system.sysContact.0 = "s30670088@kmitl.ac.th"
[root@meaw root]# snmpwalk -v 1 localhost public system
system.sysDescr.0 = "Linux version 2.4.7-10 (bhcompile@stripples.devel.redhat.com) (gcc version 2.96 20000731
(Red Hat Linux 7.1 2.96-98)) #1 Thu Sep 6 17:27:27 EDT 2001"
system.sysObjectID.0 = OID: enterprises.1701.1.1
system.sysUpTime.0 = Timeticks: (17968) 0:02:59
system.sysContact.0 = "s30670088@kmitl.ac.th"
system.sysName.0 = "meaw" Hex: 6D 65 61 77
system.sysLocation.0 = "Home" Hex: 48 6F 6D 65
system.sysServices.0 = 72
[root@meaw root]#
```

รูปที่ 2.10 ตัวอย่างการใช้งานคำสั่ง snmpwalk ของ SNMPv1

- คำสั่ง *snmpset*

คำสั่งนี้ช่วยให้เราสามารถจัดการกับค่าของตัวแปร MIB ที่เก็บอยู่ใน agent ได้ แต่จะทำได้ครั้งละ 1 ตัวแปรเท่านั้น มีโครงสร้างของคำสั่งดังนี้

snmpset -v 1 hostname community_name MIB_object_instance type value

```
File Sessions Settings Help
[root@meaw root]# snmpset -v 1 localhost neon system.syscontact.0 s niramol@kmitl.ac.
th
system.sysContact.0 = "niramol@kmitl.ac.th"
[root@meaw root]#
```

รูปที่ 2.11 ตัวอย่างการใช้งานคำสั่ง snmpset ของ SNMPv1

ยกตัวอย่างรูปที่ 2.11 คำสั่ง '*snmpset -v 1 localhost neon system.syscontact.0 s niramol@kmitl.ac.th*' จะเป็นการกำหนดชื่อของ system contact ให้เป็นชนิดข้อความ (type s = string) ซึ่งให้ชื่อว่า niramol@kmitl.ac.th ผ่านทาง private community name คือ neon โดยการเปลี่ยนแปลงที่เกิดขึ้นนี้จะถูกบันทึกลงใน /ect/snmpd.conf (หมายเหตุ : คำสั่ง *snmpset* นั้นจะอนุญาตให้ใช้กับ private community name เท่านั้น)

- คำสั่ง *snmptrapd*

CMU SNMP ได้เตรียม daemon ที่ชื่อ snmptrapd ซึ่งทำให้เครื่องที่เป็น manager สามารถรับ traps และเก็บบันทึกลงใน syslogd แต่ก่อนหน้านั้นเครื่องที่เป็น agent จะต้องทำการรัน snmptrapd ซึ่งมีรูปแบบคำสั่งดังนี้

snmptrapd [-v #] [-P #] [-p] [-s] [-d]

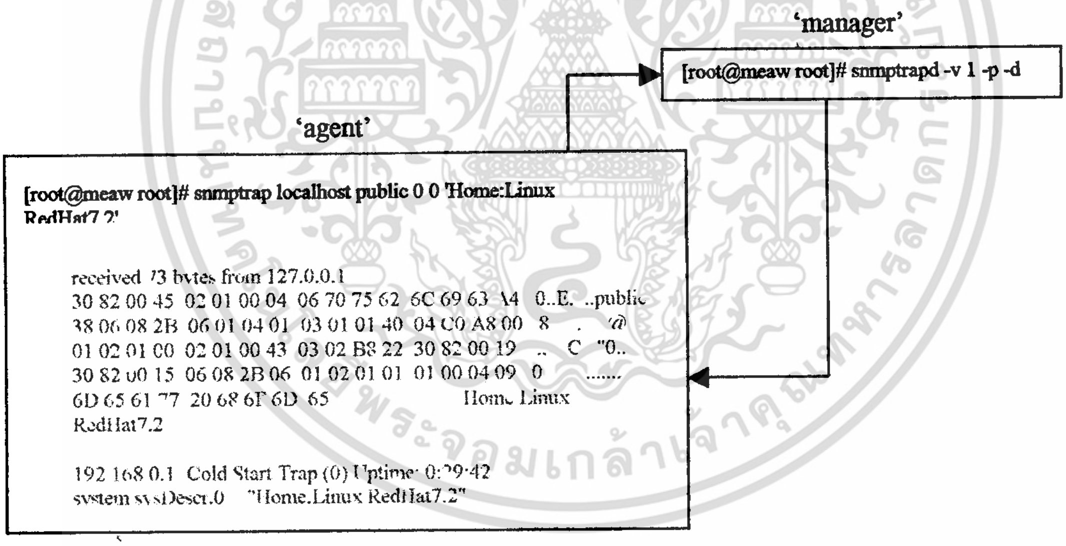
ยกตัวอย่างเช่นคำสั่ง '*snmptrapd -v 1 -p –d*' จะเป็นการสั่งรัน snmptrapd สำหรับรับ traps ที่สนับสนุนการทำงานของ SNMPv1 โดยให้แสดงผลลัพธ์ และข้อมูลที่ได้จากการ dump packet ออกทางหน้าจอ

- คำสั่ง *snmptrap*

คำสั่งนี้ช่วยให้ agent สามารถส่งข้อมูลเหตุการณ์ต่างๆ ที่ถูกกำหนดไว้ ไปยัง manager ได้ มีโครงสร้างของคำสั่งดังนี้

snmptrap host community trap-type specific-type device-description [-a agent-addr]

ยกตัวอย่างเช่นคำสั่ง 'snmptrap localhost public 0 0 "Home:Linux RedHat7.2".' เป็นการส่ง trap ไปยัง localhost เพื่อแจ้งว่าได้ทำการ reinitial หลังจากที่ได้มีการปรับเปลี่ยนค่าบางอย่างเรียบร้อยแล้ว โดยภาพรวมของการใช้งาน snmptrapd และคำสั่ง snmptrap แสดงได้ดังรูปที่ 2.12



รูปที่ 2.12 snmptrapd และคำสั่ง snmptrap ของ SNMPv1

2.11.4 CMU-SNMP toolkit : SNMPv2 Configuration

CMU SNMP package ได้บรรจุ configuration files ซึ่งเกี่ยวข้องกับ SNMPv2 ไว้ดังนี้

- party.conf

กำหนด IP address privacy keys และ authentication keys ของ parties ในไฟล์ /etc/party.conf ดังตัวอย่างต่อไปนี้

robin .1.3.6.1.6.3.3.1.3.192.168.0.1.1

snmpUDPDomain 192.168.0.1 161

noAuth noPriv

300 484

29F660EA

00000000000000000000000000000000 Null

00000000000000000000000000000000 Null

batman .1.3.6.1.6.3.3.1.3.192.168.0.1.2

snmpUDPDomain 0.0.0.0 0

noAuth noPriv

300 484

29F660EA

00000000000000000000000000000000 Null

00000000000000000000000000000000 Null

เป็นการกำหนด 2 parties ใหม่ขึ้นมาได้แก่ batman (manager) และ robin (agent) ซึ่งจะติดต่อสื่อสารกันแบบไม่มี authentication (noAuth) และไม่มี privacy (noPriv) โดย robin และ batman จะเป็น party หมายเลข 1 และ 2 ตามลำดับบน local system (ดูจากเลขตัวสุดท้ายของ object IDs) สำหรับชุดของตัวเลข 192.168.0.1 เป็นการระบุตำแหน่งว่า party นั้นอยู่บน local host

หมายเลขของเส้นทางที่ถูกกำหนดใน parties นั้น (.1.3.6.1.6.3.3.1.3) สามารถเขียนเป็นชื่อเต็มๆ ได้ดังนี้

.iso.org.dod.internet.snmpV2.snmpModules.partyMIB.partyAdmin.initialPartyID

- view.conf

กำหนดเกี่ยวกับ MIB subtrees ในไฟล์ /etc/view.conf ดังตัวอย่างต่อไปนี้ เป็นการกำหนด view ขึ้นมาใหม่ (view หมายเลข 3)

3 .iso.org.dod.internet.mgmt included Null

- context.conf

กำหนด MIB View หรือ Proxy Relationship สำหรับแต่ละ context ในไฟล์ /etc/context.conf ดังตัวอย่างต่อไปนี้

batcave	.1.3.6.1.6.3.3.1.4.192.168.0.1.3	
3	Null	currentTime
0	0	0

กำหนดให้ batcave เป็น context ลำดับที่ 3 บน local machine (ดูจากเลขตัวสุดท้ายของ object ID) และ context นี้อ้างอิงถึง MIB view หมายเลข 3 ส่วนในบรรทัดสุดท้ายที่ใส่เลข 0 ทั้งสามตัวนั้นเป็นการระบุว่าไม่มี proxy relationship

หมายเลขของเส้นทางที่ถูกกำหนดใน parties นั้น (.1.3.6.1.6.3.3.1.4) สามารถเขียนเป็นชื่อเต็มๆ ได้ดังนี้

.iso.org.dod.internet.snmpV2.snmpModules.partyMIB.partyAdmin.initialContextID

- acl.conf

กำหนดเกี่ยวกับ context และ privileges ของ parties ที่จะติดต่อสื่อสารกัน ในไฟล์ /etc/acl.conf ดังตัวอย่างนี้

robin	batmat	batcave	GNB
batman	robin	batcave	RU

บรรทัดแรกเป็นการกำหนดว่า batman (source party) สามารถส่งคำร้องขอประเภท get (G), getnext (N) และ getbluk (B) เกี่ยวกับ batcave จาก robin (target party) ได้

ส่วนบรรทัดที่สอง เป็นการกำหนดว่า robin (target party) สามารถจัดเตรียมข้อมูล (R = get Response) จาก batcave เพื่อส่งให้กับ batman (source party) ตามคำร้องขอ และยังสามารถส่ง trap ไปให้กับ batman ได้อีกด้วย (U = trap2)

2.11.4.1 ตัวอย่างการใช้งานคำสั่งสำหรับ SNMPv2

- คำสั่ง *snmpget*

ช่วยให้เราสามารถดึงค่าที่เป็นเดี่ยวของตัวแปร MIB มาจาก agent ได้ เมื่อเราทราบ MIB object instance ที่แน่นอน โดยมีโครงสร้างของคำสั่งดังนี้

snmpget [-v 2] hostname src_party dest_party context MIB_object_instance

File Sessions Settings Help

```
[root@meaw root]# snmpget localhost batman robin batcave system.sysdescr.0
system.sysDescr.0 = "Linux version 2.4.7-10 (bhcompile@stripples.devel.redhat.com) (gcc version 2.96 20000731
(Red Hat Linux 7.1 2.96-98)) #1 Thu Sep 6 17:27:27 EDT 2001"
[root@meaw root]# 
```

รูปที่ 2.13 ตัวอย่างการใช้งานคำสั่ง snmpget ของ SNMPv2

ยกตัวอย่างดังรูปที่ 2.13 คำสั่ง '*snmpget localhost batman robin batcave system.sysdescr.0*' จะหมายถึงการดึงรายละเอียดของ system จาก local machine โดยใช้ batcave context

- snmptest sample program

โปรแกรม snmptest.c เป็นโปรแกรมแบบ command line-driven ที่ใช้สำหรับสร้าง SNMP PDUs ชนิดต่างๆ ซึ่งมีโครงสร้างของคำสั่งดังนี้

snmptest -v 1 [options] hostname community

หรือ *snmptest [-v 2] [options] hostname noAuth*

หรือ *snmptest [-v 2] [options] hostname srcParty dstParty context*

โดยที่ *options* คือ

- -d : dump incoming and outgoing packets
- -p portnum : specify destination port number
- -t timeout : specify timeout
- -r retry : specify retry
- -c clock : specify clock

เมื่อเริ่มต้นรันโปรแกรมจะปรากฏ *Variable :* prompt ซึ่งเราสามารถระบุ object ID หรือคำสั่ง snmptest ลงไปก็ได้ โดยคำสั่งต่างๆ มีดังนี้

- $B : specify BULK_TRANSFER_REQUEST PDU
- $D : toggle packet dumping on / off
- $G : specify GET_REQUEST PDU
- $I : specify INFORM_REQUEST PDU
- $N : specify GET_NEXT_REQUEST PDU

- $Q : quit snmptest
- $S : specify SET_REQUEST PDU
- $T : specify SNMP2_TRAP PDU

ถ้าใช้คำสั่ง $I, $S หรือ $T จะปรากฏ prompt ขึ้นสองครั้ง โดยครั้งแรกจะให้ระบุชนิดของข้อมูลดังนี้ *Type [i | s | x | d | n | o | t | a] :*

- i : INTEGER
- s : STRING
- x : STRING
- d : STRING
- n : NULLOBJ
- o : OBJID
- t : TIMETICKS
- a : IPADDRESS

prompt ตัวที่สองคือ *Value :* เป็นการกำหนดค่าที่ต้องการใส่ลงใน PDU และในขั้นตอนสุดท้ายคือการส่ง PDU ออกไป ทำได้โดยการกด ENTER ที่ *Variable :* prompt

- SNMPv2_TRAP example

ก่อนที่จะทดลองส่ง trap เราจะต้องทำการรัน snmptrapd daemon เสียก่อนด้วยคำสั่ง *"snmptrapd -p -d"* โดย *-p* flag เป็นการบอกว่าให้ผลลัพธ์ที่ได้ปรากฏที่ stdout ส่วน *-d* flag จะเป็นการระบุว่า raw packet ที่ได้มานั้นจะถูก dump ออกมาแสดงผลด้วย หลังจากนั้นทดลองใช้คำสั่งดังนี้

snmptest –p 162 localhost batman robin batcave

–p 162 flag เป็นการบอกว่าควรส่งคำร้องไปที่ port หมายเลข 162 ซึ่งเป็น port ที่เตรียมไว้สำหรับรอรับ trap และหลังจากคำสั่งข้างต้นจะปรากฏ prompt ต่างๆขึ้นมาให้เราใส่ค่า ดังรูปที่ 2.14

```
File  Sessions  Settings  Help
[root@meaw root]# snmptest -p 162 localhost batman robin batcave
Variable: $T
Request type is SNMPv2 Trap Request
(Are you sending to the right port?)
Variable: system.sysuptime.0
Type [i|s|x|d|n|o|t|a]: t
Value: 6000
Variable: .1.3.6.1.6.3.1.1.4.1.0
Type [i|s|x|d|n|o|t|a]: o
Value: .1.3.6.1.6.3.1.1.5.4
Variable: interfaces.iftable.ifentry.ifindex.1
Type [i|s|x|d|n|o|t|a]: i
Value: 1
Variable:
Variable: █
```

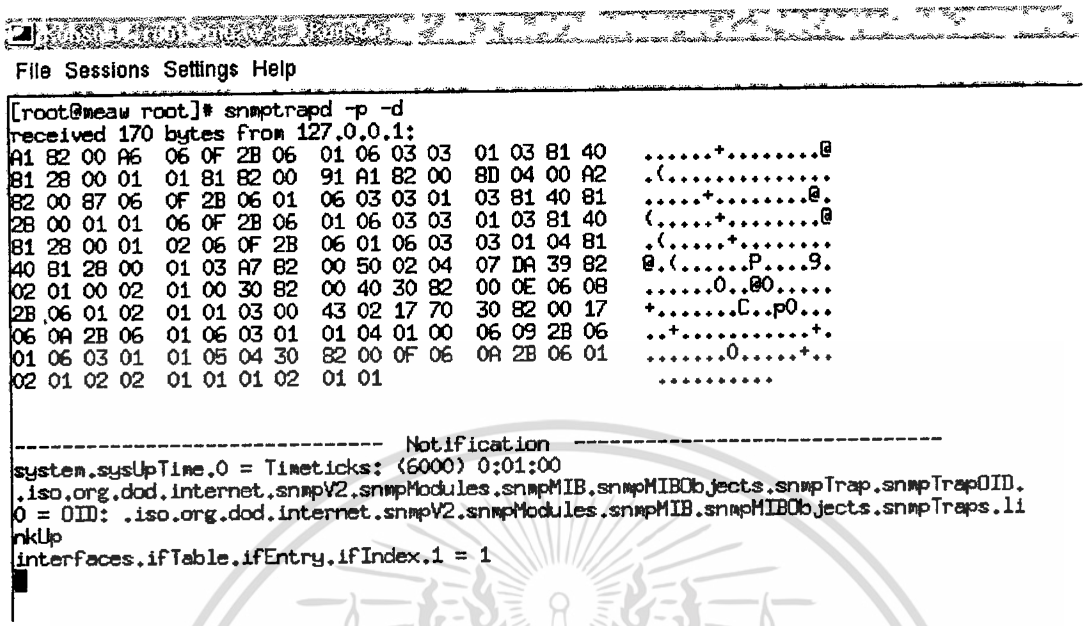
รูปที่ 2.14 ตัวอย่างการใช้ snmptest เพื่อส่ง SNMPv2-TRAP

$T เป็นการบอกว่าเราต้องการจะส่ง SNMPv2 trap เมื่อส่งคำสั่งนี้ไปแล้ว snmptest จะมีข้อความเตือนว่าเราต้องการส่งไปยัง port นี้แน่นอนหรือไม่ (ในที่นี้คือ port 162 ซึ่งถูกต้อง) ถัดจากนั้นจะเป็นการระบุส่วนต่างๆ 3 ส่วน(ตามลำดับ) เพื่อเป็นการส่ง linkUp trap ดังนี้

1. system.sysUptime
2. snmpTrapOID
3. ifIndex

เราสามารถใช้ system.sysUptime.0 แทนที่ชื่อ object ID แบบเต็มได้ เนื่องจากเป็นโหนดลูกของ " .iso.org.dod.internet.mgmt.mib-2 " แต่ snmpTrapOID เป็นโหนดลูกของ " .iso.org.dod.internet.snmpV2" ดังนั้นจึงต้องระบุเส้นทางแบบเต็ม ซึ่งจะเขียนเป็นตัวเลขดังตัวอย่างเพื่อความกะทัดรัด หรือใส่เป็นชื่อ "snmpModules.snmpMIB.snmpMIBObjects.snmpTrap.snmpTrapOID.0" ก็ได้ โดยในส่วนที่ให้ใส่ค่าของ snmpTrapOID นั้นคือการระบุชนิดของ trap ที่จะส่ง (ในที่นี้คือ linkUp) สำหรับตัวแปรสุดท้ายคือการระบุ interface (link) ที่กลับมาเริ่มต้นทำงานอีกครั้ง (came up)

หลังจากทำการส่ง trap ออกไปแล้ว ทางฝั่งของ snmptrapd จะนำ trap ที่ได้รับมาแสดงผลดังรูปที่ 2.15



```
File Sessions Settings Help
[root@meaw root]# snmptrapd -p -d
received 170 bytes from 127.0.0.1:
A1 82 00 A6  06 0F 2B 06  01 06 03 03  01 03 81 40   ......+.........@
81 28 00 01  01 81 82 00  91 A1 82 00  8D 04 00 A2   .(..............
82 00 87 06  0F 2B 06 01  06 03 03 01  03 81 40 81   .....+........@.
2B 00 01 01  06 0F 2B 06  01 06 03 03  01 03 81 40   (.....+........@
81 28 00 01  02 06 0F 2B  06 01 06 03  03 01 04 81   .(.....+........
40 81 28 00  01 03 A7 82  00 50 02 04  07 DA 39 82   @.(......P....9.
02 01 00 02  01 00 30 82  00 40 30 82  00 0E 06 08   ......0..@0.....
2B 06 01 02  01 01 03 00  43 02 17 70  30 82 00 17   +.......C..p0...
06 0A 2B 06  01 06 03 01  01 04 01 00  06 09 2B 06   ..+...........+.
01 06 03 01  01 05 04 30  82 00 0F 06  0A 2B 06 01   .......0.....+..
02 01 02 02  01 01 01 02  01 01                       ..........

------------------------------ Notification ------------------------------
system.sysUpTime.0 = Timeticks: (6000) 0:01:00
.iso.org.dod.internet.snmpV2.snmpModules.snmpMIB.snmpMIBObjects.snmpTrap.snmpTrapOID.
0 = OID: .iso.org.dod.internet.snmpV2.snmpModules.snmpMIB.snmpMIBObjects.snmpTraps.li
nkUp
interfaces.ifTable.ifEntry.ifIndex.1 = 1
```

รูปที่ 2.15 ผลลัพธ์ที่ได้จาก SNMPv2-TRAP

จากการศึกษารูปแบบโครงสร้างการจัดการเครือข่าย ทำให้ทราบปัญหาที่เกิดขึ้นว่าเมื่อมีจำนวน agents มากขึ้นทำให้ manager ต้องทำการหนัก การแตกโครงสร้างให้เป็นลำดับชั้นจึงเป็นวิธีแก้ไขปัญหาทางหนึ่ง ซึ่งแนวทางนี้ได้ถูกนำเสนอในบทความ Hierarchical Network Management A Concept and its Prototype in SNMPv2 (Manfred R.Siegl และ Georg Trausmuth : 1996) รวมทั้งมีการพัฒนาโปรแกรม SubManager ขึ้น แต่อย่างไรก็ตามโปรแกรมนี้ยังไม่สามารถจัดการกับ traps ที่รับเข้ามาได้ ดังนั้นโครงงานนี้จึงนำโปรแกรม SubManager มาพัฒนาเพิ่มเติมในส่วนที่เกี่ยวข้องกับการจัดการ traps โดยอาศัยทฤษฎีต่างๆ ดังที่กล่าวมาข้างต้น ซึ่งรายละเอียดในการพัฒนาจะกล่าวถึงในบทถัดไป

บทที่ 3
กรอบการพัฒนาระบบงาน

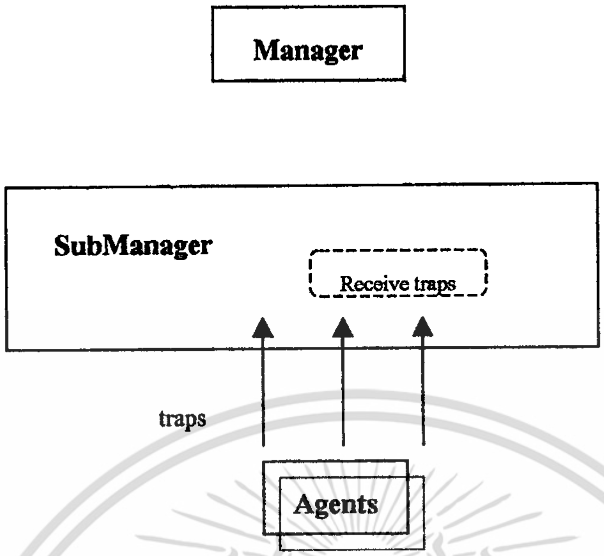
เนื้อหาในบทก่อนหน้านี้ได้อธิบายถึงที่มาของโครงงาน และอธิบายถึงหลักการทำงานของโปรแกรม SubManager ต้นแบบมาบ้างแล้ว สำหรับในบทนี้จะกล่าวถึงเครื่องมือที่จะใช้ในการพัฒนาระบบงาน และการพัฒนาโปรแกรม SubManager ในส่วนที่เกี่ยวข้องกับการจัดการ trap ที่รองรับเฉพาะ SNMPv1 เท่านั้น โดยจะแสดงให้เห็นถึงภาพรวมของการทำงานเกี่ยวกับ trap ของโปรแกรมต้นแบบ ฟังก์ชันการกรอง traps ที่จะทำการพัฒนาเพิ่มเติม พร้อมทั้งอธิบายหลักการทำงานของส่วนต่างๆ ด้วย

3.1 เครื่องมือที่ใช้ในการพัฒนาระบบงาน

- CMU-SNMP toolkit distribution : ใช้ library ซึ่งประกอบไปด้วยฟังก์ชันที่จำเป็นในการพัฒนาระบบงาน
- Emacs : ใช้ในการเขียนโปรแกรม
- Microsoft Word : ใช้ในการทำเอกสารประกอบโครงงาน

3.2 การทำงานเกี่ยวกับ traps ของโปรแกรม SubManager ต้นแบบ

ดังที่ได้กล่าวมาแล้วในบทที่ 2 ว่าการทำงานที่เกี่ยวข้องกับ trap ในการจัดการเครือข่ายนั้นจะต้องประกอบไปด้วยผู้ส่ง trap ซึ่งคือ agent และผู้รับ trap ซึ่งก็คือ manager แต่ตามทฤษฎีของ SubManager แล้ว โปรแกรมถูกพัฒนาขึ้นเพื่อให้เป็นตัวกลางระหว่าง manager และ agent ดังนั้นจึงจำเป็นจะต้องมีความสามารถทั้งการรับ trap และส่ง trap แต่จากภาพรวมการทำงานเกี่ยวกับ traps ของโปรแกรม SubManager ต้นแบบ ดังรูปที่ 3.1 จะเห็นได้ว่าโปรแกรมทำได้แค่เพียงรับ trap มาเก็บเอาไว้เท่านั้น ซึ่งจะสามารถอธิบายการทำงานโดยแยกออกเป็นสองส่วนดังนี้



รูปที่ 3.1 ภาพรวมการทำงานเกี่ยวกับ traps ของโปรแกรม SubManager ต้นแบบ

3.2.1 การรับ traps จาก agent

โปรแกรม SubManager ต้นแบบนั้น ได้ใช้ฟังก์ชันการจัดการ trap ที่มีอยู่ใน CMU-SNMP toolkit distribution ที่ชื่อว่า snmptrapd ทำให้เครื่องที่เป็น manager สามารถรับ traps เก็บบันทึกลงใน syslogd และแสดงผลลัพธ์ทางหน้าจอได้ ซึ่งทำงานโดยอาศัยคำสั่งที่มีรูปแบบดังนี้

snmptrapd [-v #] [-P #] [-p] [-s] [-d]

- *-v* หมายถึงเวอร์ชันของโพโตคอล SNMP
- *-P* หมายถึงหมายเลขพอร์ต ซึ่งกำหนดค่า default ให้ใช้หมายเลข 162 ตามหลักการของโพโตคอล SNMP
- *-p* หมายถึงให้แสดงข้อมูล trap ที่ได้รับเข้ามาทาง standard output
- *-s* หมายถึงให้บันทึกข้อมูล trap ลงใน syslogd
- *-d* หมายถึงให้มีการ dump packet ด้วย

ยกตัวอย่างเช่นคำสั่ง '*snmptrapd -v 1 -p –d*' จะเป็นการสั่งรัน snmptrapd สำหรับรอรับ SNMPv1 traps โดยให้แสดงผลลัพธ์ และข้อมูลที่ได้จากการ dump packet ออกทางหน้าจอ

ผลลัพธ์จากการตีความ trap ที่ได้รับเข้ามาของส่วน Receive traps นั้นจะมีรูปแบบดังนี้

address : trap-type (specific-type) : uptime : system-descritption

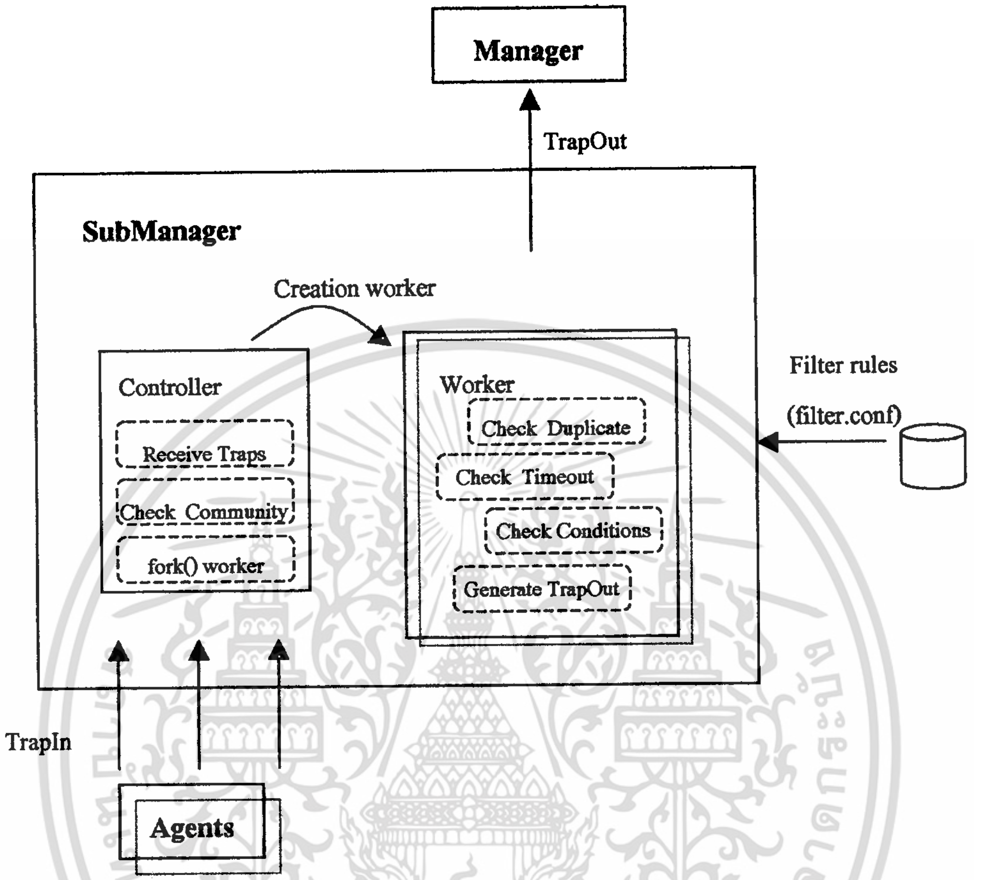
- *address* **คือส่วนที่ระบุเลข IP ของ agent ที่ทำการส่ง trap**
- *trap-type* **คือตัวเลขที่ระบุชนิดของ SNMPv1 trap ซึ่งแบ่งออกเป็น 7 ประเภท ดังที่เคยแสดงไว้แล้วในตารางที่ 2.2**
- *specific-type* **คือส่วนที่ผู้ใช้กำหนดขึ้นมาเพื่อใช้เป็นข้อมูลเฉพาะในการทำงาน**
- *uptime* **คือระยะเวลาตั้งแต่ที่ agent เริ่มทำงาน จนถึงที่ได้ทำการส่ง trap ไปให้ manager**
- *system -description* **คือข้อความแสดงรายละเอียดของเครื่องที่เป็น agent**

3.2.2 การส่ง traps ให้กับ manager

โปรแกรม SubManager ต้นแบบนั้น ยังไม่มีฟังก์ชันใดๆ ที่จะจัดการกับ traps ที่ได้รับมา และยังไม่สามารถส่งต่อ traps เหล่านั้นไปยัง manager ได้

3.3 การพัฒนาฟังก์ชันเพิ่มเติมในส่วนการจัดการ traps

โครงงานนี้ได้นำส่วน snmptrapd จากโปรแกรม SubManager ต้นแบบ ซึ่งแต่เดิมมีความสามารถแค่รับ trap เข้ามาเก็บไว้ มาทำการเพิ่มเติมฟังก์ชันต่างๆ ในการกรอง traps รวมทั้งพัฒนาความสามารถในการส่ง trap ไปยัง manager ซึ่งจะเป็นการเพิ่มประสิทธิภาพในการจัดการ trap ให้กับโปรแกรมดังกล่าว และจะเปลี่ยนชื่อคำสั่งจาก snmptrapd เป็น submgrtrapd เพื่อไม่ให้เกิดความสับสน เนื่องจากการทดสอบในบทที่ 5 จำเป็นต้องอ้างอิงถึง snmptrapd ในส่วนที่เป็น manager



รูปที่ 3.2 ภาพรวมการทำงานเกี่ยวกับ trap ที่พัฒนาเพิ่มเติม

ภาพการทำงานโดยรวมของ SubManager ในส่วนที่เกี่ยวข้องกับ trap จะเป็นดังรูปที่ 3.2 กล่าวคือเมื่อ agent ต้องการแจ้งเหตุการณ์บางอย่างก็จะทำการส่ง trap มายัง SubManager โดยในโครงงานนี้จะเรียก SNMPv1 trap ที่ได้รับมาจาก agent ว่า TrapIn โปรแกรมจะนำ TrapIn นี้มาประมวลผลตามฟังก์ชันต่างๆ ที่ได้พัฒนาเพิ่มเติมขึ้น ก่อนที่จะทำการส่งต่อ หรือสรุปเป็นเหตุการณ์ส่งไปยัง manager ซึ่งในโครงงานนี้จะเรียก trap ที่ส่งไปยัง manager ว่า TrapOut ภาพรวมดังกล่าวสามารถอธิบายโดยแยกออกเป็นสองส่วนดังนี้

3.3.1 การรับ trap จาก agent

เมื่อเริ่มต้นทำงาน submgrtrapd จะรอรับ TrapIn ที่ส่งมาจาก agent เมื่อมี TrapIn เข้ามาก็จะมีการตรวจสอบสิทธิ์ในการส่ง ถ้าพบว่าถูกต้องก็จะสร้าง Worker ให้กับแต่ละ TrapIn ที่ได้รับ

มา เพื่อทำงานตามฟังก์ชันต่างๆ โดยอาศัยข้อกำหนดจากคอนฟิกไฟล์ที่ชื่อ filter.conf ดังรูปที่ 3.2 องค์ประกอบภายในของ submgrtrapd ประกอบไปด้วย 3 ส่วนหลักคือ

- Filter rules : เป็นส่วนที่กำหนดเงื่อนไขในการทำงานของ submgrtrapd จะถูกจัดเก็บอยู่ในรูปแบบของคอนฟิกไฟล์ที่ชื่อ filter.conf ประกอบไปด้วย community name ข้อกำหนดเกี่ยวกับการกรองซ้ำ ช่วงเวลา timeout และเงื่อนไขที่จะทำให้เกิดสถานการณ์ต่างๆ ซึ่งจะทำการอธิบายอย่างละเอียดในส่วนของการออกแบบระบบในบทถัดไป

- Controller : เป็นส่วนที่ติดต่อกับ agent มีหน้าที่รับ TrapIn เข้ามาแล้วทำงานตามฟังก์ชันย่อยดังนี้
  - Receive Traps คือส่วนที่ทำการรับ traps จาก agent มาแสดงผล โดยฟังก์ชันนี้จะใช้งานของเดิมที่มีอยู่แล้ว ดังหัวข้อ 3.2.1
  - Check Community คือการตรวจสอบ community name เพื่อดูว่า agent นั้นมีสิทธิ์ในการส่ง TrapIn ดังกล่าวเข้ามาหรือไม่ ถ้ามีก็จะทำงานส่วนถัดไป
  - Create Worker คือการสร้าง environment ในการทำงานให้กับแต่ละ TrapIn ที่ได้รับมาจาก agents

- Worker : เป็นส่วนที่ทำหน้าที่เกี่ยวกับการกรอง TrapIn ที่ได้รับมาจาก agents ซึ่งแบ่งฟังก์ชันการทำงานเป็น 3 ส่วนคือ
  - Check Duplicate จะตรวจสอบดูจำนวนครั้งที่ได้รับ TrapIn เดิมซ้ำๆ กัน แล้วนำมาเปรียบเทียบกับจำนวนที่ได้ระบุไว้ในคอนฟิกไฟล์
  - Check Timeout เมื่อได้รับ TrapIn เข้ามาจะมีการบันทึกเวลาที่ได้รับเอาไว้เพื่อนำมาเปรียบเทียบกับเวลาของเครื่องว่าช่วงเวลาที่ได้รับ TrapIn มานั้นครบกำหนด timeout แล้วหรือยัง
  - Check Condition จะเป็นการนำ TrapIn ต่างๆ ที่ได้รับมาตรวจสอบเงื่อนไขร่วมกัน เพื่อดูว่าในขณะนั้นได้เกิดเหตุการณ์ตามที่ได้ระบุไว้ในคอนฟิกไฟล์หรือไม่

3.3.2 การส่ง trap ให้กับ manager

เมื่อทำการประมวลผล TrapIn ตามฟังก์ชันต่างๆ ที่กล่าวมาข้างต้นแล้วพบว่า ตรงตามข้อกำหนดในคอนฟิกไฟล์ ส่วนที่เป็น Worker ก็จะเรียกใช้งานฟังก์ชัน Generate TrapOut เพื่อทำการ

ส่ง TrapOut ที่ครบจำนวนการกรองซ้ำ หรือครบกำหนด timeout หรือส่งเหตุการณ์ที่เกิดขึ้นจากการนำ TrapIn ต่างๆ มาตรวจสอบเงื่อนไขร่วมกันเพื่อรายงานไปยัง manager

โดยฟังก์ชัน Generate TrapOut นี้จะนำส่วนที่เรียกว่า snmptrap จาก CMU-SNMP toolkit distribution ซึ่งใช้ทดสอบการส่ง SNMPv1 trap สำหรับ agent มาใช้งาน ซึ่งโครงสร้างของคำสั่งดังกล่าวเป็นดังนี้

snmptrap host community trap-type specific-type device-description [-a agent-addr]

- *snmptrap* หมายถึงชื่อคำสั่งที่ใช้ในการทดสอบการส่ง SNMPv1 trap
- *host* หมายถึงชื่อ หรือเลข IP ของ host ที่จะทำการส่ง trap ไปให้
- *community* หมายถึง community name ซึ่งเป็นเหมือนรหัสผ่านที่ใช้ในการติดต่อกันสำหรับ SNMPv1 นั้น community name ที่อนุญาตให้ใช้ได้มีเพียงสองตัวคือ "public" (default) และ "private" (หรืออาจเป็นชื่อใดๆ ก็ได้ที่เราตั้งให้สำหรับ private community)
- *trap-type* หมายถึงตัวเลขที่ระบุชนิดของ trap โดยสำหรับ SNMPv1 จะประกอบด้วย trap ทั้งหมด 7 แบบ ดังที่ได้กล่าวไปแล้วในตารางที่ 2.2
- *specific-type* หมายถึงส่วนที่ให้ผู้ใช้สามารถกำหนดข้อมูลเฉพาะขึ้นมาใช้งานได้
- *device-description* หมายถึงข้อความแสดงรายละเอียดของเครื่องที่ทำการส่ง SNMPv1 trap
- *[-a agent-addr]* คือหมายเลข IP ของ agent ที่จะทำการส่ง SNMPv1 trap ถ้าไม่ระบุค่าที่ส่วนนี้จะถือว่าหมายเลข IP ของ agent คือค่าเดียวกับ host ที่เรียกใช้งานคำสั่งนี้

สำหรับในโครงงานนี้ ได้มีการกำหนดเพิ่มเติมในส่วนของ *trap-type* และ specific-type เพื่อให้สอดคล้องกับฟังก์ชันการกรอง trap ในหัวข้อ 3.3.1 และเปลี่ยนชื่อคำสั่งเป็น submgrtrap เพื่อไม่ให้เกิดความสับสน เนื่องจากการทดสอบในบทที่ 5 จำเป็นต้องอ้างอิงถึง snmptrap ในส่วนที่เป็น agent ดังนั้นโครงสร้างของคำสั่ง และค่าที่จะกำหนดให้แต่ละส่วนเป็นดังนี้

submgrtrap host community trap-type specific-type device-description [-a agent-addr]

- *host* จะใส่หมายเลข IP ของเครื่องที่เป็น Manager
- *community* สำหรับ SubManager ที่จะส่ง SNMPv1 trap ให้กับ manager กำหนดให้เป็น public

- *device-description* จะทำการอ่านมาจากไฟล์ /proc/version ของเครื่องที่ทำการรันโปรแกรม SubMamager อยู่
- *trap-type* สำหรับชนิดของ trap ที่ใช้ในโครงงานนี้จะใช้ trap หมายเลข 0 ถึง 6 ตาม RFC 1157 ดังที่กล่าวไว้แล้วในตารางที่ 2.2 และได้มีการเพิ่มเติม trap หมายเลข 7 เพื่อใช้ในการส่งรายงานเหตุการณ์ที่ตรงตามเงื่อนไขที่ได้ระบุไว้ในคอนฟิกไฟล์ไปยัง manager ดังตารางที่ 3.1

ตารางที่ 3.1 ชนิดข้อมูล trap สำหรับ submgrtrapd

trap-type	meaning
0	ColdStart
1	WarmStart
2	LinkDown
3	LinkUp
4	AuthenticationFailure
5	EgpNeighborLoss
6	EnterpriseSpecific
7	SituationReport

ค่าของ *trap-type* จะถูกพิจารณาโดยแบ่งออกเป็น 2 กรณีคือ

- กรณีที่เมื่อตรวจสอบพบว่าได้รับ TrapIn เดิมซ้ำกันจนครบจำนวนการกรองซ้ำ หรือครบกำหนด timeout จะทำการส่ง TrapOut ไปยัง manager โดยใช้ข้อมูลเดิมในส่วน *trap-type* ของ TrapIn ที่ได้รับมาจาก agent
- กรณีที่เมื่อนำ TrapIn ต่างๆ มาตรวจสอบเงื่อนไขร่วมกันแล้วทำให้เกิดเหตุการณ์ดังที่ได้ระบุไว้ในคอนฟิกไฟล์ขึ้น ข้อมูลในส่วน *trap-type* จะถูกระบุให้เป็นหมายเลข 7

- *specific-type* จะถูกพิจารณาโดยแบ่งออกเป็น 2 กรณีเช่นเดียวกันคือ
  - กรณีที่เมื่อตรวจสอบพบว่าได้รับ TrapIn เดิมซ้ำกันจนครบจำนวนการกรองซ้ำ หรือครบกำหนด timeout จะทำการส่ง TrapOut ไปยัง manager โดยใช้ข้อมูลเดิมในส่วน *specific-type* ของ TrapIn ที่ได้รับมาจาก agent

- กรณีที่เมื่อนำ TrapIn ต่างๆ มาตรวจสอบเงื่อนไขร่วมกันแล้วทำให้เกิดเหตุการณ์ดังที่ได้ระบุไว้ในคอนฟิกไฟล์ขึ้น ข้อมูลในส่วน *specific-type* จะถูกระบุให้เป็นหมายเลขประจำเงื่อนไขนั้น

- *-a agent-addr* คือส่วนที่จะบอก manager ว่า trap ที่ได้รับนั้นเป็นของ agent ใด โดยการพิจารณาจะแบ่งออกเป็น 2 กรณีเช่นเดียวกันคือ
  - กรณีที่เมื่อตรวจสอบพบว่าได้รับ TrapIn เดิมซ้ำกันจนครบจำนวนการกรองซ้ำ หรือครบกำหนด timeout จะทำการส่ง TrapOut ไปยัง manager โดยใช้ข้อมูลในส่วน *fhost* ที่ตรงกับ TrapIn ซึ่งได้รับมาจาก agent
  - กรณีที่เมื่อนำ TrapIn ต่างๆ มาตรวจสอบเงื่อนไขร่วมกันแล้วทำให้เกิดเหตุการณ์ดังที่ได้ระบุไว้ในคอนฟิกไฟล์ขึ้น ข้อมูลในส่วน *agent-addr* จะถูกระบุให้เป็นหมายเลข IP ของเครื่องที่ทำการรัน submgrtrapd อยู่

สำหรับลักษณะการทำงานโดยละเอียดภายในฟังก์ชันต่างๆ ของ submgrtrapd และ submgrtrap จะทำการอธิบายในส่วนของการออกแบบระบบงาน ซึ่งอยู่ในบทถัดไป

บทที่ 4

การออกแบบระบบงาน

สำหรับรายละเอียดในการออกแบบระบบงานในบทนี้ ได้มีการออกแบบฟังก์ชันการทำงานของ submgrtrapd ออกเป็น 3 ส่วนหลัก ส่วนแรกคือคอนฟิกไฟล์สำหรับเก็บข้อกำหนดและเงื่อนไขในการทำงาน ส่วนที่สองคือ Controller ทำหน้าที่รับ trap เข้ามาแล้วทำการตรวจสอบ community name ถ้าพบว่า trap นั้นมีสิทธิในการส่งถูกต้อง ก็จะทำการสร้างส่วนที่สามซึ่งเรียกว่า Worker ขึ้นมาเพื่อทำการกรอง traps ที่ซ้ำ ตรวจสอบ timeout ตรวจสอบเงื่อนไข รวมทั้งสร้าง trap เพื่อส่งรายงานเหตุการณ์ต่างๆ ไปยัง manager เมื่อพบว่าการตรวจสอบดังกล่าวข้างต้นเป็นไปตามที่ระบุไว้ในคอนฟิกไฟล์

4.1 รูปแบบคอนฟิกไฟล์

ฟังก์ชันต่างๆ ของ submgrtrapd ทั้งในส่วนของ Controller และ Worker จะทำงานโดยอาศัยข้อกำหนดที่อยู่ใน /etc/filter.conf ซึ่งคิดขึ้นมาใหม่เพื่อใช้ในการพัฒนาโปรแกรม SubManager ในส่วนที่จัดการเกี่ยวกับ SNMPv1 trap โดยภายในไฟล์นี้จะประกอบไปด้วยชุดของข้อกำหนด ดังรูปที่ 4.1 ซึ่งแบ่งออกเป็น 2 ส่วนได้แก่

```
#filter-id: fhost: fcommunity: ftrap-type: fspecific-type: counter: timeout
F1:    192.168.1.2:  public:  0: 0: 3: 30
F2:    192.168.1.2:  public:  1: 0: 2: -1
F3:    192.168.1.2:  public:  2: 0: 3: -1
F4:    192.168.1.2:  public:  3: 0: 10: -1
F5:    192.168.1.2:  jupiter: 4: 0: 3: -1
F6:    192.168.1.2:  public:  5: 0: 3: -1
F7:    192.168.1.14: public:  0: 0: 3: 30
F8:    192.168.1.14: public:  1: 0: 3: -1
F9:    192.168.1.14: public:  2: 0: 3: -1
F10:   192.168.1.14: sister:  3: 0: 3: -1
F11:   192.168.1.14: public:  4: 0: 5: -1
F12:   192.168.1.14: public:  5: 0: 3: -1
F13:   192.168.1.3:  public:  2: 0: 3: -1
F14:   192.168.1.3:  public:  3: 0: 3: -1
F15:   192.168.1.4:  public:  2: 0: 3: -1
F16:   192.168.1.4:  public:  3: 0: 3: -1
F17:   192.168.1.5:  public:  2: 0: 3: -1
F18:   192.168.1.5:  public:  3: 0: 3: -1

#condition-id: situation: filter-id#, filter-id#, filter-id#…
C1:  Electical Down: F3,F9,F13,F15,F17
C2:  Electical Up:    F4,F10,F14,F16,F18
```

รูปที่ 4.1 ตัวอย่างข้อกำหนดใน filter.conf

4.1.1 filter rules

คือส่วนที่เกี่ยวกับการกรอง trap มีรูปแบบดังนี้

filter-id : fhost : fcommunity : ftrap-type : fspecific-type : counter : timeout

- *filter-id* **คือหมายเลขประจำข้อกำหนด โดยส่วนนี้จะขึ้นต้นด้วย *'F'* เช่น *F1, F2, ..***
- *fhost* **คือส่วนที่ระบุชื่อ หรือเลข IP ของ host ที่ทำการส่ง TrapIn เข้ามา**
- *fcommunity* **คือส่วนที่ระบุ community name เพื่อใช้ตรวจสอบสิทธิ์ในการรับ TrapIn จาก agent เข้ามาประมวลผล**
- *ftrap-type* **คือตัวเลขระบุชนิดของ SNMPv1 trap ที่ต้องการจะกรองซ้ำ**
- *fspecific-type* **คือตัวเลขระบุชนิดของข้อมูลพิเศษที่ได้ทำการเพิ่มเติมขึ้น**
- *counter* **คือจำนวนครั้งที่ซ้ำของ TrapIn ที่รับเข้ามา ก่อนทำการส่งไปยัง manager**
- *timeout* **คือช่วงเวลาตั้งแต่ได้รับ TrapIn เข้ามาจนครบกำหนด มีหน่วยเป็น seconds ถ้าหากมีค่าเท่ากับ –1 จะหมายถึงกำหนดช่วงเวลาให้เป็น infinity**

ตัวอย่างการกำหนดคอนฟิกไฟล์

F3 : localhost : public : 2 : 0 : 10 : -1

เงื่อนไขนี้เป็นการระบุว่า agent ที่ชื่อ localhost มีสิทธิที่จะส่ง TrapIn แจ้งการ linkDown เข้ามาได้ และเมื่อโปรแกรมรับมาแล้วจะทำการเก็บข้อมูลดังกล่าวไว้ก่อนจนกว่าจะมีการส่งซ้ำเป็นครั้งที่ 10 หรือรอจนกว่าจะหมดเวลา(ในที่นี้คือไม่มีกำหนดเพราะค่าเท่ากับ -1) ก็จะส่งต่อข้อมูลดังกล่าวไปยัง manager

4.1.2 condition rules

หลังจากนำ TrapIn ที่รับเข้ามาตรวจสอบกับข้อกำหนดเกี่ยวกับการกรองซ้ำ และ timeout ข้างต้นแล้ว ก็จะนำ TrapIn นั้นมาพิจารณาร่วมกับ TrapIn อื่นๆ เพื่อดูว่าเกิดเหตุการณ์ตามที่กำหนดไว้ในส่วนการตรวจสอบเงื่อนไขนี้หรือไม่ ซึ่งมีรูปแบบดังนี้

condition-id : situation : filter-id#, filter-id#, filter-id#...

- *condition-id* **คือหมายเลขประจำเงื่อนไข โดยส่วนนี้จะขึ้นต้นด้วย *'C'* เช่น *C1, C2, ..***
- *situation* **คือสถานการณ์ที่จะรายงานไปยัง manager**
- *filter-id#* **คือเงื่อนไขต่างๆ ที่จะทำให้เกิดสถานการณ์ขึ้น ซึ่งจะระบุด้วยค่า *filter-id* ที่กล่าวถึงในหัวข้อก่อนหน้านี้ โดยการกำหนดเงื่อนไขขึ้นอยู่กับผู้ใช้งาน**

ตัวอย่างข้อกำหนดของการตรวจสอบเงื่อนไข

C1 : Electical Down : F2 , F3

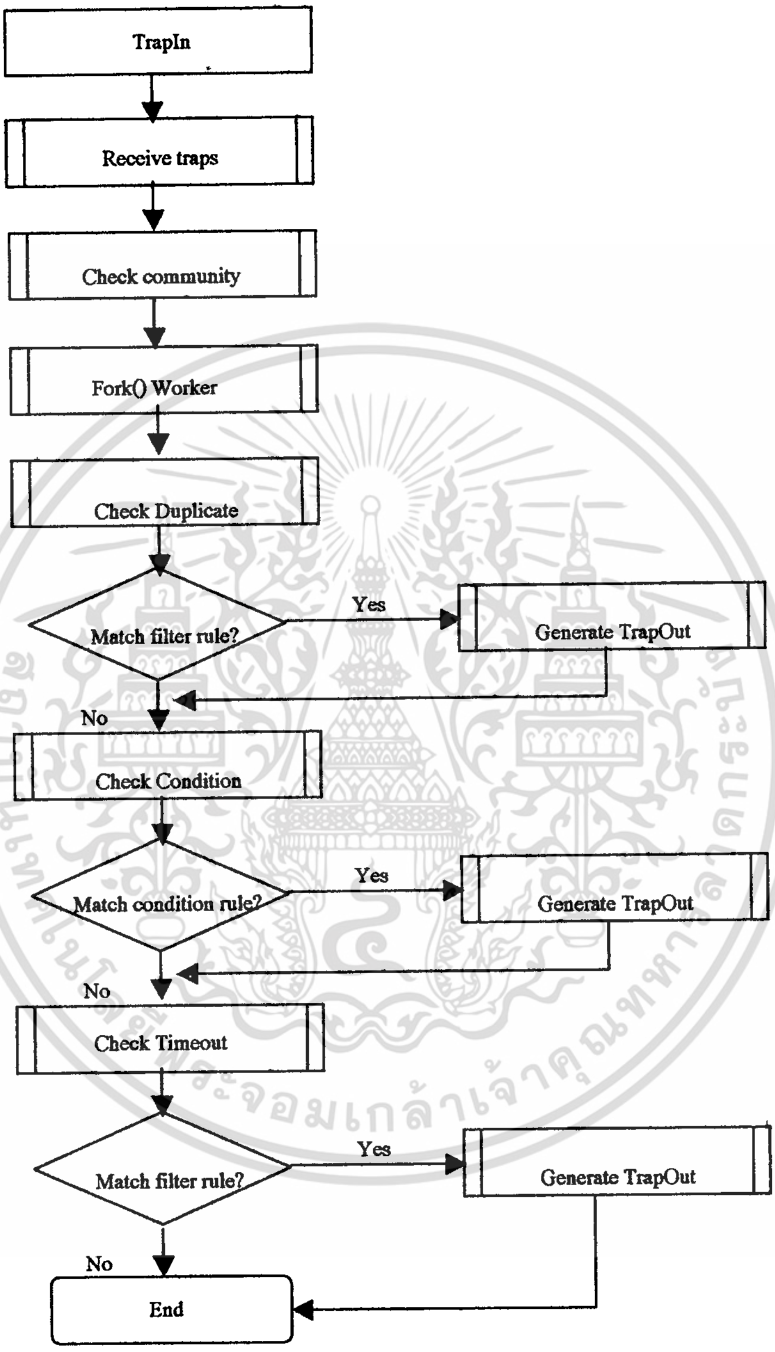
ข้อกำหนดนี้เป็นการระบุว่าถ้าโปรแกรมได้รับ TrapIn ที่ตรงกับที่กำหนดไว้ใน *filter-id F2* และ *F3* แสดงว่าในขณะนั้นเกิด Electical Down ขึ้น ให้ทำการส่ง TrapOut แจ้งเหตุการณ์ดังกล่าวไปยัง manager

4.2 ลำดับการทำงานของ submgrtrapd

ลำดับการทำงานของ submgrtrapd นั้นจะเป็นดังรูปที่ 4.2 เริ่มต้นจากการที่มีการส่ง TrapIn เข้ามา โดยฟังก์ชัน Receive traps จะทำการรับ TrapIn เข้ามา หลังจากนั้นจะเป็นหน้าที่ของฟังก์ชัน Check community ทำการตรวจสอบว่า TrapIn ดังกล่าวมีสิทธิ์ที่จะนำไปประมวลผลต่อหรือไม่ ถ้ามีก็จะเรียกใช้งานฟังก์ชัน Fork Worker แล้วส่งต่อ TrapIn ดังกล่าวไปทำการตรวจสอบการกรองซ้ำยังฟังก์ชัน Check Duplicate ถ้าพบว่าตรงตามที่กำหนดไว้ในคอนฟิกไฟล์ก็จะทำการเรียกใช้งานฟังก์ชัน Generate TrapOut เพื่อส่งต่อ TrapIn ดังกล่าวไปยัง manager

หลังจากนั้นก็จะทำงานต่อในส่วนของฟังก์ชัน Check Condition เพื่อตรวจสอบดูว่า TrapIn ดังกล่าวเป็นส่วนหนึ่งที่ทำให้เกิดเหตุการณ์ดังที่กำหนดเอาไว้ในคอนฟิกไฟล์หรือไม่ ถ้าพบว่าตรงตามที่กำหนดไว้ก็จะทำการเรียกใช้งานฟังก์ชัน Generate TrapOut เพื่อส่งรายงานเหตุการณ์ดังกล่าวไปยัง manager

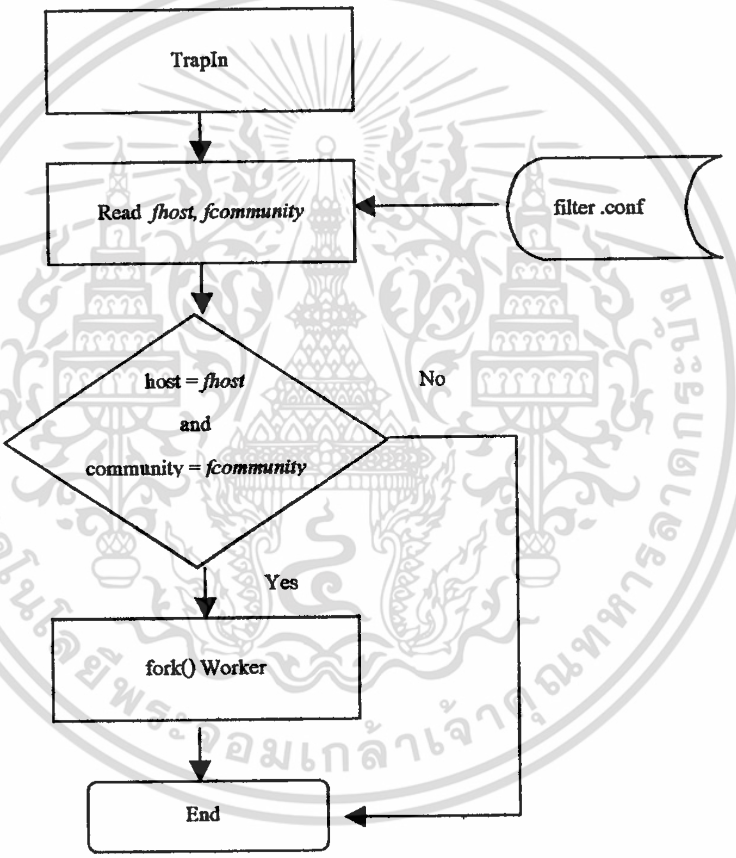
เมื่อตรวจสอบเงื่อนไขเรียบร้อยแล้ว ลำดับสุดท้ายจะเป็นการทำงานของฟังก์ชัน Check Timeout ทำการตรวจสอบดูว่าได้รับ TrapIn เข้ามาครบกำหนดเวลาที่ได้กำหนดเอาไว้แล้วหรือยัง ถ้าพบว่าตรงตามที่กำหนดไว้ในคอนฟิกไฟล์ก็จะทำการเรียกใช้งานฟังก์ชัน Generate TrapOut เพื่อส่งต่อ TrapIn ดังกล่าวไปยัง manager



รูปที่ 4.2 ลำดับการทำงานของ submgrtrapd

4.3 การทำงานของ Controller

เมื่อรัน submgrtrapd ส่วนนี้จะรอรับ TrapIn ที่ส่งมาจาก agents หลังจากนั้นทำการตรวจสอบ community name เพื่อดูว่า agent นั้นมีสิทธิ์ในการส่ง TrapIn ดังกล่าวเข้ามาหรือไม่ โดยจะทำการอ่านค่าใน field ที่ชื่อ *fhost* และ *fcommunity* จากส่วน filter rules ในไฟล์ filter.conf มาเปรียบเทียบกับค่า *host* และ *community* ที่ได้รับจาก TrapIn ถ้าพบว่าตรงกันทั้งสองส่วน ก็จะสร้าง Worker สำหรับ TrapIn นั้น เพื่อทำการกรอง และตรวจสอบเงื่อนไข แต่ถ้าไม่ตรงกันก็จะจบการทำงานในส่วนนี้ ดังรูปที่ 4.3



รูปที่ 4.3 การทำงานภายใน Controller

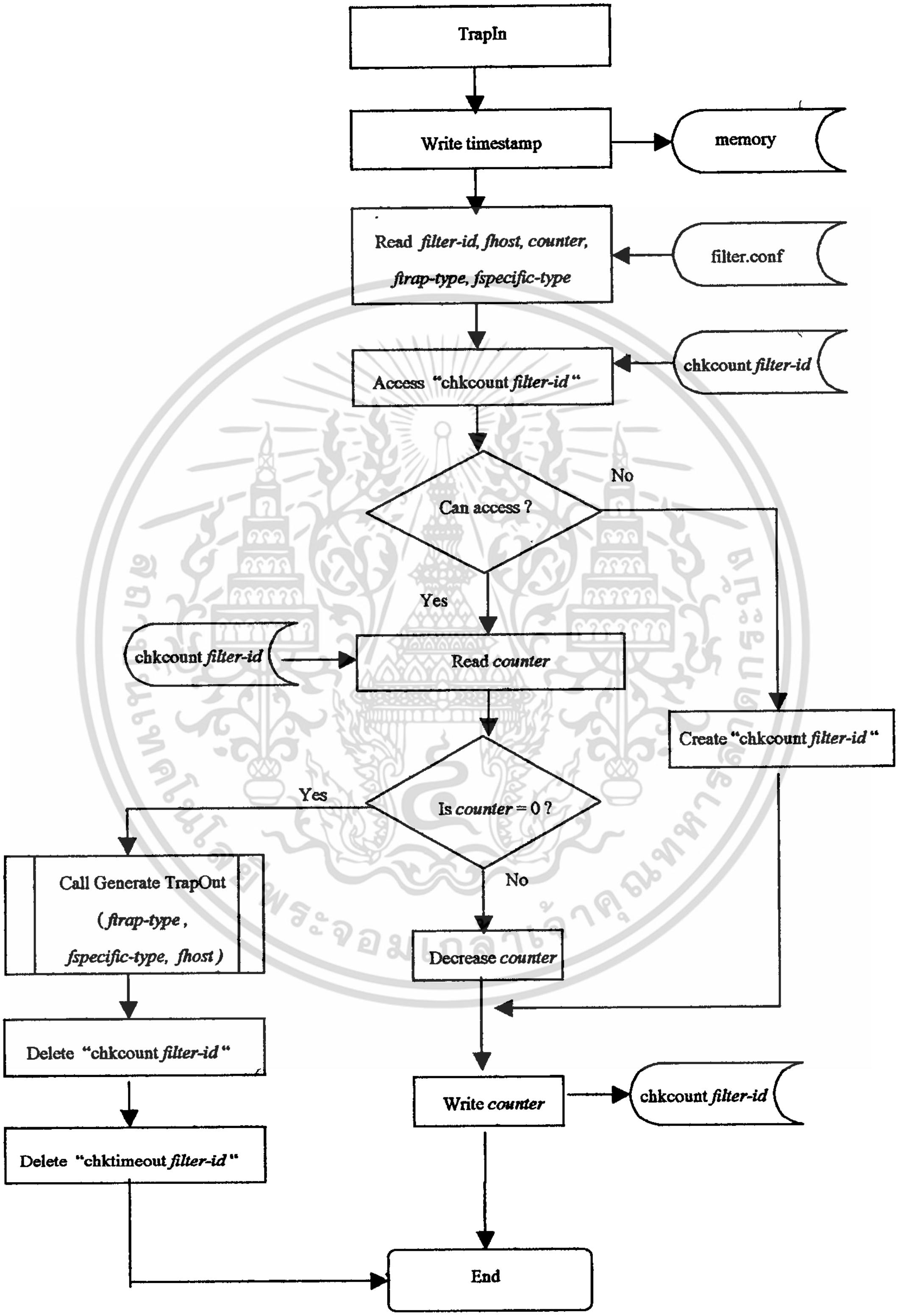
4.4 การทำงานของ Worker

เมื่อ Worker ถูกสร้างขึ้นขั้นตอนแรกที่จะต้องทำคือเก็บค่าของเวลาที่ได้รับ TrapIn นั้นไว้ในตัวแปรที่ชื่อ timestamp หลังจากนั้นก็จะนำข้อมูล TrapIn มาตรวจสอบการกรองซ้ำ ตรวจสอบ timeout และทำการตรวจสอบเงื่อนไข ซึ่งการทำงานแต่ละส่วนจะใช้ไฟล์ในการเก็บค่าตัวแปรที่ต้องใช้งานร่วมกัน มีรายละเอียดดังต่อไปนี้

4.4.1 การกรองซ้ำ

เมื่อได้รับ TrapIn เข้ามาก็จะบันทึกเวลาที่ได้รับนั้นไว้ในตัวแปรที่ชื่อ timestamp เพื่อใช้ในการตรวจสอบ timeout ในขั้นตอนต่อไป หลังจากนั้นจะทำงานตามขั้นตอนดังรูปที่ 4.4 ซึ่งอธิบายได้ดังนี้

- อ่าน และตีความไฟล์ filter.conf ในส่วน filter rules เพื่อดูว่า TrapIn ที่รับเข้ามานั้นตรงกับกฎการกรองข้อใด แล้วจึงดึงค่าที่เก็บอยู่ใน field ที่ชื่อ *filter-id fhost counter ftrap-type* และ *fspecific-type* มาเก็บไว้
- ทำการอ่านไฟล์ที่ชื่อ "chkcount*filter-id*" ซึ่งการที่ต้องนำค่าใน field ที่ชื่อ *filter-id* มาต่อท้ายชื่อไฟล์ ก็เพื่อให้ทราบว่าเป็นการตรวจสอบการกรองซ้ำของ TrapIn ใด
- ถ้าเข้าถึงไฟล์ดังกล่าวไม่ได้แสดงว่าเพิ่งได้รับ TrapIn นี้เป็นครั้งแรก ดังนั้นจะทำการสร้างไฟล์แล้วบันทึกค่า *counter* ลงไป
- แต่ถ้าเข้าถึงไฟล์นี้ได้แสดงว่าได้เคยรับ TrapIn นี้มาแล้ว ให้ทำการอ่านค่า *counter* ขึ้นมาว่าเท่ากับศูนย์หรือไม่
- ถ้ายังไม่ใช่ก็จะลดค่า *counter* ลงหนึ่ง แล้วบันทึกกลับลงไปในไฟล์
- แต่ถ้าค่า *counter* ลดลงจนเท่ากับศูนย์ แสดงว่าได้รับ TrapIn นี้ซ้ำจนครบที่กำหนดไว้แล้วจะทำการเรียกใช้งานฟังก์ชัน Generate TrapOut โดยส่ง argument คือ *ftrap-type fspecific-type* และ *fhost* ไปให้
- หลังจากนั้นจะทำการลบไฟล์ "chkcount*filter-id*" ที่ใช้ในการตรวจสอบจำนวนซ้ำนี้ทิ้งไปรวมทั้งยกเลิกการทำงานส่วน timeout ของ TrapIn นี้ด้วย

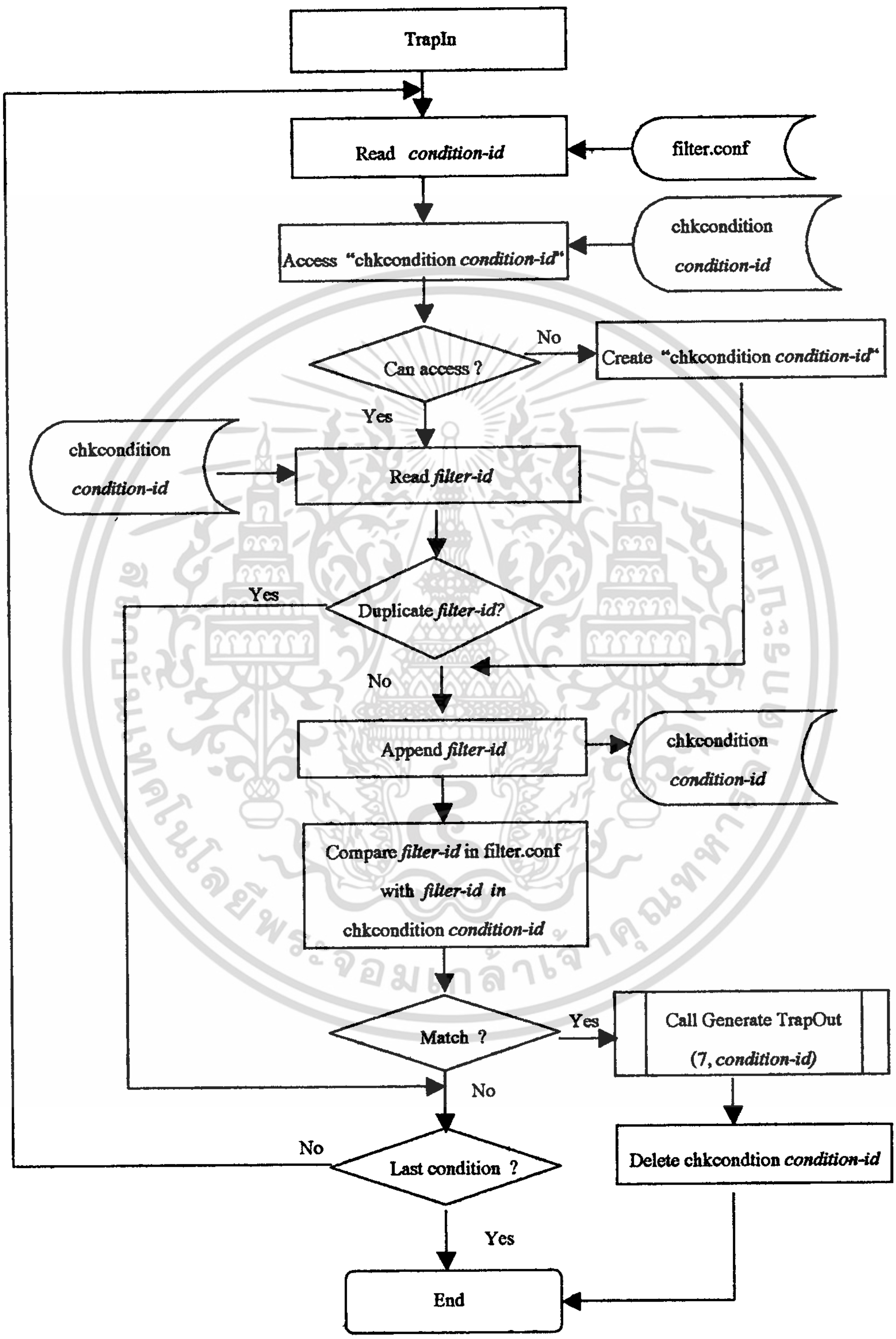


รูปที่ 4.4 การทำงานของการกรองซ้ำ

4.4.2 การตรวจสอบเงื่อนไข

เป็นการตรวจสอบว่า TrapIn ที่ได้รับมานั้น เมื่อนำมาพิจารณาร่วมกับ TrapIn อื่นๆ แล้วทำให้เกิดเหตุการณ์ตรงกับที่ระบุไว้ในส่วน condition rule ที่อยู่ในไฟล์ filter.conf หรือไม่ ซึ่งจะทำงานตามขั้นตอนดังรูปที่ 4.5 อธิบายได้ดังนี้

- อ่าน และตีความไฟล์ filter.conf ในส่วน condition rules ทีละเงื่อนไขเพื่อดูว่า TrapIn ที่รับเข้ามานั้นเป็นส่วนหนึ่งที่ทำให้เกิดเหตุการณ์ใดบ้าง
- ถ้าพบว่า TrapIn ดังกล่าวถูกระบุไว้เป็นส่วนหนึ่งที่ทำให้เกิดเหตุการณ์ในเงื่อนไขใด ก็จะอ่านค่าใน field ที่ชื่อ *condition-id* มาเก็บไว้
- หลังจากนั้นทำการอ่านไฟล์ที่ชื่อ "chkcondition *condition-id*" ซึ่งการที่ต้องนำค่าใน field ที่ชื่อ *condition-id* มาต่อท้ายก็เพื่อให้ทราบว่าเป็นการตรวจสอบเงื่อนไขของเหตุการณ์ใดอยู่
- ถ้าเข้าถึงไฟล์ "chkcondition *condition-id*" ไม่ได้แสดงว่าเพิ่งได้รับ TrapIn ที่เป็นส่วนหนึ่งของเหตุการณ์นี้ครั้งแรก ดังนั้นจะทำการสร้างไฟล์ แล้วบันทึกค่า *filter-id* ของ TrapIn ดังกล่าวลงไป
- แต่ถ้าเข้าถึงไฟล์นี้ได้แสดงว่าได้เคยรับ TrapIn ที่เป็นส่วนหนึ่งของเหตุการณ์นี้มาแล้ว ให้ทำการตรวจดูว่าในไฟล์ดังกล่าวมีค่า *filter-id* ที่ตรงกับ TrapIn นี้หรือไม่ ถ้ามีแล้วจะสิ้นสุดการทำงานของฟังก์ชันนี้ แต่ถ้ายัง ให้ทำการบันทึกค่า *filter-id* ลงในไฟล์
- อ่าน และตีความไฟล์ filter.conf ส่วน condition rules ในบรรทัดที่ตรงกับ *condition-id* แล้วนำค่าต่างๆ ใน field ที่ชื่อว่า *filter-id#* มาเปรียบเทียบกับ *filter-id* ที่ถูกบันทึกเอาไว้ในไฟล์ "chkcondition *condition-id*"
- ถ้าพบว่าได้รับเงื่อนไขครบตามที่ระบุไว้ในส่วน condition rules แสดงว่าได้เกิดเหตุการณ์ดังที่ระบุไว้ใน field ที่ชื่อ *situation* ดังนั้นจะทำการเรียกใช้งานฟังก์ชัน Generate TrapOut โดยส่ง argument คือ *ftrap-type* ซึ่งในกรณีนี้มีค่าเท่ากับ 7 (SituationReport) และ *specific-type* ซึ่งในกรณีนี้มีค่าเท่ากับหมายเลขหลังตัวอักษร C ซึ่งตรงกับ *condition-id* ที่กำลังทำการตรวจสอบเงื่อนไขอยู่
- หลังจากนั้นทำการลบไฟล์ "chkcondition *condition-id* " ที่ใช้ตรวจสอบเหตุการณ์นี้ทิ้งไป
- หลังจากนั้นจะตรวจสอบว่ายังมีเงื่อนไขต่อไปอีกหรือไม่ ถ้ามีก็จะทำงานตามขั้นตอนต่างๆ ดังที่กล่าวข้างต้นทั้งหมดซ้ำอีก และจะทำงานไปเรื่อยๆ จนกว่าจะหมดเงื่อนไขที่ได้ระบุไว้ในคอนฟิกไฟล์

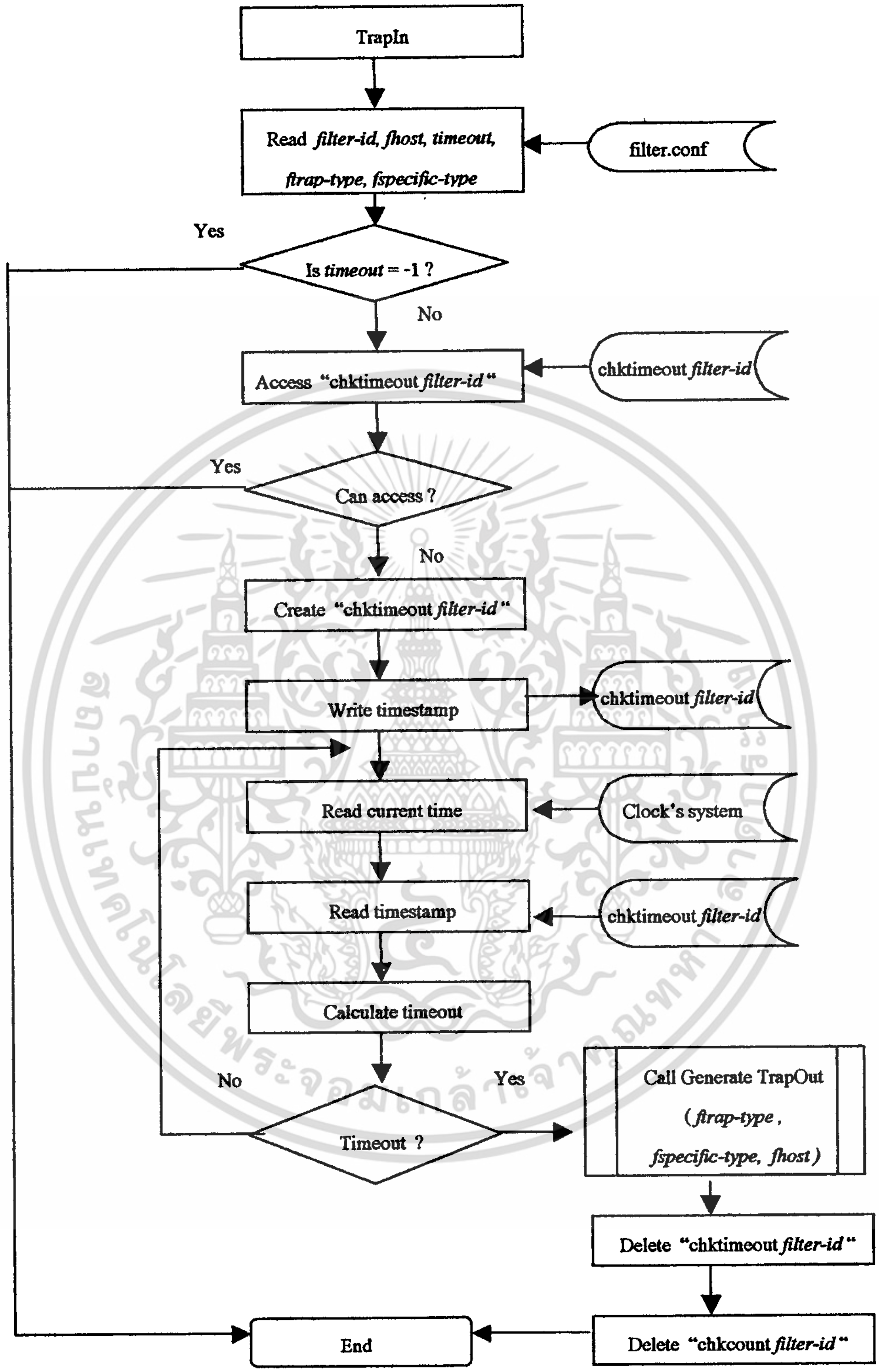


รูปที่ 4.5 การทำงานของการตรวจสอบเงื่อนไข

4.4.3 การตรวจสอบ timeout

ฟังก์ชันนี้จะใช้ค่าที่เก็บอยู่ในตัวแปรที่ชื่อ timestamp มาเปรียบเทียบกับเวลาของเครื่องที่ทำการรัน submgrtrapd อยู่ เพื่อหาผลต่างของเวลา ถ้าผลลัพธ์ที่ได้ยังไม่ตรงกับค่าที่กำหนดไว้ในคอนฟิกไฟล์ ฟังก์ชันนี้ก็จะทำการคำนวณเวลาไปเรื่อยๆ จนกว่าจะพบว่าครบกำหนดเวลาก็จะทำการส่งต่อ TrapIn นั้นไปยัง manager ซึ่งเป็นไปตามตามขั้นตอนดังรูปที่ 4.6 อธิบายได้ดังนี้

- อ่าน และตีความไฟล์ filter.conf ในส่วน filter rules เพื่อดูว่า TrapIn ที่รับเข้ามานั้นตรงกับกฎการกรองข้อใด แล้วจึงดึงค่าที่เก็บอยู่ใน field ที่ชื่อ *filter-id fhost timeout ftrap-type* และ *fspecific-type* มาเก็บไว้
- ดูว่าค่าของ *timeout* ที่กำหนดไว้มีค่าเป็น –1 หรือไม่ ถ้าใช่แสดงว่ากำหนดให้เป็น infinity ถือว่าสิ้นสุดการทำงานในส่วนนี้
- ทำการอ่านไฟล์ที่ชื่อ "chktimeout *filter-id*" ซึ่งการที่ต้องนำค่าใน field ที่ชื่อ *filter-id* มาต่อท้ายก็เพื่อให้ทราบว่าเป็นการตรวจสอบ timeout ของ TrapIn ใด
- ถ้าเข้าถึงไฟล์นี้ได้แสดงว่าได้เคยรับ TrapIn นี้มาแล้ว และกำลังตรวจสอบ timeout อยู่ จะสิ้นสุดการทำงานของฟังก์ชันนี้
- แต่ถ้าเข้าถึงไฟล์ดังกล่าวไม่ได้แสดงว่าเพิ่งได้รับ TrapIn นี้เป็นครั้งแรก ดังนั้นจะทำการสร้างไฟล์ แล้วบันทึกค่าที่เก็บไว้ในตัวแปรที่ชื่อ timestamp ลงไป
- ทำการอ่านเวลาปัจจุบันจากนาฬิกาของเครื่อง และค่า timestamp มาคำนวณหาผลต่างของช่วงเวลาไปเรื่อยๆ จนกว่าจะพบว่าค่าที่ได้เกินกว่าค่าใน *timeout* แสดงว่าครบกำหนดเวลาจะทำการเรียกใช้งานฟังก์ชัน Generate TrapOut โดยส่ง argument คือ *ftrap-type* และ *fspecific-type* และ *fhost* ไปให้
- หลังจากนั้นจะทำการลบไฟล์ "chktimeout *filter-id*" ที่ใช้ในการตรวจสอบนี้ทิ้งไป รวมทั้งยกเลิกการทำงานส่วนกรองซ้ำของ TrapIn นี้ด้วย



รูปที่ 4.6 การทำงานของการตรวจสอบ timeout

4.5 การทำงานของ Generate TrapOut

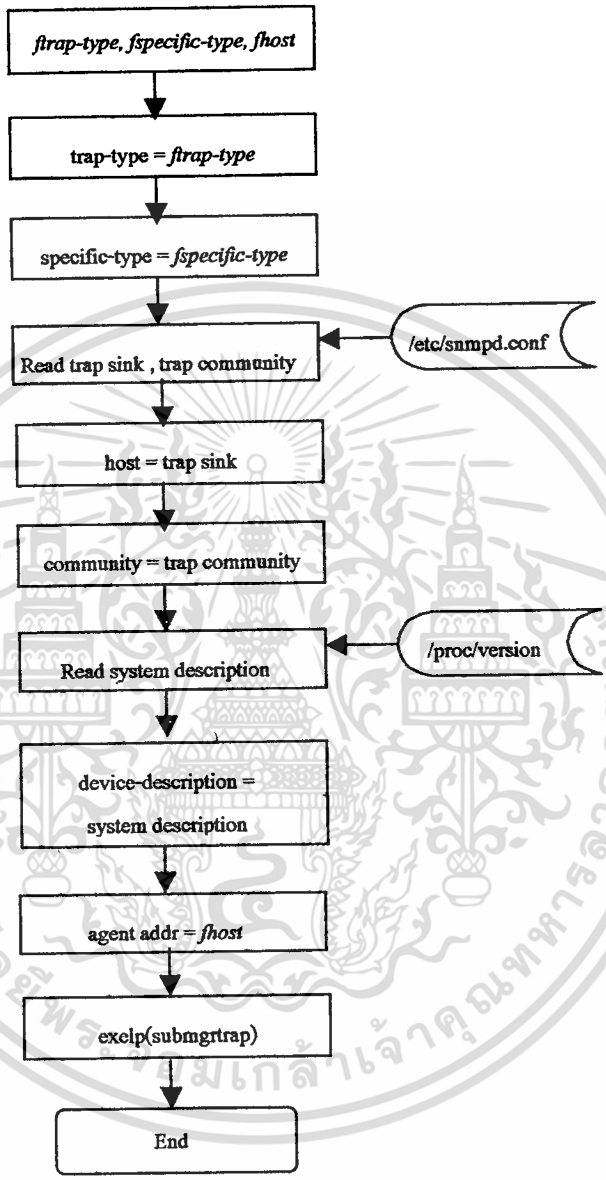
จากการทำงานของฟังก์ชันต่างๆ ดังกล่าวข้างต้น เมื่อตรวจสอบพบว่าตรงกับข้อกำหนดที่ได้ระบุไว้ก็จะต้องเรียกใช้งานฟังก์ชัน Generate TrapOut นี้ เพื่อให้ทำการส่งรายงานในรูปของ trap ไปยัง manager ซึ่งในโครงงานนี้จะเรียกว่า TrapOut

ฟังก์ชันนี้จะทำงานโดยนำคำสั่ง submgrtrap ดังที่ได้กล่าวไปแล้วในหัวข้อที่ 3.3.2 ซึ่งมีรูปแบบคือ *"submgrtrap host community trap-type specific-type device-description"* มาทำการ execute โดยขั้นตอนการทำงานจะแยกอธิบายเป็น 2 ส่วน ได้แก่ Generate TrapOut สำหรับกรณีที่ตรงตามเงื่อนไขในส่วน Filter และกรณีที่ตรงตามเงื่อนไขในส่วน Condition

4.5.1 กรณีที่ตรงตามเงื่อนไขในส่วน Filter

เมื่อการทำงานในส่วนของฟังก์ชันกรองซ้ำ และตรวจสอบ Timeout พบว่าตรงตามเงื่อนไขที่ได้กำหนดไว้ในคอนฟิกไฟล์ ก็จะทำการเรียกใช้งานฟังก์ชัน Generate TrapOut โดยส่งต่อค่า *ftrap-type* และ *fspecific-type* มาให้ ซึ่งเมื่อได้รับค่าดังกล่าวฟังก์ชันก็จะทำงานตามขั้นตอนดังรูปที่ 4.7

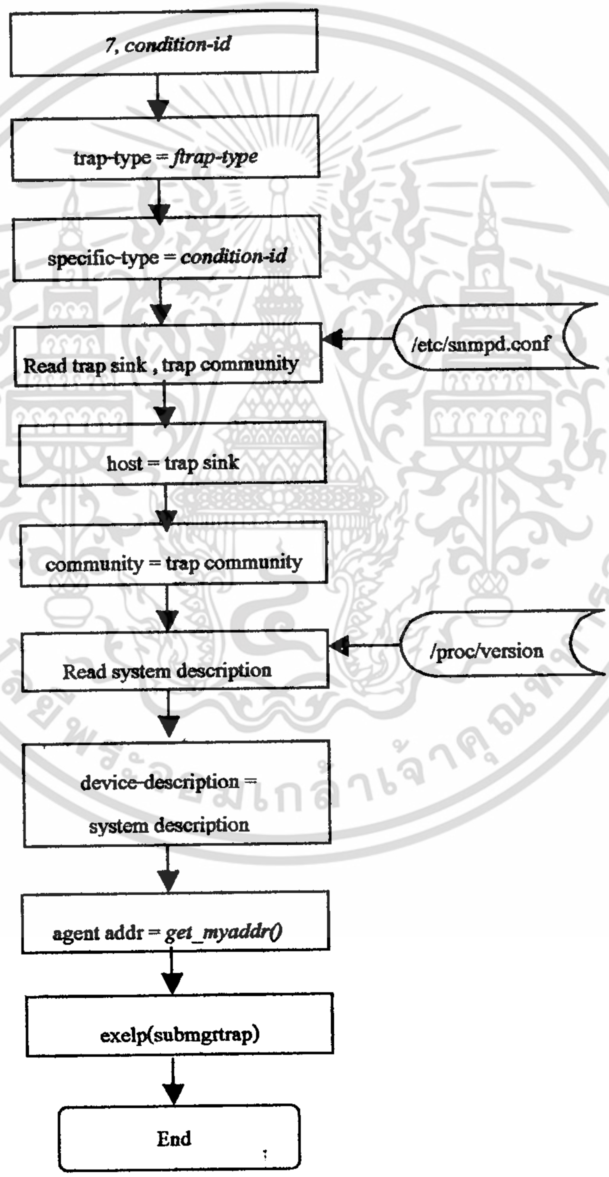
- รับ argument *ftrap-type* มาใส่ให้กับ field ที่ชื่อ *trap-type* ซึ่งค่านี้จะเป็นค่าเดียวกันกับชนิดของ TrapIn ที่รับเข้ามา
- รับ argument *fspecific-type* มาใส่ให้กับ field ที่ชื่อ *specific-type*ซึ่งค่านี้จะเป็นค่าเดียวกันกับชนิดของ TrapIn ที่รับเข้ามา
- อ่านค่าใน field ที่ชื่อ trap sink จากไฟล์ /etc/snmpd.conf ซึ่งเป็นไฟล์ที่กำหนดค่าต่างๆ เกี่ยวกับโปรแกรม SubManager มาใส่ให้กับ field ที่ชื่อ *host* โดยรายละเอียดเกี่ยวกับไฟล์ snmpd.conf จะขอกล่าวถึงในส่วนภาคผนวก ข.
- อ่านค่าใน field ที่ชื่อ trap community จากไฟล์ /etc/snmpd.conf ซึ่งเป็นไฟล์ที่กำหนดค่าต่างๆ เกี่ยวกับโปรแกรม SubManager มาใส่ให้กับ field ที่ชื่อ *community*
- ทำการอ่านรายละเอียดของเครื่องที่ทำการรันโปรแกรม SubManager มาจากไฟล์ /proc/version มาใส่ให้กับ field ที่ชื่อ *device-description*
- รับ argument *fhost* มาใส่ให้กับ field ที่ชื่อ *agent addr* ซึ่งค่านี้จะเป็นค่าเดียวกันกับหมายเลข IP ของ agent ที่ทำการส่ง TrapIn เข้ามา
- หลังจากนั้นก็ทำการ execute คำสั่ง *submgrtrap* ซึ่งเราได้กำหนดค่าให้ครบทุก field แล้วเพื่อทำการส่งต่อไปยัง manager



รูปที่ 4.7 การทำงานของ Generate TrapOut ในส่วน Filter

4.5.2 กรณีที่ตรงตามเงื่อนไขในส่วน Condition

เมื่อการทำงานในส่วนของฟังก์ชันตรวจสอบเงื่อนไขพบว่าเกิดเหตุการณ์ตรงตามที่ได้ระบุไว้ในคอนฟิกไฟล์ ก็จะทำการส่งค่า *ftrap-type* ซึ่งมีค่าเท่ากับ 7 (Situation Report) และ *fspecific-type* ซึ่งได้มาจากหมายเลขประจำเงื่อนไขที่ตามหลังตัวอักษร C มาให้กับฟังก์ชัน Generate TrapOut ซึ่งเมื่อได้รับค่าดังกล่าวฟังก์ชันก็จะทำงานตามขั้นตอนดังรูปที่ 4.8



รูปที่ 4.8 การทำงานของ Generate TrapOut ในส่วน Condition

- รับ argument *ftrap-type* มาใส่ให้กับ field ที่ชื่อ *trap-type* ซึ่งจะมีค่าเท่ากับ 7 หมายถึง trap ประเภท Situation Report
- รับ argument *condition-id* มาใส่ให้กับ field ที่ชื่อ *specific-type* เพื่อเป็นการระบุว่าเป็นเหตุการณ์ของเงื่อนไขใด
- อ่านค่าใน field ที่ชื่อ trap sink จากไฟล์ /etc/snmpd.conf ซึ่งเป็นไฟล์ที่กำหนดค่าต่างๆ เกี่ยวกับโปรแกรม SubManager มาใส่ให้กับ field ที่ชื่อ *host* โดยรายละเอียดเกี่ยวกับไฟล์ snmpd.conf จะขอกล่าวถึงในส่วนภาคผนวก ข.
- อ่านค่าใน field ที่ชื่อ trap community จากไฟล์ /etc/snmpd.conf ซึ่งเป็นไฟล์ที่กำหนดค่าต่างๆ เกี่ยวกับโปรแกรม SubManager มาใส่ให้กับ field ที่ชื่อ *community*
- ทำการอ่านรายละเอียดของเครื่องที่ทำการรันโปรแกรม SubManager มาจากไฟล์ /proc/version มาใส่ให้กับ field ที่ชื่อ *device-description*
- เรียกใช้ฟังก์ชัน get_myaddr() แล้วนำค่าที่ได้ซึ่งก็คือหมายเลข IP ของเครื่องที่เป็น SubManager มาใส่ให้กับ field ที่ชื่อ *agent addr*
- หลังจากนั้นก็ทำการ execute คำสั่ง *submgrtrap* ซึ่งเราได้กำหนดค่าให้ครบทุก field แล้วเพื่อทำการส่งต่อไปยัง manager

บทที่ 5

การทดสอบระบบงาน

ในบทนี้จะกล่าวถึงการทดสอบการทำงานของโปรแกรม SubManager ในส่วนที่เกี่ยวข้องกับการจัดการ SNMPv1 trap ดังที่ได้ออกแบบไว้ในบทที่ 4 โดยจะมีการสร้างสถานการณ์ในการรับ TrapIn เข้ามาในหลายๆ กรณี เพื่อให้ครอบคลุมการทำงานของทุกฟังก์ชัน และแสดงผลลัพธ์ที่ได้จากการทดสอบดังกล่าว

5.1 การทดสอบระบบงาน

การทดสอบระบบงานของโครงงานนี้ จะทำบนระบบปฏิบัติการ Linux Redhat ซึ่งการทดสอบนี้ใช้เครื่องเพียงเครื่องเดียวเป็นทั้ง manager SubManager และ agent โดยเริ่มแรกต้องเตรียมเครื่องให้พร้อมใช้งานดังขั้นตอนต่อไปนี้

1. นำ source code มาทำการคอมไพล์ด้วยคำสั่ง "make all install" จะเป็นการเรียกใช้งาน utitlty make เพื่อคอมไพล์ และติดตั้งโปรแกรม ดังรูปที่ 5.1

```
root@jupiter:/home/meaw - Shell - Konsole
Session Edit View Settings Help
[root@jupiter root]# cd /home/meaw
[root@jupiter meaw]# make all install
cd snmplib; make
make[1]: Entering directory `/home/meaw/snmplib'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/home/meaw/snmplib'
cd apps; make
make[1]: Entering directory `/home/meaw/apps'
cc -O  -I../snmplib    -c -o submgrtrapd.o submgrtrapd.c
cc -o submgrtrapd submgrtrapd.o -L../snmplib -lsnmp
make[1]: Leaving directory `/home/meaw/apps'
cd apps/snmpnetstat; make
make[1]: Entering directory `/home/meaw/apps/snmpnetstat'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/home/meaw/apps/snmpnetstat'
cd agent; make
make[1]: Entering directory `/home/meaw/agent'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/home/meaw/agent'
[root@jupiter root]# cd /home/meaw
[root@jupiter meaw]# make all install
cd snmplib; make
make[1]: Entering directory `/home/meaw/snmplib'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/home/meaw/snmplib'
cd apps; make
make[1]: Entering directory `/home/meaw/apps'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/home/meaw/apps'
```

รูปที่ 5.1 ตัวอย่างการติดตั้งโปรแกรมด้วยคำสั่ง "make all install"

2. ทำการเปิด console ขึ้นมารันคำสั่งดังนี้
   - console ที่ 1 รันคำสั่ง "snmptrapd -v 1 -P 163 -p" เพื่อให้เป็น manager รอรับ TrapOut ที่จะส่งมาจาก SubManager และที่ต้องระบุหมายเลขพอร์ตให้เป็น 163 เนื่องจากพอร์ต 162 ซึ่งเป็น default ในการรับ trap นั้น จะถูกใช้โดย SubManager
   - console ที่ 2 รันคำสั่ง "submgrtrapd -v 1 -p" เพื่อให้เป็น SubManager รอรับ trap ที่จะส่งมาจาก agent
   - ส่วน console ที่ 3 เปรียบเสมือนเป็น agent เอาไว้พิมพ์คำสั่ง snmptrap ซึ่งคำสั่งดังกล่าวได้นำมาจาก CMU-SNMP Toolkit ที่ได้กล่าวถึงไปแล้วในหัวข้อ 3.3.2 โดยการทดสอบการส่ง trap ในส่วนนี้จะทดลองระบุหมายเลข IP หลายๆ ค่าใน field ที่ชื่อ *[-a agent address]* ของคำสั่ง snmptrap แทนการส่ง trap มาจาก agent หลายๆ เครื่อง
3. ทำการกำหนดกฎในการกรองซ้ำ (filter rules) และเงื่อนไขที่จะทำให้เกิดเหตุการณ์ต่างๆ (condition rules) ลงในไฟล์ filter.conf ดังรูปที่ 5.2

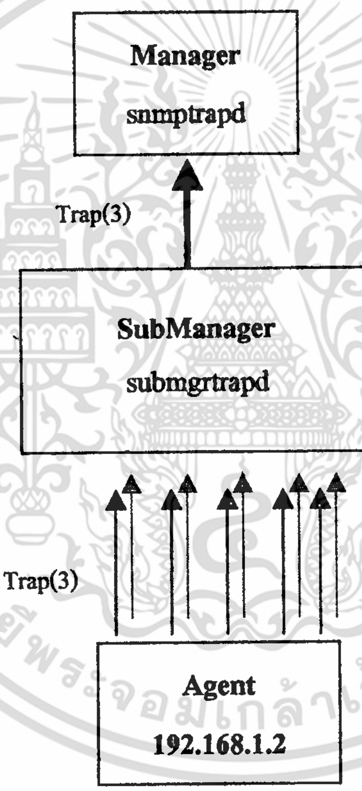
```
#filter-id: fhost: fcommunity: ftrap-type: fspecific-type: counter: timeout
F1:    192.168.1.2:  public:  0: 0: 3: 30
F2:    192.168.1.2:  public:  1: 0: 2: -1
F3:    192.168.1.2:  public:  2: 0: 3: -1
F4:    192.168.1.2:  public:  3: 0: 10: -1
F5:    192.168.1.2:  jupiter: 4: 0: 3: -1
F6:    192.168.1.2:  public:  5: 0: 3: -1
F7:    192.168.1.14: public:  0: 0: 3: 30
F8:    192.168.1.14: public:  1: 0: 3: -1
F9:    192.168.1.14: public:  2: 0: 3: -1
F10:   192.168.1.14: sister:  3: 0: 3: -1
F11:   192.168.1.14: public:  4: 0: 5: -1
F12:   192.168.1.14: public:  5: 0: 3: -1
F13:   192.168.1.3:  public:  2: 0: 3: -1
F14:   192.168.1.3:  public:  3: 0: 3: -1
F15:   192.168.1.4:  public:  2: 0: 3: -1
F16:   192.168.1.4:  public:  3: 0: 3: -1
F17:   192.168.1.5:  public:  2: 0: 3: -1
F18:   192.168.1.5:  public:  3: 0: 3: -1

#condition-id: situation: filter-id#, filter-id#, filter-id#…
C1:  Electical Down: F3,F9,F13,F15,F17
C2:  Electical Up:   F4,F10,F14,F16,F18
```

รูปที่ 5.2 ตัวอย่าง filter rules และ condition rules ในไฟล์ filter.conf ที่ใช้ทดลอง

5.2 สถานการณ์ที่ใช้ทดสอบระบบงาน

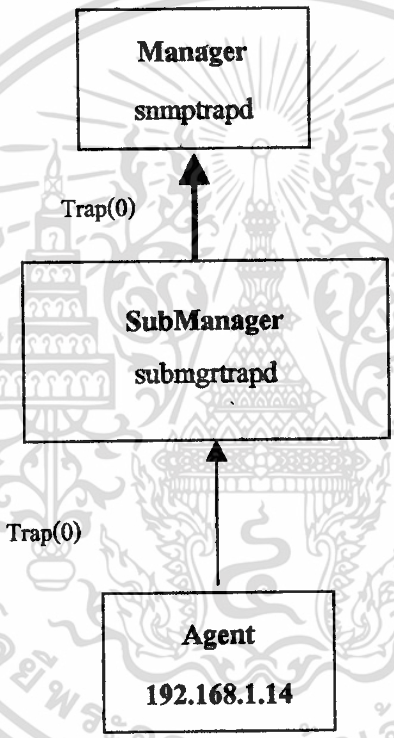
เมื่อทั้ง manager SubManager **และ** agent **พร้อมที่จะทำงานแล้ว ต่อไปนี้ก็จะเป็นการกำหนดสถานการณ์ทั้งหมด** 3 **กรณี เพื่อใช้ทดสอบฟังก์ชันต่างๆ**

- **<u>กรณีที่ 1</u> จะสมมุติให้** agent **ที่มีหมายเลข** IP 192.168.1.2 **ทำการส่ง** TrapIn **แจ้ง** linkUp **มายัง**submgrtrapd **จนครบตามจำนวนครั้งที่ระบุไว้ใน** field **ที่ชื่อ** *counter* **ของข้อกำหนด** F4 **ซึ่งในตัวอย่างนี้คือ** 10 **ครั้ง ดังนั้น** submgrtrapd **ก็จะส่งต่อ** TrapIn **ดังกล่าวไปยัง** manager **ดังรูปที่** 5.3



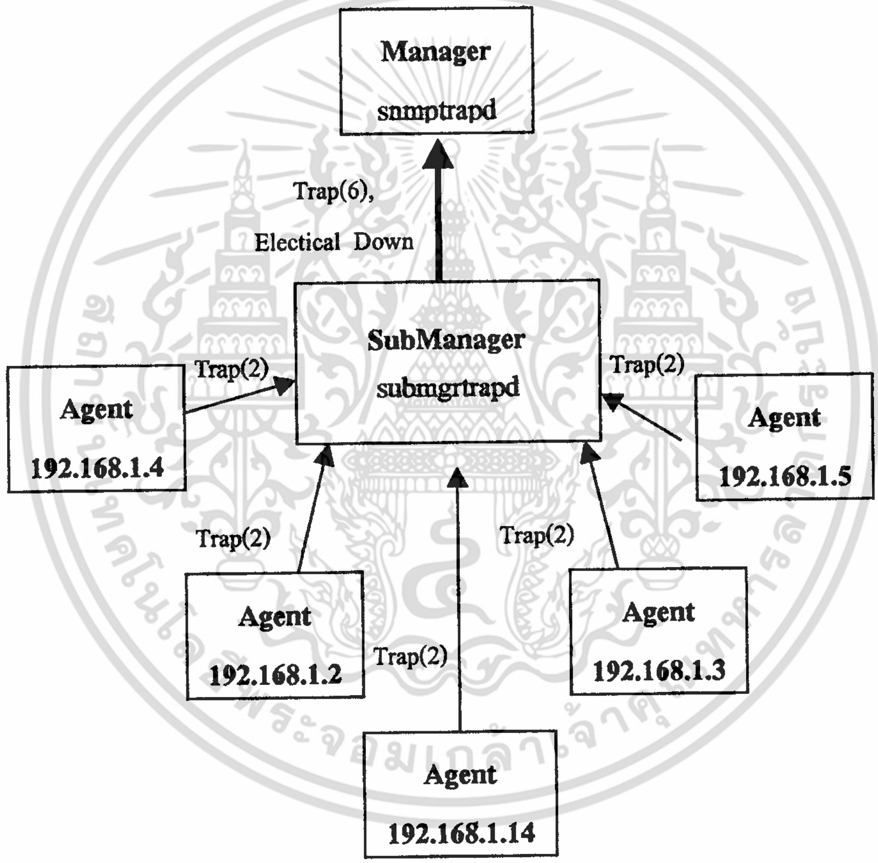
รูปที่ 5.3 การทดสอบฟังก์ชันการกรองซ้ำ

- <u>กรณีที่ 2</u> จะสมมุติให้ agent ที่มีหมายเลข IP 192.168.1.14 ทำการส่ง TrapIn แจ้ง coldStart มายัง submgrtrapd และหลังจากนั้นก็ไม่มีการส่ง TrapIn ดังกล่าวมาซ้ำอีกเลย เมื่อฟังก์ชัน ตรวจสอบ timeout พบว่าช่วงเวลาที่ได้รับ TrapIn นี้เกินกว่าที่ระบุอยู่ใน field *timeout* โปรแกรม submgrtrapd ก็จะทำการส่งต่อ TrapIn ดังกล่าวไปยัง manager ถึงแม้ว่าจะยังไม่ครบตามจำนวนครั้งที่ระบุไว้ใน field ที่ชื่อ *counter* ก็ตาม ดังรูปที่ 5.4



รูปที่ 5.4 การทดสอบฟังก์ชันการตรวจสอบ Timeout

- <u>กรณีที่ 3</u> สมมุติให้ agent ที่มีหมายเลข IP 192.168.1.2, 192.168.1.14, 192.168.1.3, 192.168.1.4 และ 192.168.1.5 ทำการส่ง TrapIn แจ้ง linkDown มายัง submgrtrapd เมื่อได้รับ TrapIn จาก agent ต่างๆ ดังกล่าว จะพบว่าตรงกับข้อกำหนด C1 ที่ระบุไว้ในไฟล์ filter.conf แสดงว่าเกิด Electrical Down ขึ้น ดังนั้น submgrtrapd ก็จะทำการแจ้งเหตุการณ์ที่เกิดขึ้นดังกล่าวไปยัง manager ดังรูปที่ 5.5



รูปที่ 5.5 การทดสอบฟังก์ชันการตรวจสอบเงื่อนไข

5.3 TrapIn ที่ใช้ทดสอบระบบงาน

การทดสอบระบบงานจะนำคำสั่ง snmptrap ซึ่งมีอยู่ใน CMU-SNMP toolkit มาใช้เพื่อส่ง TrapIn มายัง submgrtrapd ซึ่งคำสั่งดังกล่าวมีรูปแบบดังนี้

snmptrap host community trap-type specific-type device-description [-a agent-addr]

- *snmptrap* คือ SNMP application ที่ใช้ในการส่ง trap
- *host* คือส่วนที่ระบุชื่อ หรือเลข IP ของ host ที่จะทำการส่ง trap ไปให้
- *community* คือส่วนที่ระบุ community name ซึ่งเป็นเหมือนรหัสผ่านที่ใช้ในการติดต่อกัน สำหรับ SNMPv1 นั้นcommunity name ที่อนุญาตให้ใช้ได้มีเพียงสอง ตัวคือ “public” (default) และ “private” (หรืออาจเป็นชื่อใดๆ ก็ได้ที่เราตั้งให้สำหรับ private community)
- *trap-type* คือตัวเลขที่ระบุชนิดของ trap โดยสำหรับ SNMPv1 จะประกอบด้วย trap ทั้งหมด 7 แบบ ซึ่งได้กล่าวถึงไปแล้วในบทที่ 2 ดังตารางที่ 2.2
- *specific-type* คือส่วนที่ให้ผู้ใช้สามารถกำหนดข้อมูลเฉพาะขึ้นมาใช้งานได้
- *device-description* คือข้อความแสดงรายละเอียดของเครื่องที่ทำการส่ง trap
- *[-a agent-addr]* คือหมายเลข IP ของ agent ที่ทำการส่ง SNMPv1 trap ถ้าไม่ระบุค่าที่ส่วนนี้จะถือว่าหมายเลข IP ของ agent คือค่าเดียวกับ host

5.4 ผลการทดสอบระบบงาน

- <u>กรณีที่ 1</u> เมื่อ agent ที่มีหมายเลข IP 192.168.1.2 ทำการส่ง TrapIn แจ้ง linkUp ดังรูปที่ 5.6 ไปยังส่วนที่เป็น SubManager โปรแกรม submgrtrapd สามารถรับ TrapIn ดังกล่าวมาแสดงผลดังรูปที่ 5.7 และเมื่อ agent ที่มีหมายเลข IP ดังกล่าวทำการส่ง TrapIn ชนิดเดิมมาซ้ำจนครบ 10 ครั้งตามที่ระบุไว้ใน field ที่ชื่อ *counter* ของข้อกำหนด F4 โปรแกรม submgrtrapd ก็สามารถทำการส่งต่อ TrapIn ดังกล่าวไปยัง manager ได้ โดยข้อมูลที่ manager ได้รับจะเป็นดังรูปที่ 5.8

```
[root@meaw root]# snmptrap localhost public 3 0 "linux" -a 192.168.1.2
[root@meaw root]# snmptrap localhost public 3 0 "linux" -a 192.168.1.2
[root@meaw root]# snmptrap localhost public 3 0 "linux" -a 192.168.1.2
[root@meaw root]# snmptrap localhost public 3 0 "linux" -a 192.168.1.2
[root@meaw root]# snmptrap localhost public 3 0 "linux" -a 192.168.1.2
[root@meaw root]# snmptrap localhost public 3 0 "linux" -a 192.168.1.2
[root@meaw root]# snmptrap localhost public 3 0 "linux" -a 192.168.1.2
[root@meaw root]# snmptrap localhost public 3 0 "linux" -a 192.168.1.2
[root@meaw root]# snmptrap localhost public 3 0 "linux" -a 192.168.1.2
[root@meaw root]# snmptrap localhost public 3 0 "linux" -a 192.168.1.2
[root@meaw root]#
```

รูปที่ 5.6 ตัวอย่างการส่ง TrapIn จนครบตามจำนวน *counter*

```
[root@meaw root]# submgrtrapd -v 1 -p
192.168.1.2: Link Up Trap-type (0) Uptime: 8:20:01
system.sysDescr.0 = "linux"
--------------------------------------------------------
192.168.1.2: Link Up Trap-type (0) Uptime: 8:20:14
system.sysDescr.0 = "linux"
--------------------------------------------------------
192.168.1.2: Link Up Trap-type (0) Uptime: 8:20:15
system.sysDescr.0 = "linux"
--------------------------------------------------------
192.168.1.2: Link Up Trap-type (0) Uptime: 8:20:15
system.sysDescr.0 = "linux"
--------------------------------------------------------
192.168.1.2: Link Up Trap-type (0) Uptime: 8:20:16
system.sysDescr.0 = "linux"
--------------------------------------------------------
192.168.1.2: Link Up Trap-type (0) Uptime: 8:20:17
system.sysDescr.0 = "linux"
--------------------------------------------------------
192.168.1.2: Link Up Trap-type (0) Uptime: 8:20:17
system.sysDescr.0 = "linux"
--------------------------------------------------------
192.168.1.2: Link Up Trap-type (0) Uptime: 8:20:18
system.sysDescr.0 = "linux"
--------------------------------------------------------
192.168.1.2: Link Up Trap-type (0) Uptime: 8:20:19
system.sysDescr.0 = "linux"
--------------------------------------------------------
192.168.1.2: Link Up Trap-type (0) Uptime: 8:20:19
system.sysDescr.0 = "linux"
--------------------------------------------------------
send report trap to manager: match check count
The file /root/My Document/testSub/etc/sharefile/chkcountF4 has been deleted.
--------------------------------------------------------
```

รูปที่ 5.7 การทำงานของ submgrtrapd เมื่อได้รับ TrapIn ครบตามจำนวน *counter*

```
[root@meaw root]# snmptrapd -v 1 -P 163 -p
192.168.1.2: Link Up Trap (0) Uptime: 8:20:19
system.sysDescr.0 = "Linux version 2.4.7-10 (bhcompile@stripples.devel.redhat.com) (
(Red Hat Linux 7.1 2.96-98)) #1 Thu Sep 6 17:27:27 EDT 2001"
--------------------------------------------------------
```

รูปที่ 5.8 การทำงานของ snmptrapd เมื่อได้รับ TrapOut จากการกรองซ้ำ

- กรณีที่ 2 เมื่อ agent ที่มีหมายเลข IP 192.168.1.14 ทำการส่ง TrapIn แจ้ง coldStart ดังรูปที่ 5.9 ไปยังส่วนที่เป็น SubManager โปรแกรม submgrtrapd สามารถทำการรับ TrapIn ดังกล่าวมาแสดงผลดังรูปที่ 5.10 และหลังจากนั้น agent ที่มีหมายเลข IP ดังกล่าวก็ไม่ได้ส่ง TrapIn นี้ซ้ำอีกเลย เมื่อเวลาผ่านไป 30 วินาที เครื่องที่เป็น manager ได้รับ trap แจ้ง coldStart จาก agent IP 192.168.1.14 ดังรูปที่ 5.11 นั่นแสดงว่าฟังก์ชันตรวจสอบ timeout สามารถทำงานได้ถูกต้องกล่าวคือทำการเปรียบเทียบเวลาที่ได้รับ TrapIn กับเวลาปัจจุบันของเครื่องที่เป็น SubManager เมื่อพบว่าเกินกว่าที่ระบุอยู่ใน field *timeout* โปรแกรม submgrtrapd จึงทำการส่งต่อ TrapIn ดังกล่าวไปยัง manager ถึงแม้ว่าจะยังไม่ครบตามจำนวนครั้งที่ระบุไว้ใน field ที่ชื่อ *counter* ก็ตาม

```
[root@meaw root]# snmptrap localhost public 0 0 "linux" -a 192.168.1.14
[root@meaw root]#
```

รูปที่ 5.9 ตัวอย่างการส่ง TrapIn ที่จะทำให้เกิด Timeout

```
[root@meaw root]# submgrtrapd -v 1 -p
192.168.1.14: Cold Start Trap-type (0) Uptime: 8:26:31
system.sysDescr.0 = "linux"
-------------------------------------------------------
send report trap to manager: match check timeout
The file /root/My Document/testSub/etc/sharefile/chktimeoutF7, has been deleted.
The file /root/My Document/testSub/etc/sharefile/chkcountF7 has been deleted.
-------------------------------------------------------
```

รูปที่ 5.10 การทำงานของ submgrtrapd เมื่อได้รับ TrapIn แล้วเกิด Timeout

```
[root@meaw root]# snmptrapd -v 1 -P 163 -p
192.168.1.14: Cold Start Trap (0) Uptime: 8:27:00
system.sysDescr.0 = "Linux version 2.4.7-10 (bhcompile@stripples.devel.redhat.com)
(Red Hat Linux 7.1 2.96-98)) #1 Thu Sep 6 17:27:27 EDT 2001"
-------------------------------------------------------
```

รูปที่ 5.11 การทำงานของ snmptrapd เมื่อได้รับ TrapOut จากกรณี Timeout

- <u>กรณีที่ 3</u> เมื่อ agent ที่มีหมายเลข IP 192.168.1.2, 192.168.1.14, 192.168.1.3, 192.168.1.4 และ 192.168.1.5 ทำการส่ง TrapIn แจ้ง linkDown ดังรูปที่ 5.12 ไปยังส่วนที่เป็น SubManager โปรแกรม submgrtrapd สามารถทำการรับ TrapIn ดังกล่าวมาแสดงผลดังรูปที่ 5.13 และเมื่อได้รับ TrapIn จาก agent ต่างๆ ดังกล่าวจนครบ เครื่องที่เป็น manager ก็ได้รับ trap แจ้ง Situation Report ว่าเกิด Electrical Down ขึ้น จาก agent IP 192.168.0.1 ซึ่งคือเครื่องที่ทำการรับ submgrtrapd ดังรูปที่ 5.14 นั่นแสดงว่าฟังก์ชันตรวจสอบเงื่อนไขทำงานได้ถูกต้อง คือพบว่า TrapIn ต่างที่ได้รับมาตรงกับข้อกำหนด C1 ที่ระบุไว้ในไฟล์ filter.conf

```
[root@meaw root]# snmptrap localhost public 2 0 "linux" -a 192.168.1.14
[root@meaw root]# snmptrap localhost public 2 0 "linux" -a 192.168.1.2
[root@meaw root]# snmptrap localhost public 2 0 "linux" -a 192.168.1.5
[root@meaw root]# snmptrap localhost public 2 0 "linux" -a 192.168.1.4
[root@meaw root]# snmptrap localhost public 2 0 "linux" -a 192.168.1.3
[root@meaw root]#
```

รูปที่ 5.12 ตัวอย่างการส่ง TrapIn ที่จะทำให้เกิด *situation*

```
[root@meaw root]# submgrtrapd -v 1 -p
192.168.1.14: Link Down Trap-type (0) Uptime: 8:31:16
system.sysDescr.0 = "linux"
------------------------------------------------------
192.168.1.2: Link Down Trap-type (0) Uptime: 8:31:22
system.sysDescr.0 = "linux"
------------------------------------------------------
192.168.1.5: Link Down Trap-type (0) Uptime: 8:31:25
system.sysDescr.0 = "linux"
------------------------------------------------------
192.168.1.4: Link Down Trap-type (0) Uptime: 8:31:28
system.sysDescr.0 = "linux"
------------------------------------------------------
192.168.1.3: Link Down Trap-type (0) Uptime: 8:31:31
system.sysDescr.0 = "linux"
------------------------------------------------------
The file /root/My Document/testSub/etc/sharefile/chkconditionC1 has been deleted.
------------------------------------------------------
```

รูปที่ 5.13 การทำงานของ submgrtrapd เมื่อได้รับ TrapIn แล้วเกิด *situation*

```
[root@meaw root]# snmptrapd -v 1 -P 163 -p
192.168.0.1: Situation Report Trap (1) Uptime: 8:31:31
system.sysDescr.0 = "Linux version 2.4.7-10 (bhcompile@stripples.devel.redhat.com)
(Red Hat Linux 7.1 2.96-98)) #1 Thu Sep 6 17:27:27 EDT 2001"

 Receive situation : Electical Down
------------------------------------------------------
```

รูปที่ 5.14 การทำงานของ snmptrapd เมื่อได้รับ TrapOut จากการตรวจสอบเงื่อนไขร่วมกัน

บทที่ 6
สรุปผลและข้อเสนอแนะ

เนื้อหาในบทนี้จะเป็นการสรุปผลโดยรวมของการทดสอบการใช้งานโปรแกรม SubManager ในส่วนที่เกี่ยวข้องกับการจัดการ SNMPv1 trap ที่ได้แสดงไว้ในบทที่ 5 และอธิบายถึงภาพรวมของการนำโปรแกรมดังกล่าวไปใช้งาน พร้อมทั้งข้อเสนอแนะสำหรับผู้ที่สนใจจะนำโปรแกรมไปทำการพัฒนาต่อไป

6.1 สรุปผลการทดสอบระบบงาน

ผลการทดลองในบทก่อนหน้านี้ ทำให้เห็นว่าโปรแกรม SubManager ในส่วนที่เกี่ยวข้องกับการจัดการ SNMPv1 trap นี้ สามารถทำงานตามฟังก์ชันที่ได้พัฒนาเพิ่มเติมได้อย่างถูกต้อง กล่าวคือสามารถกรอง trap ที่ซ้ำ ตรวจสอบ timeout และสามารถนำ traps ที่รับจาก agents ต่างๆ มาตรวจสอบเงื่อนไขร่วมกันได้

ดังนั้นในเครือข่ายที่มีการส่งข้อมูล trap เป็นจำนวนมาก สามารถนำโปรแกรมดังกล่าวเข้าไปช่วยประมวลผล trap ตามฟังก์ชันต่างๆ ก่อนได้ โดยทำการรันโปรแกรมดังกล่าวไว้ที่ host ในเครือข่าย แล้วให้ทุก agents ทำการส่ง trap แจ้งเหตุการณ์ต่างๆ มายัง host นี้ และถ้าหากเครือข่ายมีขนาดใหญ่มาก หรือในกรณีเชื่อมต่อหลายๆ เครือข่ายเข้าด้วยกัน ก็สามารถทำเป็น hierarchy ได้ด้วยการรันโปรแกรมไว้ที่ host มากกว่าหนึ่งเครื่อง แล้วใช้ field ที่ชื่อ specific-type ในการระบุลำดับชั้น หรือหมายเลขประจำ host ก็ได้

เมื่อ host ที่ทำการรันโปรแกรมดังกล่าวในเครือข่ายได้รับ trap จาก agents ที่ตนเองรับผิดชอบก็จะนำข้อมูลจาก trap เหล่านั้นมาประมวลผลร่วมกัน โดยจะพิจารณาบทสรุปของเหตุการณ์ที่เกิดขึ้นตามที่ได้ระบุไว้ใน configuration file เช่นในกรณีที่ทุก agents มีการส่ง trap แจ้ง link down เข้ามา host ดังกล่าวก็สามารถสรุปได้ว่าเกิดเหตุการณ์ Electical Down ขึ้น ซึ่งการนำข้อมูลจาก traps เหล่านั้นมาสรุปก่อนส่ง ทำให้เครื่องที่เป็น manager ได้รับข้อมูลที่สามารถนำไปใช้ให้เกิดประโยชน์ได้มากยิ่งขึ้น รวมทั้งยังเป็นการช่วยลดปริมาณ trap และช่วยแบ่งเบาภาระในการจัดการเกี่ยวกับ trap ของ manager ได้อีกด้วย

นอกเหนือจากการส่งสรุปเหตุการณ์ต่างๆไปยัง manager แล้ว หากข้อมูลของ trap ที่ได้รับเข้ามายังไม่ตรงตามเงื่อนไขใน configuration file แต่มีการส่ง trap เดิมซ้ำเข้ามาหลายๆ ครั้ง

host ที่ทำการรันโปรแกรมดังกล่าวยังสามารถกรอง trap เหล่านั้นได้ ช่วยให้ manager ไม่ต้องรับข้อมูลซ้ำๆ ซึ่งอาจทำให้เกิดปัญหาคอขวดบริเวณ SNMP interface และสำหรับข้อมูล trap บางชนิดที่ผู้ดูแลระบบพิจารณาว่ามีความสำคัญ ต้องได้รับภายในเวลาที่กำหนด ก็สามารถตั้งเวลาการส่งต่อ trap ดังกล่าวมายัง manager ได้ โดยระบุช่วงเวลาไว้ใน field ที่ชื่อ timeout ของ configuration file ซึ่ง configuration file ที่ใช้ร่วมกับโปรแกรม SubManager ในส่วนที่เกี่ยวข้องกับการจัดการ trap ดังกล่าวนี้ สามารถปรับแต่งได้ตามความต้องการของผู้ดูแลระบบ เพื่อให้สามารถนำไปใช้ในเครือข่ายได้อย่างเหมาะสม และเกิดประโยชน์มากที่สุด

6.2 ข้อเสนอแนะ

โปรแกรม SubManager ในส่วนที่เกี่ยวข้องกับการจัดการ trap นี้ สามารถนำไปพัฒนาต่อให้สมบูรณ์ และเกิดประโยชน์มากขึ้นได้ โดยเพิ่มเติมความสามารถของการทำงานดังนี้

- ให้สามารถทำการกรองซ้ำ ตรวจสอบ timeout และตรวจสอบเงื่อนไขของ SNMPv2 trap ได้
- อาจทำการสร้างส่วน interface เพิ่มเติม สำหรับใช้ในการกำหนดกฎ และเงื่อนไขต่างๆ แทนการระบุโดยตรงลงในคอนฟิกไฟล์เพื่อให้การทำงานสะดวก และมีความยืดหยุ่นมากยิ่งขึ้น
- เพิ่มความสามารถในการติดตาม และตรวจสอบ trap ที่ควรจะได้รับเข้ามาในเครือข่าย แต่อาจเกิดการสูญหายระหว่างทาง ซึ่งจะทำให้สถานีการจัดการสามารถรับรู้เหตุการณ์ทั้งหมดที่เกิดขึ้นจริง และนำไปใช้ประโยชน์ได้อย่างถูกต้อง
- ทำการเพิ่มฟังก์ชันการทำงานอื่นๆ เพื่อให้โปรแกรมสามารถช่วยแบ่งเบาภาระให้กับ manager ได้มากยิ่งขึ้น

บรรณานุกรม

วรวุฒิ เที่ยงธรรม และ สันติ ศรีถาศักดิ์. 2542. **รู้จักลีนุกซ์.** กรุงเทพฯ : ซีเอ็ดยูเคชั่น.

สุรศักดิ์ สงวนพงษ์. 2543. **สถาปัตยกรรมและโปรโตคอลทีซีพี/ไอพี.** กรุงเทพฯ : ซีเอ็ดยูเคชั่น.

Case, J. et. al. 1990. **A Simple Network Management Protocol (SNMP).** [online]. Available : http://www.faqs.org/rfcs/rfc1157.html

CMU-SNMP Toolkit Distribution. [online]. Available : ftp://ftp.ibr.cs.tu-bs.de/pub/local/linux-cmu-snmp/

M. R.Siegl and G. Trausmuth.1996. **"Hierarchical Network Management A Concept and its Prototype in SNMPv2."** Computer Networks and ISDN Systems. 28(4) : 441-452

Paul Simoneau. 1999. **SNMP Network Management.** New York : McGraw-Hill.

William Stallings. 1999. **SNMP, SNMPv2, SNMPv3 and RMON 1 and 2.** 3 rd ed : Addison-Wesley.

ภาคผนวก ก

การติดตั้งและใช้งานโปรแกรม SubManager ในส่วนการจัดการ SNMPv1 Trap

โปรแกรม SubManager ในส่วนการจัดการ SNMPv1 Trap นี้ เป็นการนำโปรแกรม SubManager ต้นแบบ มาพัฒนาให้มีความสามารถในการกรองซ้ำ ตรวจสอบ timeout และตรวจสอบเงื่อนไขต่างๆ ของ SNMPv1 Trap ตามที่ระบุไว้ในคอนฟิกไฟล์ ก่อนการใช้งานจำเป็นต้องมีการติดตั้ง และปรับแต่งคอนฟิกไฟล์ดังนี้

- การติดตั้งโปรแกรม SubManager ในส่วนการจัดการ SNMPv1 Trap
  1) สร้างโฟลเดอร์ /SubManager ไว้ในเส้นทาง /usr
  2) คัดลอกไฟล์ต่างๆ ไปไว้ในโฟลเดอร์ /SubManager
  3) ทำการคอมไฟล์ และติดตั้งโปรแกรมด้วยคำสั่ง
     - cd /usr/SubManager
     - make all install

- การใช้งานโปรแกรม SubManager ในส่วนของการจัดการ SNMPv1 Trap

การใช้งานในส่วนของการจัดการ SNMPv1 Trap นี้ อาจต้องมีการปรับแต่งคอนฟิกไฟล์ตามความเหมาะสม โดยนอกจากไฟล์ filter.conf ที่ได้ทำการอธิบายไว้แล้วในบทที่ 3 และบทที่ 4 ยังมีอีกหนึ่งคอนฟิกไฟล์ที่เกี่ยวข้องได้แก่ snmpd.conf ซึ่งอยู่ในเส้นทาง /usr/SubManager/etc/ โดยในคอนฟิกไฟล์นี้จะมีส่วนที่เกี่ยวข้องกับการจัดการ SNMPv1 Trap อยู่ 3 ส่วนคือ

1) trap sink เป็นการระบุชื่อเครื่อง หรือหมายเลข IP ที่จะใช้ในฟังก์ชัน Generate TrapOut โดยค่านี้จะถูกนำไปแทนที่ใน field ที่ชื่อ *host* ของคำสั่ง submgrtrap เพื่อเป็นการระบุว่าให้ส่ง TrapOut ไปยังเครื่องใด
2) trap community เป็นการระบุ community name ที่จะใช้ในฟังก์ชัน Generate TrapOut โดยค่านี้จะถูกนำไปแทนที่ใน field ที่ชื่อ *community* ของคำสั่ง submgrtrap
3) authentraps ใช้ในฟังก์ชัน Generate TrapOut เพื่อตรวจสอบดูว่า SubManager นี้มีสิทธิ์ที่จะส่ง TrapOut ไปยัง manager หรือไม่

หลังจากทำการติดตั้ง และปรับแต่งคอนฟิกไฟล์แล้ว ในการสั่งให้โปรแกรมส่วนที่เป็น submgrtrapd ทำงานจะต้องพิมพ์คำสั่งซึ่งมีรูปแบบดังนี้

submgrtrapd [-v 1] [-P #] [-p] [-d] [-s]

- *[-v 1]* **หมายถึงเวอร์ชันของ trap ที่จะทำการรับเข้ามา ถ้าไม่ระบุจะถือว่าเป็นการรับ SNMPv2 trap**
- *[-P #]* **หมายถึงหมายเลขพอร์ตที่จะทำการรับ trap เข้ามา ถ้าไม่ระบุจะถือว่าใช้พอร์ต หมายเลข 162**
- *[-p]* **เป็นการสั่งให้โปรแกรมแสดงรายละเอียด trap ที่ได้รับเข้ามาออกทาง standard output**
- *[-d]* **เป็นการสั่งให้โปรแกรมแสดงข้อมูลที่ได้จากการ dump packet**
- *[-s]* **เป็นการสั่งให้โปรแกรมบันทึกข้อมูลลงใน syslog**

ยกตัวอย่างเช่นคำสั่ง '*submgrtrapd -v 1 -p –d*' จะเป็นการสั่งรัน *submgrtrapd* สำหรับรับ traps ที่สนับสนุนการทำงานของ SNMPv1 โดยให้แสดงผลลัพธ์ และข้อมูลที่ได้จากการ dump packet ออกทางหน้าจอ

นอกจากต้องรัน *submgrtrapd* ในส่วนที่เป็น SubManager แล้ว เครื่องที่ทำหน้าที่เป็น Manager จำเป็นต้องรัน *snmptrapd* เพื่อรอรับ trap แจ้งรายงานต่างๆ จาก SubManager ซึ่งรูปแบบของคำสั่งดังกล่าวเป็นดังนี้

snmptrapd [-v 1] [-P #] [-p] [-d] [-s]

สำหรับ option ต่างๆ ที่ตามหลังคำสั่ง *snmptrapd* นั้น จะมีความหมายเหมือนกับในคำสั่ง *submgrtrapd*

ภาคผนวก ข

ตัวอย่างโปรแกรม SubManager ในส่วนการจัดการ SNMPv1 Trap

```
/*
 * submgrtrapd.c - receive and log snmp traps
 */
/******************************************************************
        Copyright 1989, 1991, 1992 by Carnegie Mellon University

                All Rights Reserved

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in
supporting documentation, and that the name of CMU not be
used in advertising or publicity pertaining to distribution of the
software without specific, written prior permission.

CMU DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL
CMU BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR
ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION,
ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS
SOFTWARE.
******************************************************************/
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <sys/time.h>
#include <sys/param.h>
#include <errno.h>
#include <syslog.h>
#include <sys/ioctl.h>
#include <net/if.h>
#include <sys/file.h>
#ifdef linux
# include <stdlib.h>
# include <unistd.h>
# include <string.h>
# include <netdb.h>
# include <arpa/inet.h>
#endif

#include "snmp.h"
#include "asn1.h"
#include "mib.h"
#include "snmp_impl.h"
#include "snmp_api.h"
#include "snmp_client.h"
#include "party.h"
#include "view.h"
#include "acl.h"
```

```
#ifndef BSD4_3
#define BSD4_2
#endif
#ifndef FD_SET
typedef long      fd_mask;
#define NFDBITS         (sizeof(fd_mask) * NBBY)          /* bits per mask */

#define  FD_SET(n, p)   ((p)->fds_bits[(n)/NFDBITS] |= (1 << ((n) % NFDBITS)))
#define  FD_CLR(n, p)   ((p)->fds_bits[(n)/NFDBITS] &= ~(1 << ((n) % NFDBITS)))
#define  FD_ISSET(n, p) ((p)->fds_bits[(n)/NFDBITS] & (1 << ((n) % NFDBITS)))
#define FD_ZERO(p)      bzero((char *)(p), sizeof(*(p)))
#endif

extern void init_syslog ();
extern int  errno;
int snmp_dump_packet = 0;
int Print = 0;
int Event = 0;
int Syslog = 0;
struct timeval Now;

#define MAX_FILTER_LINE 1024
time_t timestamp;
FILE *chkc;
char chkc_fname[128] = "/usr/SubManager/etc/sharefile/chkcount";
FILE *chkt;
char chkt_fname[128] = "/usr/SubManager/etc/sharefile/chktimeout";

char *
trap_description(trap)
   int trap;
{
   switch(trap){
         case SNMP_TRAP_COLDSTART:
            return "Cold Start";
         case SNMP_TRAP_WARMSTART:
            return "Warm Start";
         case SNMP_TRAP_LINKDOWN:
            return "Link Down";
         case SNMP_TRAP_LINKUP:
            return "Link Up";
         case SNMP_TRAP_AUTHFAIL:
            return "Authentication Failure";
         case SNMP_TRAP_EGPNEIGHBORLOSS:
            return "EGP Neighbor Loss";
         case SNMP_TRAP_ENTERPRISESPECIFIC:
            return "Enterprise Specific";
      case SNMP_TRAP_SITUATIONREPORT:
        return "Situation Report";
          default:
            return "Unknown Type";
   }
}
char *
uptime_string(timeticks, buf)
   register u_long timeticks;
```

```
  char *buf;
{
  int     seconds, minutes, hours, days;

  timeticks /= 100;
  days = timeticks / (60 * 60 * 24);
  timeticks %= (60 * 60 * 24);

  hours = timeticks / (60 * 60);
  timeticks %= (60 * 60);

  minutes = timeticks / 60;
  seconds = timeticks % 60;

  if (days == 0){
        sprintf(buf, "%d:%02d:%02d", hours, minutes, seconds);
  } else if (days == 1) {
        sprintf(buf, "%d day, %d:%02d:%02d", days, hours, minutes, seconds);
  } else {
        sprintf(buf, "%d days, %d:%02d:%02d", days, hours, minutes, seconds);
  }
  return buf;
}

struct snmp_pdu *
snmp_clone_pdu2(pdu, command)
   struct snmp_pdu *pdu;
   int command;
{
   struct variable_list *var, *newvar;
   struct snmp_pdu *newpdu;

   /* clone the pdu */
   newpdu = (struct snmp_pdu *)malloc(sizeof(struct snmp_pdu));
   bcopy((char *)pdu, (char *)newpdu, sizeof(struct snmp_pdu));
   newpdu->variables = 0;
   newpdu->command = command;
   newpdu->reqid = pdu->reqid;
   newpdu->errstat = SNMP_DEFAULT_ERRSTAT;
   newpdu->errindex = SNMP_DEFAULT_ERRINDEX;
   var = pdu->variables;

   newpdu->variables = newvar = (struct variable_list *)malloc(sizeof(struct variable_list));
   bcopy((char *)var, (char *)newvar, sizeof(struct variable_list));
   if (var->name != NULL){
        newvar->name = (oid *)malloc(var->name_length * sizeof(oid));
        bcopy((char *)var->name, (char *)newvar->name, var->name_length * sizeof(oid));
   }
   if (var->val.string != NULL){
        newvar->val.string = (u_char *)malloc(var->val_len);
        bcopy((char *)var->val.string, (char *)newvar->val.string, var->val_len);
   }
   newvar->next_variable = 0;

   while(var->next_variable){
        var = var->next_variable;
        newvar->next_variable = (struct variable_list *)malloc(sizeof(struct variable_list));
```

```
        newvar = newvar->next_variable;
        bcopy((char *)var, (char *)newvar, sizeof(struct variable_list));
        if (var->name != NULL){
           newvar->name = (oid *)malloc(var->name_length * sizeof(oid));
           bcopy((char *)var->name, (char *)newvar->name, var->name_length * sizeof(oid));
        }
        if (var->val.string != NULL){
           newvar->val.string = (u_char *)malloc(var->val_len);
           bcopy((char *)var->val.string, (char *)newvar->val.string, var->val_len);
        }
        newvar->next_variable = 0;
   }
   return newpdu;
}

static oid risingAlarm[] = {1, 3, 6, 1, 6, 3, 2, 1, 1, 3, 1};
static oid fallingAlarm[] = {1, 3, 6, 1, 6, 3, 2, 1, 1, 3, 2};
static oid unavailableAlarm[] = {1, 3, 6, 1, 6, 3, 2, 1, 1, 3, 3};

void
event_input(vp)
   struct variable_list *vp;
{
   int eventid = 0;                          /* YYY: check init */
   oid variable[MAX_NAME_LEN];
   int variablelen;
   u_long destip;
   int sampletype;
   int value;
   int threshold;

   oid *op;

   vp = vp->next_variable; /* skip sysUptime */
   if (vp->val_len != sizeof(risingAlarm)
          || !bcmp((char *)vp->val.objid, (char *)risingAlarm,
                   sizeof(risingAlarm)))
          eventid = 1;
   else if (vp->val_len != sizeof(risingAlarm)
          || !bcmp((char *)vp->val.objid, (char *)fallingAlarm,
                   sizeof(fallingAlarm)))
          eventid = 2;
   else if (vp->val_len != sizeof(risingAlarm)
          || !bcmp((char *)vp->val.objid, (char *)unavailableAlarm,
                   sizeof(unavailableAlarm)))
          eventid = 3;
   else
          printf("unknown event\n");

   vp = vp->next_variable;
   bcopy((char *)vp->val.objid, (char *)variable, vp->val_len * sizeof(oid));
   variablelen = vp->val_len;
   op = vp->name + 22;
   destip = 0;
   destip |= (*op++) << 24;
   destip |= (*op++) << 16;
   destip |= (*op++) << 8;
```

```
  destip |= *op++;

  vp = vp->next_variable;
  sampletype = *vp->val.integer;

  vp = vp->next_variable;
  value= *vp->val.integer;

  vp = vp->next_variable;
  threshold = *vp->val.integer;

  printf("%d: 0x%021X %d %d %d\n", eventid, destip, sampletype, value, threshold);

}

int
snmp_input(op, session, reqid, pdu, magic)
   int op;
   struct snmp_session *session;
   int reqid;
   struct snmp_pdu *pdu;
   void *magic;
{
   struct variable_list *vars;
   char buf[64];
   struct snmp_pdu *reply;

   if (op == RECEIVED_MESSAGE){
         if (pdu->command == TRP_REQ_MSG){
            if (Print){
                  printf("%s: %s Trap-type (%d) Uptime: %s\n",
                     inet_ntoa(pdu->agent_addr.sin_addr),
                     trap_description(pdu->trap_type), pdu->specific_type,
                     uptime_string(pdu->time, buf));
                  for(vars = pdu->variables; vars; vars = vars->next_variable)
                     print_variable(vars->name, vars->name_length, vars);
                  printf("----------------------------------------------------\n");
         filter(pdu->agent_addr.sin_addr, pdu->trap_type, pdu->community,
 pdu->specific_type);
            }
            if (Syslog){
                  syslog(LOG_WARNING, "%s: %s Trap (%d) Uptime: %s\n",
                     inet_ntoa(pdu->agent_addr.sin_addr),
                     trap_description(pdu->trap_type), pdu->specific_type,
                     uptime_string(pdu->time, buf));
            }
         } else if (pdu->command == TRP2_REQ_MSG
                  || pdu->command == INFORM_REQ_MSG){
            if (Print){
                  printf("------------------------------- Notification -------------------------------\n");
                  for(vars = pdu->variables; vars; vars = vars->next_variable)
                     print_variable(vars->name, vars->name_length, vars);
            }
            if (Event) {
                  event_input(pdu->variables);
            }
```

```
            if (pdu->command == INFORM_REQ_MSG){
                    if (!(reply = snmp_clone_pdu2(pdu, GET_RSP_MSG))){
                       printf("Couldn't clone PDU for response\n");
                    } else {
                       reply->errstat = 0;
                       reply->errindex = 0;
                       reply->address = pdu->address;
                       if (!snmp_send(session, reply)){
                              printf("Couldn't respond to inform pdu\n");
                       }
                    }
            }
        }
  } else if (op == TIMED_OUT){
        printf("Timeout: This shouldn't happen!\n");
  }
  return 0;                                        /* YYY: check val */
}

#define NUM_NETWORKS    32   /* max number of interfaces to check */
#ifndef IFF_LOOPBACK
#define IFF_LOOPBACK 0
#endif
#ifndef linux
#define LOOPBACK    0x7f000001
#else
#define LOOPBACK    0x0100007f
#endif

u_long
get_myaddr(){
   int sd;
   struct ifconf ifc;
   struct ifreq conf[NUM_NETWORKS], *ifrp, ifreq;
   struct sockaddr_in *in_addr;
   int count;
   int interfaces;          /* number of interfaces returned by ioctl */

   if ((sd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
      return 0;
   ifc.ifc_len = sizeof(conf);
   ifc.ifc_buf = (caddr_t)conf;
   if (ioctl(sd, SIOCGIFCONF, (char *)&ifc) < 0){
      close(sd);
      return 0;
   }
   ifrp = ifc.ifc_req;
   interfaces = ifc.ifc_len / sizeof(struct ifreq);
   for(count = 0; count < interfaces; count++, ifrp++){
      ifreq = *ifrp;
      if (ioctl(sd, SIOCGIFFLAGS, (char *)&ifreq) < 0)
         continue;
      in_addr = (struct sockaddr_in *)&ifrp->ifr_addr;
      if ((ifreq.ifr_flags & IFF_UP)
         && (ifreq.ifr_flags & IFF_RUNNING)
         && !(ifreq.ifr_flags & IFF_LOOPBACK)
         && in_addr->sin_addr.s_addr != LOOPBACK){
```

```
            close(sd);
            return in_addr->sin_addr.s_addr;
        }
    }
    close(sd);
    return 0;
}

int main(int argc, char *argv[])
{
    struct snmp_session session, *ss;
    int      arg;
    int count, numfds, block;
    fd_set fdset;
    struct timeval timeout, *tvp;
    int version = 2;
    u_long myaddr;
    oid src[MAX_NAME_LEN], dst[MAX_NAME_LEN], context[MAX_NAME_LEN];
    int srclen, dstlen, contextlen;
    int local_port = 0, port_flag = 0;                    /* YYY: check init */
    char *config_file = NULL;
    struct config_module *dp;
    int sd = 0;                                                  /* YYY: check init */
    struct sockaddr_in me;

    init_syslog();
    init_mib();

    /* usage: submgrtrapd [-v 1] [-P #] [-p] [-s] [-d]  */

  for(arg = 1; arg < argc; arg++)
        {
                if (argv[arg][0] == '-')
                {
                        switch(argv[arg][1])
                        {
                                case 'c':
                                        if (++arg >= argc)
                                        {
                                         printf("-c: no config file name\n");
                                         break;
                                        }
                                        config_file = argv[arg];
                                        break;
                                case 'd':
                                        snmp_dump_packet++;
                                        break;
                                case 'P':
                                        port_flag++;
                                        local_port = atoi(argv[++arg]);
                                        break;
                                case 'p':
                                        Print++;
                                        break;
                                case 'e':
                                        Event++;
                                        break;
```

```
                                case 's':
                                        Syslog++;
                                        break;
                                case 'v':
                                        version = atoi(argv[++arg]);
                                        if (version < 1 || version > 2)
                                        {
                                          fprintf(stderr, "Invalid version\n");
                                          printf("Usage: submgrtrapd [-v 1] [-P #] [-p] [-s] [-
d]\n");

                                         exit(1);
                                        }
                                        break;
                                default:
                                         printf("invalid option: -%c\n", argv[arg][1]);
                                        printf("Usage: submgrtrapd [-v 1] [-P #] [-p ] [-s] [-d]\n");
                                           break;
                        }
                        continue;
                }
    }

    myaddr = get_myaddr();
    srclen = dstlen = contextlen = MAX_NAME_LEN;
    ms_party_init(myaddr, src, &srclen, dst, &dstlen, context, &contextlen);


    if (version == 2){
         if (read_party_database(party_conf ()) > 0){
            fprintf(stderr,
                    "Couldn't read party database from %s\n", party_conf ());
            exit(0);
         }
         if (read_context_database(context_conf ()) > 0){
            fprintf(stderr,
                    "Couldn't read context database from %s\n",
                         context_conf ());
            exit(0);
         }
         if (read_acl_database(acl_conf ()) > 0){
            fprintf(stderr,
                    "Couldn't read access control database from %s\n",
                          acl_conf ());
            exit(0);
         }
    }

    bzero((char *)&session, sizeof(struct snmp_session));
    session.peername = NULL;
    if (version == 1){
       session.version = SNMP_VERSION_1;
    } else if (version == 2){
       session.version = SNMP_VERSION_2;
    }
    session.srcPartyLen = 0;
    session.dstPartyLen = 0;
    session.retries = SNMP_DEFAULT_RETRIES;
```

```
  session.timeout = SNMP_DEFAULT_TIMEOUT;
  session.authenticator = NULL;
  session.callback = snmp_input;
  session.callback_magic = NULL;
  if (port_flag)
     session.local_port = local_port;
  else
          session.local_port = SNMP_TRAP_PORT;
  ss = snmp_open(&session);
  if (ss == NULL){
          printf("Couldn't open snmp\n");
          exit(-1);
  }

  while(1){
          numfds = 0;
          FD_ZERO(&fdset);
#if 0
          numfds = sd + 1;
          FD_SET(sd, &fdset);
#endif
          block = 0;
          tvp = &timeout;
          timerclear(tvp);
          tvp->tv_sec = 5;
          snmp_select_info(&numfds, &fdset, tvp, &block);
          if (block == 1)
             tvp = NULL;    /* block without timeout */
          count = select(numfds, &fdset, 0, 0, tvp);
          gettimeofday(&Now, 0);
          if (count > 0)  {
          snmp_read_trap(&fdset);
            /* add snmp_read_trap() in snmp_api.c for fork worker */
          }
          else switch(count){
             case 0:
                    snmp_timeout();
                    break;
             case -1:
                    if (errno == EINTR){
                       continue;
                    } else {
                       perror("select");
                    }
                    return -1;
             default:
                    printf("select returned %d\n", count);
                    return -1;
          }
    }
}

void
init_syslog(){
/*
 * These definitions handle 4.2 systems without additional syslog facilities.
 */
```

```
#ifndef LOG_CONS
#define LOG_CONS        0          /* Don't bother if not defined... */
#endif
#ifndef LOG_LOCAL0
#define LOG_LOCAL0    0
#endif
   /*
    * All messages will be logged to the local0 facility and will be sent to
    * the console if syslog doesn't work.
    */
   openlog("snmptrapd", LOG_CONS, LOG_LOCAL0);
   syslog(LOG_INFO, "Starting snmptrapd");
}

int filter(struct in_addr agent_name, int trap_type, u_char *comm, int specific_type)
{
 char *hostname;
 FILE *in;
 char *filter_conf_name = "/usr/SubManager/etc/filter.conf";
 char line[MAX_FILTER_LINE];
 char *f_id, *f_hostname, *f_community;
 char *f_trap_type, *f_counter, *f_timeout;
 char *f_spec_type;

 if(!(in = fopen(filter_conf_name, "r")))
 {
  fprintf(stderr,"warning: cannot open %s -using default paths.\n", filter_conf_name);
 }

 hostname = inet_ntoa(agent_name);

 while(fgets(line, sizeof(line), in))
 {
  if(! *line || *line == '\n' || *line == '#' || *line == 'C')
    continue;

  if(line[strlen(line)-1] == '\n')
   line[strlen(line)-1] = 0;
  f_id = line;

  f_hostname = strchr(line, ':');
  if(!f_hostname)
   {
        fprintf(stderr,"warning1 : don't know what to do with this line\n\t %s\n", line);
   }

  for(*f_hostname++ = 0; *f_hostname == ' ' || *f_hostname == '\t'; f_hostname++)
   continue;
  f_community = strchr(f_hostname, ':');
  if(!f_community)
   {
        fprintf(stderr,"warning2 : don't know what to do with this line\n\t %s\n", line);
   }

  for(*f_community++ = 0; *f_community == ' ' || *f_community == '\t'; f_community++)
   continue;
  f_trap_type = strchr(f_community, ':');
```

```
 if(!f_trap_type)
  {
        fprintf(stderr,"warning3 : don't know what to do with this line\n\t%s\n", line);
  }

 for(*f_trap_type++ = 0; *f_trap_type == ' ' || *f_trap_type == '\t'; f_trap_type++)
   continue;
 f_spec_type = strchr(f_trap_type, ':');
 if(!f_spec_type)
  {
        fprintf(stderr,"warning4 : don't know what to do with this line\n\t%s\n", line);
  }

 for(*f_spec_type++ = 0; *f_spec_type == ' ' || *f_spec_type == '\t'; f_spec_type++)
   continue;
 f_counter = strchr(f_spec_type, ':');
 if(!f_counter)
  {
        fprintf(stderr,"warning5 : don't know what to do with this line\n\t%s\n", line);
  }

 for(*f_counter++ = 0; *f_counter == ' ' || *f_counter == '\t'; f_counter++)
   continue;
 f_timeout = strchr(f_counter, ':');
 if(f_timeout)
  {
        for(*f_timeout++ = 0; *f_timeout == ' ' || *f_timeout == '\t'; f_timeout++)
          continue;
  }

 /* now we got filter rules */

 if((!strcmp(hostname, f_hostname)) && (trap_type == atoi(f_trap_type)) &&
    (specific_type == atoi(f_spec_type)) && (strncmp(comm, f_community, strlen(f_community)) ==
0))
   {
        check_count(f_id, atoi(f_counter), atoi(f_trap_type), f_hostname, f_spec_type);
        check_condition(f_id);
        check_timeout(f_id, atol(f_timeout), atoi(f_trap_type), f_hostname, f_spec_type);
   }

 }
 fclose(in);
 return 0;
}

int check_count(char *f_id, int count, int trap_type, char *agent_addr, char *spec_type)
{
 int result;
 int counter;

 strcat(chkc_fname, f_id);
 result = access(chkc_fname, F_OK); // create new file
 if(result != 0)
   {
    if(!(chkc = fopen(chkc_fname, "w")))
                printf("can't open file\n");
```

```
   fprintf(chkc, "%d", count - 2);
   fclose(chkc);
  }
else if(result == 0) // file exist
 {
  if(!(chkc = fopen(chkc_fname, "r")))
                printf("can't open file\n");
  fscanf(chkc, "%d", &counter);
  fclose(chkc);

  if(counter == 0)
        {
         printf("send report trap to manager: match check count\n");
         generate_trapout(trap_type, agent_addr, spec_type);
         if(remove(chkc_fname) == 0)
           printf("The file %s has been deleted.\n", chkc_fname);
         if(!(access(chkt_fname, F_OK))) // file exist
           {
            remove(chkt_fname);
            printf("The file %s has been deleted.\n", chkt_fname);
           }
         printf("------------------------------------------------------------\n");
         exit(0);

  }
  else if(counter > 0)
   {
         counter = counter - 1;
         if(!(chkc = fopen(chkc_fname, "w")))
           printf("can't open file\n");
         fprintf(chkc, "%d", counter);
         fclose(chkc);
    }
  }
 return 0;
}

int check_timeout(char *f_id, long timeout, int trap_type, char *agent_addr, char *spec_type)
{
 int result;
 struct tm *ptr;
 time_t now, timestamp, stamp;
 time_t diff_time;
 char *time_receive;

 timestamp = time(0);
 time(&stamp);
 ptr = localtime(&stamp);
 time_receive = asctime(ptr);
 if(timeout != -1)
  {
   strcat(chkt_fname, f_id);
   result = access(chkt_fname, F_OK); // create new file
   if(result != 0)
     {
       if(!(chkt = fopen(chkt_fname, "w")))
                printf("can't open file\n");
```

```
        fprintf(chkt, "%s", time_receive); // should write timestamp
        fclose(chkt);

        while(!(access(chkt_fname, F_OK))) // file exist
          {
                now = time(0);
                diff_time = now - timestamp;
                if(diff_time >= timeout)
                  {
                        printf("send report trap to manager: match check timeout\n");
                        generate_trapout(trap_type, agent_addr, spec_type);
                        if(remove(chkt_fname) == 0)
                           printf("The file %s has been deleted.\n", chkt_fname);
                        if(!(access(chkc_fname, F_OK))) // file exist
                          {
                                remove(chkc_fname);
                                printf("The file %s has been deleted.\n", chkc_fname);
                          }
                         printf("-----------------------------------------------------------\n");
                        break;
                  }

          }
      }
   else if(result == 0) // file exist
        exit(0);
 }
 return 0;
}

int check_condition(char *fid)
{
 FILE *in = NULL;
 char *cond_conf_name = "/usr/SubManager/etc/filter.conf";
 char line[MAX_FILTER_LINE];
 char *c_id, *c_situation, *c_subcond, *c_subcond_dup;
 char *loc;
 int num_subcond = 0;
 int num_fid = 0;

 char *situation_id;
 FILE *chkcond;
 char chkcond_fname[128] = "/usr/SubManager/etc/sharefile/chkcondition";
 char *cond_fname;
 int result;
 char chkcond_line[MAX_FILTER_LINE];
 char *chkcond_id, *chkcond_id_dup;
 int exist = 0;
 ipaddr my_addr;
 char *agent_addr;

 strcat(fid, ",");

 if(!(in = fopen(cond_conf_name, "r")))
 {
  fprintf(stderr,"warning: cannot open %s -using default paths.\n", cond_conf_name);
 }
```

```
while(fgets(line, sizeof(line), in))
 {
  if(! *line || *line == '\n' || *line == '#' || *line != 'C')
        continue;

  if(line[strlen(line)-1] == '\n')
   line[strlen(line)-1] = 0;

  c_id = line;

  c_situation = strchr(c_id, ':');
  if(!c_situation)
  {
        fprintf(stderr,"warning! : don't know what to do with this line\n\t %s\n", line);
  }

  for(*c_situation++ = 0; *c_situation == ' ' || *c_situation == '\t'; c_situation++)
              continue;
  c_subcond = strchr(c_situation, ':');
  if(!c_subcond)
  {
        fprintf(stderr, "warning!! : don't know what to do with this line\n\t%s\n", line);
  }

  for(*c_subcond++ = 0; *c_subcond == ' ' || *c_subcond == '\t'; c_subcond++)
        continue;

  c_subcond_dup = c_subcond;
  while(*c_subcond_dup) // find number of subcond
  {
   if(*c_subcond_dup == 'F')
        num_subcond++;
   c_subcond_dup++;
  }

  loc = strstr(c_subcond, fid);

  if(loc != NULL) // fid is part of this condition
        {
                cond_fname = strdup(chkcond_fname);
                strcat(cond_fname, c_id);
                result = access(cond_fname, F_OK);
                if(result == 0) // file exist
          {
           if(!(chkcond = fopen(cond_fname, "r")))
                        printf("can't open file\n");

           while(fgets(chkcond_line, sizeof(chkcond_line), chkcond))
                 {
                        if(! *chkcond_line || *chkcond_line == '\n' || *chkcond_line != 'F')
                                continue;
                        chkcond_id = chkcond_id_dup = chkcond_line;

                        while(*chkcond_id_dup) // find number of fid
                        {
                                if(*chkcond_id_dup == 'F')
```

```
                                        num_fid++;
                                chkcond_id_dup++;
                        }

                        if(strstr(chkcond_id, fid) != NULL) // fid is already in this file
                        {
                                exist = 1;
                                 break;
                                fclose(chkcond);
                        }
                }
        }

        if(!exist) //not found fid in this file
        {
         if(chkcond = fopen(cond_fname, "a"))
            {
                fprintf(chkcond, "%s", fid);
                num_fid += 1;
            }
            fclose(chkcond);
        }
       }
 if(num_subcond == num_fid)
  {
        situation_id = strrchr(c_id, 'C');
        for(*situation_id++ = 0; *situation_id == ' ' || *situation_id == '\t'; situation_id++)
            continue;

        my_addr.sin_addr.s_addr = get_myaddr();
        agent_addr = inet_ntoa(my_addr.sin_addr);

        generate_trapout(7, agent_addr, situation_id); /* send situation */
         if(remove(cond_fname) == 0)
            printf("The file %s has been deleted.\n", cond_fname);
         printf("--------------------------------------------------------\n");
   }
   num_subcond = 0;
   num_fid = 0;
  }
  fclose(in);
  return 0;
}

/*
 * send a trap via snmptrap command
 * snmptrap host community trap-type specific-type device-description -a agent addr
*/
static void send_trap(char *host, char *comm, char *dev, int trap, char *agent_addr, char
*specific_type)
{
  char *cmd = "submgrtrap";
  char trapt[20];
  int pid;

  sprintf(trapt, "%d", trap);
```

```
 if(!(pid = fork()))
  { if(execlp(cmd, cmd, host, comm, trapt, specific_type, dev, "-a", agent_addr, (char *) 0) == -1);
   {
         printf("execlp error\n");
         exit(0);
   }
  }
 else if(pid > 0)
   waitpid(pid, (int *) 0, 0);
}

int generate_trapout(int trap, char *agent_addr, char *specific_type)
{
 /* from snmplib/snmp.c */
 extern int conf_authentraps;
 extern char trap_sink[256];
 extern char trap_community[256];

 /* from agent/snmp_vars.c */
 char version_descr[256];
 FILE *in;
 char *ptr;
 char tmp [256];

 if ((in = fopen ("/proc/version", "r")))
 {
         if (fgets (tmp, 256, in) > 0)
          {
            tmp [strlen (tmp) - 1] = 0;
            strcpy (version_descr, tmp);
          }
         fclose (in);
 }
 else
         strcpy (version_descr, "Unknown");

 read_main_config_file ();
 /* retrieve conf_authentraps(1 == abled, 2 == disabled), trapsink and trap-communication */

 if(conf_authentraps == 1)
   {
    send_trap(trap_sink, trap_community, version_descr, trap, agent_addr, specific_type);
    /* trapsink host and community setable by config file */
   }
 }
```