다중 쓰레드를 이용한 NMS Platform 설계 및 구현

Design and Implementation of NMS Platform using Multithread

김 성 호
㈜스마트넷 테크놀로지 기술연구소
shkim@smartnet.co.kr

요 약

인터넷의 급속한 발전으로 인해 TCP/IP 망에서 관리하고자 하는 대상 시스템이 갈수록 복잡하고, 그 수가 증가함에 따라 망 관리에 대한 중요성이 날로 더해가고 있으며 이를 관리하기 위한 프로토콜로 SNMP 가 사용되고 있으며 이는 관리자(Manager)와 대리인(Agent)모델을 기본으로 하는데 본 논문에서는 기존의 관리자 즉 NMS 플랫폼의 문제점 및 성능향상을 위해 다중 쓰레드를 이용하여 NMS 플랫폼을 새로이 설계하여 구현하였다. 기존 NMS 플랫폼의 형태는 다중의 대리인을 관리하기 위해 UNIX 의 fork 를 이용하여 관리하거나 또는 순차적 폴링로 하나의 대리인에게 폴링(polling)한 후 응답이 오면 다른 대리인을 다시 폴링하는 형태를 취하였다. 여기서 fork 를 이용하는 방법은 관리자가 각 대리인과 통신하기 위해 프로세스를 만드는데 있어 부모 프로세스의 data space, heap, stack 등을 자식 프로세스에게 복사하여 사용하므로 대리인수가 증가할 시 NMS Platform 의 메모리 증가를 초래하며 또한 각각의 프로세스간의 관계가 설정 되여 있으므로 중간 프로세스를 삭제할 시 각각의 프로세스를 관리하는데 많은 어려움이 있으며 전체 성능에 영향을 미치게 된다, 또 순차적형태의 폴링은 하나의 대리인의 응답이 너무 늦게 오거나 그 응답을 기다리는 동안 다른 대리인에게 문제 발생시 NMS 플랫폼에서 이를 감지하는데 문제가 발생할 수 있다. 이런 기존의 문제점을 수정하고, 성능을 향상시키기 위해 관리자가 각각의 대리인과 통신할 때 다중 쓰레드를 이용하여 통신하게 되면, 각각의 프로세스간에는 부모-자식 개념이 없이 동일한 프로세스 형태로 존재하게 됨에 따라 프로세스 관리가 용이하고, 메모리 복사 부분이 없이 공유메모리 기법을 사용하므로 NMS 플랫폼 자원을 효율적으로 사용하게 되어 전체 NMS 플랫폼의 성능에 많은 향상을 가져왔다.

1. 서 론

대규모의 네트워크를 효율적으로 관리하기 위하여, 통신망 관련업체와 표준화 기관이 상호 협동하여 많은 연구를 수행하고 있다. 이러한 연구들은 크게 두가지 연구로 분류될 수 있는데 , 첫째는 SNMP(Simple Network Management Protocol)계열로서, 이는 IAB(Internet Activity Board)에서 TCP/IP 네트워크의 관리 표준으로 1989 년에 처음으로 발표하였으며 현재는 SNMPv3 까지 발표 되었다, 둘째는 ISO (International Organization for Standard-ization)의 OSI(Open System Inter-connection) 네트워크를 위한 관리서비스와 프로토콜인 CMIS/CMIP 과 관리정보구조에 관한 연구가 활발히 진행중에 있다.

인터넷의 급속한 발전과 함께, 새로운 형태의 비즈니스인 MSP(Managing Service Provider) 와 ISP 등에서 사용자의 QoS 를 만족시키기 위한 SLM(Service Level Management)의 중요성이 부각되고 있고 있으며 이는 모두 TCP/IP 망에서 SNMP

Protocol 를 이용한 NMS 플랫폼의 성능 및 안정성에 따라 이를 구현할 수 있기 때문에 본 논문에서는 SNMP 를 이용한 NMS 플랫폼 즉 관리자(Manager)를 다중 쓰레드를 이용하여 설계 및 구현함으로써 기존의 방식보다 성능적인 측면과 안정성을 확보하도록 하였다.

(그림 1)은 SNMP 망 관리의 기본 모델로 관리자와 대리인 모델을 보여주고 있다. 관리자와 대리인 모델은 관리시스템 즉 NMS 플랫폼과 대리인 즉 관리대상 장비로 구성된다. 관리시스템의 관리자 즉 NMS 플랫폼은 망 자원들의 여러 가지 자원들을 폴링을 통하여 관리 객체(Managed Object) 들을 수집하여 연산을 수행한다.

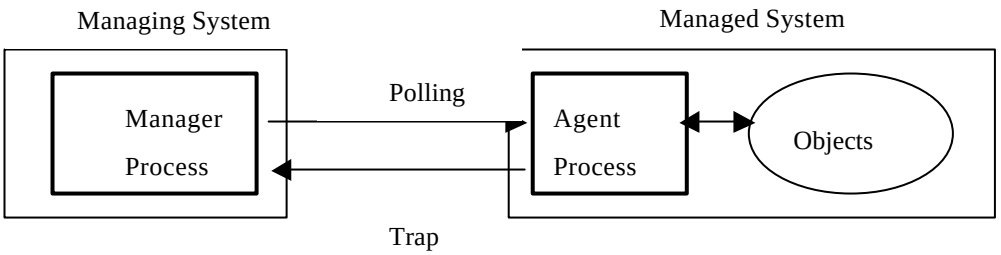
현재 가장 널리 쓰이는 SNMP Protocol 로는 CMU-SNMP 와 UCD-SNMP Protocol 이 있으며 UCD-SNMP 는 CMU-SNMP 프로토콜의 Upgrade 한 형태로 볼 수 있으며 이 두 연구는 SNMP Protocol 을 구현한 것으로 NMS Platform 의 형태는 아니다. 그러므로 이 두 프로토콜을 이용하여 NMS Platform 을 구현할 수 있다. 이때 기존의 순차적 폴링 기법과 fork 을 이용한 다중 프로세스 폴링 기법의 문제점을 알아보고 이를 극복하기 위한 새로운 기법을 제안한다.

NMS Platform 은 다수의 대리인과 각각의 다른 Interval 를 갖고 계속적으로 폴링을 하게 되며 , 또한 대리인의 시스템상 이상이 발생시 트랩을 통하여 NMS Platform 에게 알리게 된다. NMS Platform 구현 시 가장 어려운 부분이 다수의 대리인과 어떻게 통신하느냐가 NMS Platform 의 성능과 안정성에 가장 많은 영향을 미치게 된다. 이러한 형태는 그림 2 와 같다.

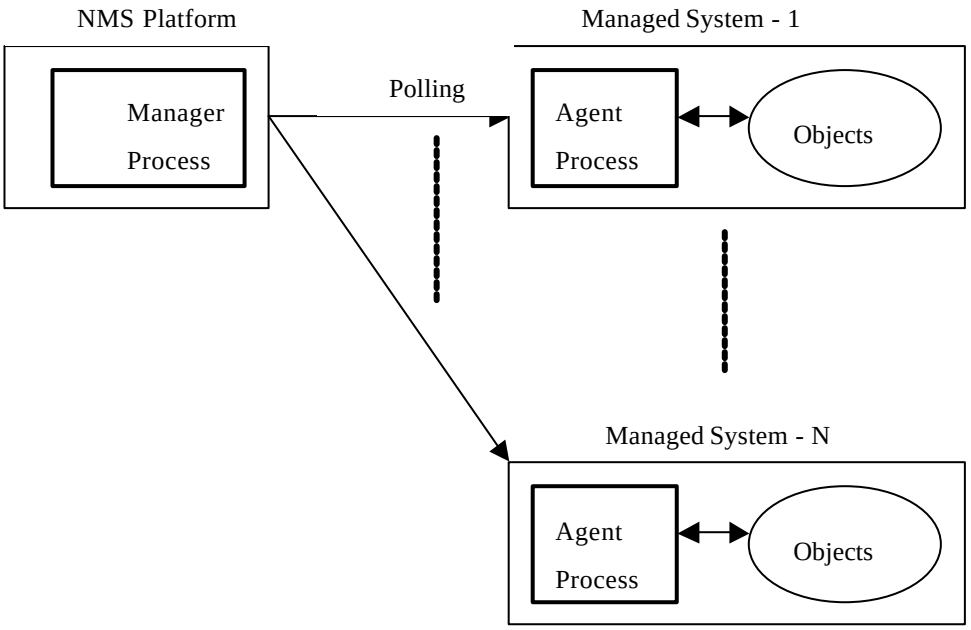
(그림 2)에서와 같이 NMS Platform 을 설계할 때 순차적인 폴링의 형태는 하나의 프로세스를 이용하여 대리인 1 을 폴링하고, 응답이 왔을 때 그 다음의 대리인 2 를 폴링하도록 하는 방식이다. 그렇게 해서 대리인 N 을 폴링했을 때 다시 대리인 1 으로 넘어가는 방식이 있고, 그 다음으로 다중 프로세스 즉 fork 를 이용하는 폴링하는 방식이 있다.

본 논문에서는 이 두가지 방법의 문제점을 다음장에서 서술하며 이 문제를 해결하기 위해 NMS Platform 의 형태를 다중 쓰레드로 설계하고 구현하는 것을 목표로 한다. 이렇게 함으로서, 각각의 대리인과 NMS Platform 과 설정된 폴링 Interval 을 정확히 지킬 수 있으며, 응용 프로그램의 응답시간을 향상시키고, 또한 공유메모리 기법을 사용함으로 플랫폼의 자원을 효율적으로 사용하게 되며, 프로그램의 구조가 객체지향설계의 클래스처럼 간단 명료하게 되어 프로그램의 수정을 용이하게 할 뿐 아니라, 대리인 수가 많을수록 관리해야 할 프로세스가 증가하게 되는데 이를 관리하는데 매우 용이하게 된다. 또한 본 논문에서는 NMS Platform 을 하나의 프로세서로 정의하여 설계 및 구현을 하였지만 관리 및 연산해야 될 객체가 많아질 때 각각의 프로세서로 쓰레드를 분산시켜 전체 시스템의 성능을 월등히 향상시킬 수 있는 장점을 가지고 있다.

본 논문에서 제안하는 NMS Platform 개발형태는 UCD-SNMP 프로토콜을 이용하여 Ultra 60 Solaris 운영체제 상에서 C 언어와 POSIX 쓰레드를 이용하여 구현하였다



(그림 1) 관리자와 대리인 모델



(그림 2) NMS Platform 의 폴링 형태

2. NMS Platform 에 대한 기존 연구

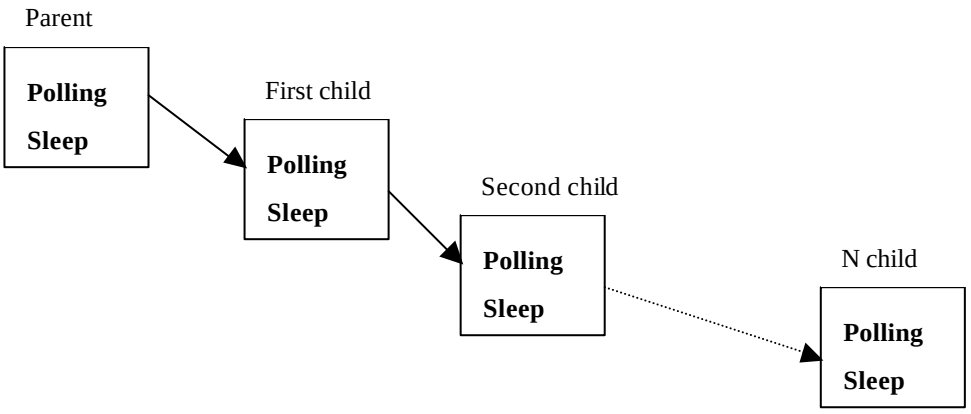
2.1 순차적 형태의 폴링 방법

NMS 플랫폼은 기본적으로 관리하고자 하는 N 개의 대리인과 폴링을 통하여 원하는 데이터를 수집하여 연산을 한다. 가장 기본적인 방법으로 대리인-1 과 폴링을 한 후 그 다음 대리인-2 와 폴링하여 데이터를 수집하고, 이렇게 대리인-N 까지 폴링한 후 다시 대리인-1 과 폴링을 하게 된다. 이때 NMS 플래폼의 구현상의 입장에서 보면 가장 쉬운 방법이지만 다음과 같은 문제점을 발생하게 된다. 첫째, 관리자 프로세스가 대리인-1 과 폴링할 때 응답이 오지 않거나, 너무 많은 객체를 폴링하게 되어, 그 데이터를 수신하는 동안 대리인-2 와의 폴링간격(Polling Interval)을 만족하지 못하는 경우가 발생한다. 이는 NMS 플랫폼의 신뢰성에 가장 큰 영향을 미치게 된다. 둘째, 관리자 프로세스와 대리인과의 폴링간격은 대리인마다 다를 수가 있다. 중요 대리인은 폴링간격을 짧게 주고, 덜 중요한 대리인에게는 폴링 Interval 을 길게 줄 수 있는데 이렇게 서로 다른 폴링간격을 관리자 프로세스는 보장할 수 없게 된다. 셋째로는 관리자 프로세스는 하나 이므로 하나의 대리인과 통신중에 오류가 생기면, 다른 대리인과의 연결상태도 유지할 수 없으므로 NMS 플랫폼의 신뢰도에 심각한 영향을 미치게 된다.

2.2 다중 프로세스를 이용한 폴링 방법

2.1 의 문제점은 하나의 관리자 프로세스가 N 개의 대리인과 통신하여 생기는 문제점이다 이를 극복하기 위해 관리자 프로세스는 N 개의 대리인만큼 프로세스를 생성하여 각각의 관리자 프로세스가 각각의 대리인과 통신하게 함으로서 각각의 대리인에게 부여된 폴링간격을 보장하게 된다. 이는 다음과 같은 문제점을 안고 있다. 첫째. 관리자 프로세스에서 자식 프로세스(Child Process)는 부모 프로세스(Parent Process)의 data space, heap, stack 등의 복사본을 유지하게 됨으로

관리하고자 하는 대리인의 수가 많을 때 NMS 플랫폼에 많은 메모리를 사용하게 된다. 둘째, UNIX 에서의 프로세스 수는 최대 256 개의 정의되어 있어 관리하고자 하는 대리인의 수는 한계를 가질 수 밖에 없다. 셋째, 관리자의 프로세스간에는 (그림 3)과 같이 서로의 relationship 이 존재함으로 프로세스 관리가 어렵게 된다.



(그림 3) 관리자 프로세스의 관계

(그림 3)의 각각의 관리자 프로세스는 각각의 대리인과 통신하게 된다. 이때 중간 프로세스의 하나만 오류가 발생하여도 수많은 좀비(Zombie) 프로세스가 만들어지게 되어 시스템에 악영향을 미치게 되고, 또한 관리하고자 하는 대리인은 상황에 때라 새로이 생기거나 삭제되게 되는데 이때 이들 프로세스 관리가 복잡하게 되어 프로그래머에게 많은 부담을 주게 된다.

3. NMS 플랫폼의 설계 및 구현

3.1 NMS 플랫폼의 설계

본 논문에서 제안하는 NMS 플랫폼의 다음과 같은 목표를 가지고 설계하였다

첫째, 병행 투명성을 제공함으로서 각 대리인과의 폴링간격 보장

NMS 플랫폼의 각각의 쓰레드는 상호 배제 문제를 해결하여 각각의 대리인에게 설정된 폴링 간격을 보장해야 한다.

둘째, 시스템 자원의 효율적 사용

각각의 쓰레드가 사용하는 메모리를 공유함으로써 시스템의 자원을 효율적으로 사용하여 시스템 전체의 성능을 향상시킨다.

셋째, 프로그램 구조의 향상

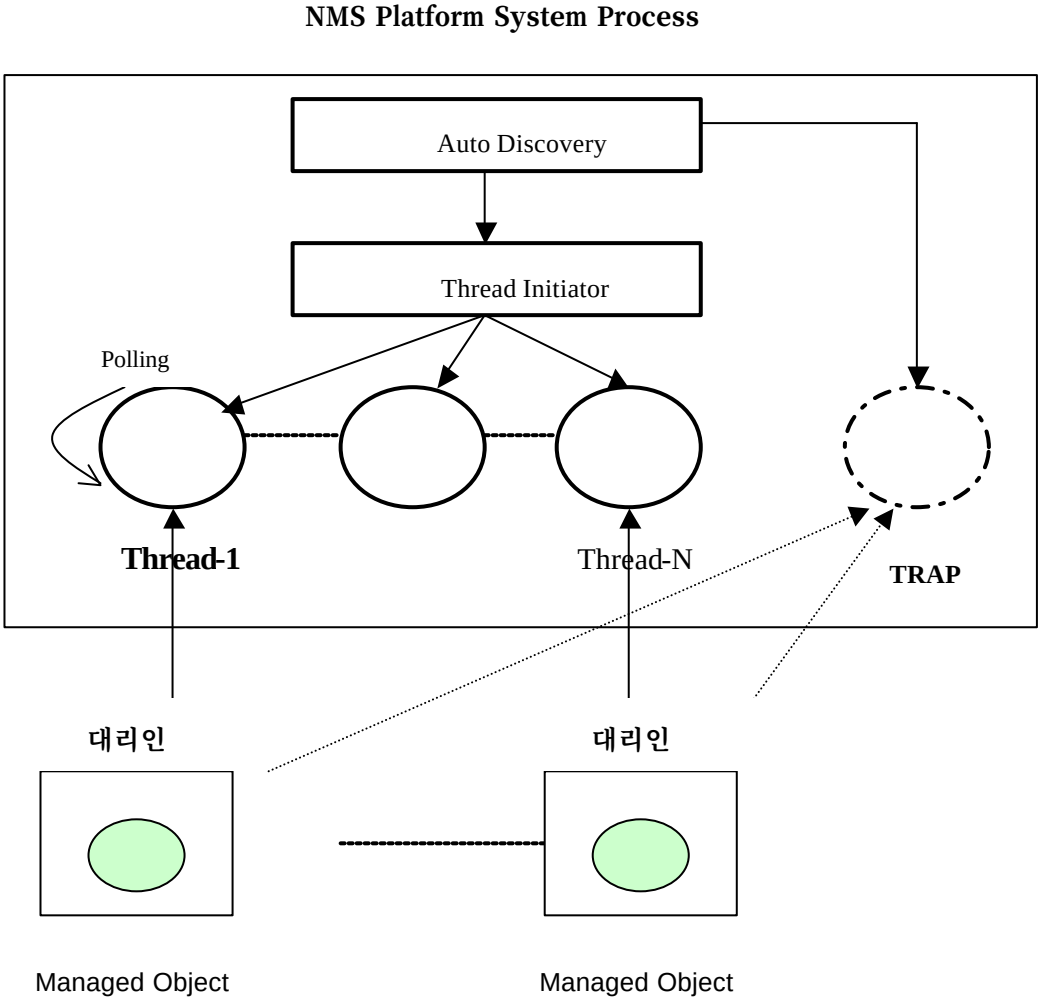
객체지향설계처럼 하나의 프로그램 구조를 선언하여 새로이 생성되는 쓰레드가 이를 사용하게 함으로서 전체 프로그램 구조를 쉽게 설계할 수 있어야 하며, 또한 새로운 객체를 선언할 때 쉽게 작성할 수 있어야 한다

넷째, 쓰레드 관리의 용이성

시간의 흐름에 따라 새로운 대리인이 생성되거나 기존의 대리인이 삭제될 수 있는데 이때 쓰레드의 생성과 삭제가 전체 시스템에 영향을 주지 않으며 관리되어야 한다.

3.2 NMS 플랫폼의 프로세스 구조

본 논문에서 제안하는 NMS 플랫폼의 프로세스 구조는 (그림 4)와 같으며 프로세스의 구조는 다중 쓰레드를 지원하며, 쓰레드간의 메모리는 전역메모리를 이용하여 상호 공유되도록 설계하였다.



(그림 4) 제안된 NMS 플랫폼의 프로세스 구조

Auto Discovery 에서는 관리하고자 하는 대리인을 주어진 IP Address Range 안에서 찾는다. 찾아진 IP Address 와 그 대리인의 특성을 Thread Initiator 에게 정보를 넘겨준다. Thread Initiator 는 그 정보를 이용하여 해당 쓰레드를 생성한다. 이때 원하는 폴링 간격에 대한 정보를 쓰레드에게 넘겨준다. 해당 쓰레드들은 관리하고자 하는 관리객체 function 을 Call 하면서 대상 대리인과 통신하게 된다. 각각의 쓰레드들은 독립적으로 실행되면서 주어진 정보에 따라 해당 대리인과 통신하면서 관리객체를 수집하여 연산한다. Thread Initiator 에서는 해당 대리인을 새로이 생성하는 Event 가 들어오면 이때 새로운 쓰레드를 생성하고, 삭제 Event 가 들어오면 해당 IP address 에 관련된 쓰레드를 삭제하는 기능을 갖는다. 각각의 쓰레드는 타이머 인터럽트를 이용하여 폴링 간격을 보장 받게 된다. 이때 각각의 쓰레드들간의 스케줄링은 라운드 로빈으로 스케줄링 한다. Thread Initiator 에서는 쓰레드 생성시기를 조절하기 위해 전체 대리인 수를 감안하여 각각의 쓰레드를 생성하게 한다. 이는 각각의 쓰레드가 동시에 폴링할 수 있는데 이때 전역메모리의 공유를 지키기 위해 상호 배제 메카니즘을 사용하지만 이는 프로세서의 성능을 저하시키므로, 가능한 한 이런 문제가 생기지 않도록 쓰레드의 생성시기를 조절한다. Auto Discovery 기능 수행 후 대리인의 Event 를 수집하기 위해 TRAP Daemon 을 Start 시킨

다. 대리인에서는 미리 정의된 TRAP 특성에 따라 Event를 발생시킨다.

3.3 구현

본 논문에서 제안한 NMS 플랫폼을 구현한 환경은 다음과 같다.

- 운영체제 : 솔라리스 2.8
- SNMP Protocol : UCD-SNMP
- 사용언어 : C
- 쓰레드 라이브러리 : PTHREAD

3.3.1 Auto Discovery 구현

Auto Discovery 의 구현은 NMS 플랫폼에서 가장 먼저 구현되어야 할 항목으로 관리하고자 하는 IP Address Range를 통해 대상 대리인을 찾아내며 여기에서 각각의 대리인에 대한 정보 즉 , 해당 IP address, 해당 장비의 형태, 폴링간격, 관리객체 등에 대한 정보를 Thread Initiator 에게 넘겨준다. Thread Initiator 에서는 이 정보를 이용하여 각각의 쓰레드를 생성할 때 쓰레드에게 정보를 넘겨준다. 또한 MIB-II 의 IP Group 에서 IpRoutingTable 을 이용하여 Next Hop 의 IP 주소를 알아내어 Router 간의 상호 연결 상태를 알아낸다.

```
int IP_Node_Search ( )
{
do {
   ipadd[0] = strtok(buf, delim);
 while(ipadd[++argc]
  =strtok(NULL,delim)) != NULL)
ipaddress1
  = (char )malloc(strlen(ipadd[0]));
    /* start ip address */
ipaddress2
  = (char *)malloc(strlen(ipadd[1]));
    /* end ip address */
community
  = (char )malloc(strlen(ipadd[2]));
    /* community string */
  } while(1);
host = head->next;
if(host == tail)
  return -1;
return 0;
}
```

위의 프로그램에서는 관리하고자 하는 대리인의 IP address 처음 시작 주소와 끝 시작주소를 저장하는 프로그램이다.

```
void Node_Find( )
{
struct snmp_session session *ss;
struct snmp_pdu *pdu, *response,
                    *pdunext;
struct socketadd_in *respIP;
struct variable_list *vars;
while(phost !=tail)
{
        snmp_sess_init(&session);
        session.peername
          = phost->hostname;
        session.peername
          = phost->community;
        session.timeout
          = DEFAULT_TIMEOUT);
        execute Polling
          with OID (IpRoutingTable);

}
}
```

위의 프로그램에서는 저장된 IP address Range 를 이용하여 각각의 대리인에게 폴링하게 된다. 이때 응답한 대리인을 관리하고자 하는 대상 노드로 정하며, 이때 IP RoutingTable 을 이용하여 라우터의 next Hop IP address 를 알 수 있으므로 이 정보를 이용하여 라우터간의 상호 연결상태를 구현한다.

3.3.2 Thread Initiator 구현

Thread Initiator 에서는 Auto Discovery 에서 넘겨준 정보, 즉 해당장비의 IP address, Community String, 장비의 형태, 폴링간격, 관리하고자 하는 대리인의 수 등의 정보를 이용하여 대리인 수 만큼 각각의 쓰레드를

생성하여 대리인과 폴링을 통하여 관리 객체에 대한 정보를 수집하고 연산하는 기능을 수행한다.

```
Thread Initiator ( )
{
    nCount = Node_Count( );
    Pthread_mutex_init( );
    Node = head ->next;
    For( I=0; I <nCount; ++I)
    {
        mutex_lock( );
        pthread_create(&thread[I],
        NULL,Polling_Thread,
        (void *) node);
        mutex_unlock();
        node = node ->next;
        Polling Frequency
        (? /nCount) ;
        /* Timer Interrupt */
    }
    for(I=0;I<nCount;++i )
    {
        pthread_join(threads[I],
        NULL); /*
    }
    return ;
}
```

위의 프로그램에서 nCount 는 관리 대상 장비의 수이며, 그 수 만큼 쓰레드를 생성하여 해당 쓰레드에게 노드에 대한 정보를 넘겨준다. 각각의 쓰레드의 생성시기를 만드는 시점을 각각의 평균 폴링간격에 전체 관리하고자 하는 대리인 수 만큼 나눈 시간만큼 sleep 한 후 새로운 쓰레드를 생성하게 했다. 이와 같은 이유는 각각의 쓰레드간의 공유메모리에 대한 mutex 부분을 최소화 하기 위함이다. 왜냐하면 mutex 를 시도하는 쓰레드가 많을수록 CPU 의 부하는 많이 걸리기 때문이다. 이를 통해 전체 시스템의 성능향상을 유도하였다. 또한 ? 는 각 장비의 평균 폴링 간격을 나타내는데 이것은 다음의 식에서 유도한다.

$$N <= \frac{15 * 60}{0.202} ? 4500 <= \frac{T}{?}$$

? : Average time required
to perform a single poll
T : Desired polling interval
N : Number of agents

여기서 ?는 네트워크 지연시간, 시스템의 응답시간 등 여러가지 요인에 의존한다. 위의 식은 TCP/IP 를 사용하는 LAN 상에서 processing time 이 50ms, network delay time 이 1ms, 폴링 간격을 15 분이라고 가정할 때 ?는 약 0.202sec 가 나오므로 위의 식대로 4500 장비를 NMS 플랫폼이 관리할 수 있다는 것을 의미한다. 이때 관리하고자 하는 대상의 수를 우리는 미리 알 수 있으므로 역으로 ? 시간을 도출할 수 있다.

위의 프로그램에서 pthread_join 을 사용한 이유는 NMS 플랫폼은 24 시간 365 일 계속해서 동작해야 함으로 main thread 가 데몬 형식으로 계속적으로 운영되도록 하기 위함이다. 이 main thread 가 종료가 되면 이를 통하여 만들어진 각각의 쓰레드들은 자동적으로 종료가 되기 때문이다.

```
Polling_Thread( *arg)
{
    pthread_t snmp_thr ;
    pthread_t ret_thr;
    pthread_setcancelstate
    (PTHREAD_CANCEL_ENABLE,
     NULL);
    pthread_setcanceltype
    (PTHREAD_CANCEL_DEFERRED,
     NULL);

    Start :
    pthread_mutex_lock( );
    Snmp_open( );
    pthread_mutex_unlock( );
    pdu
    = snmp_pdu_create(SNMP_MSG);
    snmp_add_null_var(pdu,OID);
    Receive Polling Data ( ) ;
```

```
        Snmp_close( );
        Sleep(?) ;
        Goto Start;
    }
```

위의 프로그램에서는 주어진 시간의 폴링 간격을 보장하면서 각각 생성된 쓰레드들이 관리하고자 하는 OID 정보를 대리인에게 메시지를 보내, 이에 응답한 결과를 가지고 연산을 통하여 원하는 정보를 만드는 것을 보여준다.

4. 테스트 베드 설치 후 성능평가

4.1 테스트 베드 환경

본 논문에서 제안된 NMS 플랫폼의 성능 측정을 위하여 ㈜스마트넷 테크놀로지 기술연구소에 전용으로 관리대상장비 20 대를 하나의 허브에 연결한 후 각각의 관리대상장비에 windows 98, NT, Linux 등을 설치하여 SNMP agent 가 운영되도록 하였으며, 외부망으로는 서울대 라우터를 관리대상장비로 선정하였다 . NMS 플랫폼에는 SUN 사의 Ultra 60 450Mz 의 CPU1 개와 메모리 512MB 를 가진 시스템으로, 제안된 NMS 플랫폼 프로그램과 UCD-SNMP Protocol 을 설치하였다.

4.2 성능평가

위의 테스트 베드 환경하에서 관리하고자 하는 객체는 MIB-II 에서 100 의 객체를 폴링하는 것으로 테스트 하였으며, 폴링 간격은 1 초부터 300 초까지 나누어 대리인과 폴링하였다. 이때 솔라리스의 Process Manager 를 이용하여 해당 프로그램의 메모리 사용량, CPU 사용율등을 매 1 초마다 얼마나 사용하는지 알아보았다. 성능평가후 폴링간격을 1 초로 했을 때 메모리 사용량은 2548Byte 를 사용하였으며, CPU 사용율은 0.3~0.5%를 사용하는 것으로 결과가 나왔으며, 폴링간격을 300 초로 하였을때는 메모리 사용량은 같았고, CPU 사용율이 0.0~0.1%를 사용하는 것으로 나왔다. 위의 성능평가 후 결론은 공유메모리를 사용하므로, 메모리의 사용량을 급격히 줄일 수 있었으며, 쓰레드의 생성시기를 결정하는 알고리즘을 사용하여 각 쓰레드간의 mutex 에서 생기는 CPU 부하를 줄일 수 있었으며, 또한 각각의 쓰레드에게 부여된 폴링 간격을 정확히 보장하여, 수집된 데이터의 신선도를 유지시켰으며, 시스템의 안정성도 높아졌음을 알 수 있었다.

5. 결 론

NMS 플랫폼을 개발할 때 다중 쓰레드를 사용함으로서, 메모리의 사용량을 급격히 줄일 수 있었으며, CPU 의 부하 또한 거의 주지 않는 것으로 결과가 나왔으며, 시스템의 안정성도 보장되었다. 단지 아쉬운 점이라면, 순차적 폴링 형태와 다중 프로세스의 형태로 구현한 후 성능평가를 같이 하지 못한 점이 아쉬움으로 남는다.

향후 남은 과제로는 현재의 플랫폼에 QoS 를 위한 DiffServ, RSVP, IntServ 의 MIB 을 구현하여 NMS 플랫폼에서 QoS 관리가 가능하도록 설계 및 구현하는 하고자 한다. 또한 요즘 초고속 라우터에서 지원하는 MPLS 의 기능을 관리할 수 있도록 하기 위해 NMS 플랫폼에서 이를 구현할 예정이다.

[참고문헌]

[1] J.D.Case and J.R.Davin. A Simple Network Management Protocol. Internet RFC1157, 1990

[2] William Staling. SNMP, SNMPv2, and Practical Network Management. Addison-wesley, 1996

[3] U.Black. Network Management Standards. McGraw-Hill, Inc., 1995

[4] B.Stewart. SNMP Application. Internet RFC2573, 1999

[5] Bil.Lewis. A Guide to mutithreaded programming. PTR ,1996

[6] Multithreaded Programming Guide. SUN microsystems , 2000

[7] UCD-SNMP Users Manual

[8] The Simple Times , SEPTEMBER 2000

김 성 호

1994 한국과학기술원
컴퓨터공학 석사
1995~1997 포항공대
정보통신연구소 NDS Lab
연구원
1997~2000 넷피아
컨설팅㈜ 대표이사
2000~ 현재 ㈜스마트넷 테크놀로지
기술연구소장
<관심분야> 네트워크 관리시스템,
QoS 관리,Traffic Engineering