네트웍 인터페이스를 위한 새로운 I/O 방식

전성원, 이해연, 조정완, 이흥규, 이준원, 윤현수
한국과학기술원
swjun@calab.kaist.ac.kr, hytoiy@casaturn.kaist.ac.kr

A New I/O Method for Network Interface

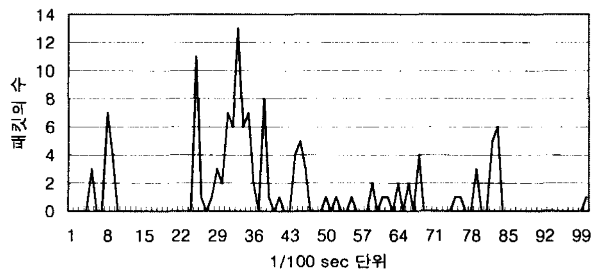
Sung-Won Jun, Hae-Yeoun Lee, Jung Wan Cho, Heung-Kyu Lee, Joonwon Lee Hyunsoo Yoon
KAIST

오늘날은 컴퓨터 하드웨어 기술이 발전함에 따라 컴퓨터 네트워크은 그 처리 속도와 처리 능력에 있어서 많은 발전이 있었다. 그러나 현재 네트워크를 사용하는 시스템은 이러한 특성을 최대한 효율적으로 이용하고 있지 않다. 초기부터 컴퓨터에 있어서 입출력 속도의 문제는 항상 시스템의 성능을 제한하는 요소였다. 따라서 이 문제의 해결 방법으로 인터럽트 기반의 입출력 방식이 나왔다. 그러나 현재는 새로운 하드웨어 기술이 발전함으로 인하여 네트워크의 속도가 매우 빨라져서 인터럽트에 의한 입출력 방식이 네트워크 인터페이스에 대해서는 시스템의 처리 속도를 떨어뜨리는 결과를 가지고 오고 있다.
그러므로 본 논문에서는 폴링 기반의 입출력 방식을 사용함으로써 기존의 인터럽트 방식보다 어느 정도 시스템의 성능을 향상 시킬 수 있는지 비교하기 위하여 폴링을 이론적으로 구현하고 비교한다. 그리고 이와 더불어 네트워크 인터페이스의 입출력을 위해 우리가 제시한 새로운 방법에 대한 소개와 그 효율성을 알아본다.

1. 서론*

컴퓨터 시스템의 입출력을 위한 방법은 인터럽트(Interrupt)와 폴링(Polling)의 두 가지로 나눌 수 있다. 두 방법의 특징을 보면, 인터럽트는 외부 기기에서 운영체제에게 어떤 사건의 발생을 알려주는 기능으로 인터럽트를 사용함으로써 운영체제는 외부 기기와 주기적으로 입출력을 요청할 필요 없이 다른 작업을 계속 수행할 수 있다. 이로 인하여 입출력 속도가 느린 디바이스의 경우에 CPU 의 처리 효율이 크게 향상된다. 그러나 폴링에 비해 수행해야 하는 코드의 크기가 크다. 이에 반해 폴링은 운영체제에서 주기적으로 외부 디바이스에게 어떤 사건이 일어났는지 물어보는 기능이다. 폴링 방식을 이용할 경우 운영체제는 외부 디바이스에게 입출력 장치에 대해 사건의 발생 여부를 주기적으로 물어봄으로써 어떤 사건이 발생하였는지 여부를 확인한다. 따라서 폴링은 사건이 없어도 운영체제가 주기적으로 작업을 해야 하므로 CPU 처리시간의 낭비가 있으나 입출력 빈도가 많은 경우 비교적 효율적으로 수행된다.

하드웨어 기술이 발달됨에 따라 네트워크의 처리 속도와 네트워크의 사용량이 급격히 증가하게 되었다. 이로 인하여 컴퓨터 시스템의 네트워크 인터페이스를 통한 입출력 처리 요구는 매우 빈번하게 일어나게 되었다. 따라서 이러한 시스템에 인터럽트를 사용하는 경우 인터럽트 처리에 들어가는 코드의 크기가 폴링을 사용하여 처리하는 코드에 비하여 매우 크기 때문에 오히려 시스템의 성능을 떨어뜨리는 결과를 나타내므로 폴링에 의한 방법이 적합할 것이다. 오늘날 네트워크 인터페이스를 통하여 들어오는 입출력 패킷(Packet)의 수는 매우 많고 빈번하게 나타나며 어떤 시간적인 지역성(Temporal Locality)이 있음을 알 수 있다. 본 논문에서는 이와 같이 입출력 사건이 빈번하게 발생하는 네트워크 인터페이스의 경우 인터럽트와 폴링의 방식에 의한 입출력 효율을 비교하고, 네트워크 인터페이스를 통해 들어오는 입출력 패킷의 시간적인 지역성을 이용하여 효과적으로 인터럽트와 폴링을 전환하는 새로운 입출력 방식을 제시하고 그 효율성을 시뮬레이션을 통해 평가하고자 한다.



< 그림 1.> 1 초간 도착한 패킷의 패턴

본 논문의 2 장에서는 인터럽트와 폴링의 전환을 위한 방식을 알

* 한국과학기술원 정보 시스템 연구소의 지원을 받음

아보며, 3 장에서는 알고리즘의 구현과 시뮬레이션에 대해 정리한 후, 4 장에서는 결과를 정리하고, 5 장에서 결론을 내린다.

2. 제안하는 새로운 I/O 방식

네트워크의 속도가 빨라지고, 그 사용 빈도가 많아질수록 네트워크 인터페이스는 사용하는 패턴의 특성상 인터럽트와 폴링을 전환하여 사용하는 것이 한가지만을 사용하는 것 보다 더 효율적일 수 있다. 이와 같이 효율적인 전환을 위해서는 인터럽트와 폴링의 두 policy 를 서로 전환(switching)하는 방법을 제안하고 시뮬레이션을 통해 그 성능을 평가하고자 한다. 본 논문에서는 제안한 인터럽트와 폴링의 전환 방법을 "IPSwitch"라고 하겠다.

제안한 IPSwitch 에서는 현재의 시점에서 이전에 도착한 패킷들의 history 를 이용한다. 기존 아키텍춰(architecture)를 고려하여 본 논문에서는 history 를 *n* (window size; 1 <= *n* <= 32) 개 고려하여 인터럽트와 폴링을 전환할 수 있는 방법을 소개한다.

2.1. 폴링에서 인터럽트로 전환

하나의 history 를 위한 변수에 매 폴링 시간마다 그 변수를 왼쪽으로 쉬프트(shift) 한다. 그리고 만일 패킷이 도착했을 경우 맨 하위 비트를 설정(set) 하고, 그렇지 않으면 그냥 다음으로 진행이 된다. 이때 history 변수에 원하는 window size n 으로 mask out 시킨 나머지 데이터의 값이 0 인지를 검사하여, 모두 0 이면 이전에 n 번의 폴링을 수행하는 동안 패킷이 도착하지 않았으므로 인터럽트로 전환을 한다. 그리고 적어도 한 개의 데이터의 값이 0 이 아니면 n 번의 폴링을 하는 동안 적어도 한번의 패킷이 도착한 것이므로 시간적 지역성을 이용하여 다시 패킷이 도착할 가능성이 존재하므로 그대로 폴링 상태를 유지한다.

2.2 인터럽트에서 폴링으로 전환

인터럽트 상태에서는 폴링을 하지 않으므로 history 에 저장하는 연산(Operation)을 수행할 수 없다. 그러나 이 기간 동안은 history 는 의미가 없으므로 history 를 고려하지 않아도 무방하다. 왜냐하면 기다리는 동안 패킷이 도착하지 않았으므로 인터럽트로 전환이 된 것이며, 또 인터럽트로 패킷의 도착을 기다리는 동안은 패킷이 도착하지 않았기 때문이다. 결과적으로 인터럽트 상태에서 history 는 항상 패킷이 도착하지 않은 상태를 나타내게 될 것이다. 그러므로 패킷이 도착하여 인터럽트가 호출되면 이전의 history 를 볼 필요 없이 폴링 상태로 전환(switching) 한다. 왜냐하면 패킷의 시간적인 지역성을 고려해 보면 앞으로 패킷이 도착할 가능성이 높기 때문에 폴링으로 전환하는 것이 효과적일 것이기 때문이다. 그리고 패킷이 도착하였음을 history 에 설정(set) 한다.

3. Implementation and Simulation

현재까지 구현되어진 어떤 단일 프로세서 OS 에서도 입출력 처리 루틴을 폴링으로 구현한 것이 없기 때문에 폴링의 오버헤드를 정확히 알 수가 없다. 그로 인하여 인터럽트와 폴링의 비교도 하기 어렵고, 또 새로운 인터럽트와 폴링 전환 알고리즘과의 비교도 할 수 없으므로, 본 논문에서는 실제로 최적의 폴링 코드를 i486 어셈블리 코드로 구현을 하고, 그 어셈블리 명령의 clock 수를 계산하여 폴링의 오버헤드로 사용함으로써 인터럽트와 폴링의 비교를 수행하고 새로운 인터럽트와 폴링 전환 알고리즘의 오버헤드도 어셈블리 코드를 사용함으로써 계산함으로써 인터럽트와 폴링과의 비교를 수행하였다.

본 논문에서 시뮬레이션을 위하여, 또 네트워크 인터페이스로 도착하는 패킷들의 패턴을 알아 보기 위해서는 실제로 시간에 따른 도착한 패킷의 수를 알아야 할 필요가 있다. 이를 위하여 실제 사용되는 컴퓨터 시스템에서는 Window NT 의 시스템 성능 모니터와 SNAPS(SNMP Network Analysis and Planning Supporting System)을 사용하여 그 값을 측정하였다.

3.1. 각 방법의 성능 평가와 시뮬레이션을 위한 가정

다음은 본 논문에서 인터럽트와 폴링의 성능을 비교하고, 제안된 알고리즘의 성능 평가를 위해, 그리고 시뮬레이션을 위해 세운 가정들이다.

3.1.1. 일반적인 가정

- No cache miss
- i80486 에서 unix 나 Windows NT 가 OS 로 사용될 수 있다.
- Sampling 한 시간 사이에서는 패킷이 random 하게 나타난다.

3.1.2. 인터럽트에 필요한 가정

- 인터럽트의 발생시 system 의 상태를 2 가지로 나눈다.
- 한번의 인터럽트에서 2 개 이상의 패킷을 처리하지 않는다.
- 인터럽트 처리를 할 때 task switching 을 하지 않는다.

3.1.3. 폴링에 필요한 가정

- 우리가 구현한 폴링 *mechanism* 이 맞다.
- 폴링 기법에서는 한번의 폴링으로 네트워크 인터페이스의 버퍼 안에 있는 모든 패킷을 한꺼번에 처리한다.

3.1.4. 인터럽트와 폴링 전환 알고리즘에 필요한 가정

- 구현된 폴링 코드는 운영체제의 일부분으로 실행이 되어진다.
- 인터럽트와 폴링의 전환은 네트워크 인터페이스의 특정 register 를 간단히 프로그램 함으로써 가능하다.
- OS 는 하드웨어를 이용하여 간단히 현재 상태가 폴링인지 인터럽트인지 알 수 있다.

3.2. 인터럽트와 폴링, 인터럽트와 폴링 전환에 대한 오버헤드

본 절에서는 인터럽트와 폴링을 비교하기 위한 각각의 오버헤드와, 본 논문에서 제안된 새로운 인터럽트 폴링 전환 알고리즘의 오버헤드, 그리고 실제 세부적인 구현에 대해 정리한다. 본 논문에서 주어진 오버헤드와 clock 수는 i80486 시스템을 기준[1]으로 하고 있다.

3.2.1 인터럽트

인터럽트는 일반적으로 입출력 부분(Body)을 수행하기 전에 일어나는 오버헤드와 입출력 부분을 수행한 후에 원래의 위치로 되돌아갈 때 발생하는 오버헤드가 존재한다.

Overhead (Interrupt) :
Protected mode, Same Privilege : 44 clock
Protected mode, Different Privilege : 71 clock

Overhead (Interrupt return) :
Protected mode, Same Privilege : 20 clock
Protected mode, Different Privilege : 36 clock

Total Overhead of Same Privilege 64
Total Overhead of Different Privilege 107

3.2.2 폴링
폴링은 입출력 사건이 일어났는지 검사하는 부분에 오버헤드가 존재한다. 본 논문에서는 폴링 기반의 입출력 처리 루틴을 다음과 같은 어셈블리 코드를 사용하여 구현함으로써 오버헤드를 계산하였다.

폴링의 어셈블리 코드와 오버헤드

```
IN acc, port                    8 clock
      - 네트워크 인터페이스에서 flag를 가지고 온다.
CMP acc, 0                      1 clock
      - flag로 패킷이 도착했는지 check 한다.
JE End_of_Body -- Taken         3 clock    : 도착하지 안았으면
               -- Not Taken     1 clock    : 도착 했으면

폴링 Overhead : Polling Taken       12 clock
                Polling Not Taken   10 clock
```

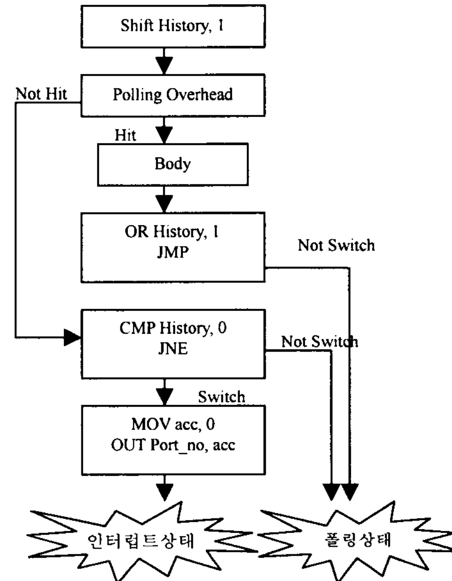
3.2.3 인터럽트 폴링 전환 알고리즘(IPSwitch algorithm)

3.2.3.1. Interrupt to polling switch 오버헤드
제안된 IPSwitch 알고리즘에서는 인터럽트가 수행되면 무조건 폴링 상태로 전환을 하며, 또 패킷이 도착한 history를 남기기 위하여 아래의 어셈블리 코드들이 body 부분을 수행 하기 전에 추가로 수행이 되어야 한다.

```
MOV acc, POLLING      1 clock
OUT PORT_NO, acc      10 clock
OR HISTORY, 1         1 clock

Total                 12 clock
```

3.2.3.2. Polling to interrupt switch 오버헤드
폴링 상태에서 기존의 일반적인 폴링을 수행하며 추가로 패킷이 도착 하였는지 여부를 history에 저장을 하고, policy switch 여부를 판단하기 위하여 history를 본다. 인터럽트의 경우 policy의 전환이 간단하지만, 폴링의 경우 history를 기록하고 확인을 해야 하는 오버헤드가 존재한다. 폴링에서 인터럽트로 전환되는 과정의 개략도는 <그림 2>에서 제시되었다.
매 폴링 시간마다 패킷의 도착 유무와 history window를 고려하여 policy를 결정하게 되는데, 3가지 경우가 있다. 그 각각의 경우마다 수행되는 어셈블리 명령은 서로 다르며, 그 각각의 경우의 오버헤드는 다음과 같다.



< 그림 2 > 폴링과 인터럽트 전환도.

Polling Hit
폴링을 했을 경우 도착한 패킷이 있어서 처리를 하는 경우, 패킷이 도착하였음을 history에 기록을 하기 때문에 인터럽트로 전환하는 것을 고려할 필요가 없다.

```
Shift History, 1              4 clock : Move history window
IN acc, port                  8 clock
CMP acc, 0                    1 clock
JE End_of_Body  -- Not Taken           1 clock
Body
OR Register, 0x00000001       1 clock : write to history
JMP Next                      3 clock

Total Polling Overhead                 18 clock
```

Polling not Hit but Continue to Poll
폴링을 수행했으나 도착한 패킷이 없는 경우, 그러나 history를 고려한 결과 그대로 폴링을 하기로 결정이 난 경우이다.

```
Shift History, 1              4 clock : Move history window
IN acc, port                  8 clock
CMP acc, 0                    1 clock
JE End_of_Body  -- Taken      3 clock
CMP History, 0                2 clock
```

```
JNE Next          -- Taken    3 clock

Total Polling Overhead                21 clock
```

Polling not Hit and Switch to Interrupt

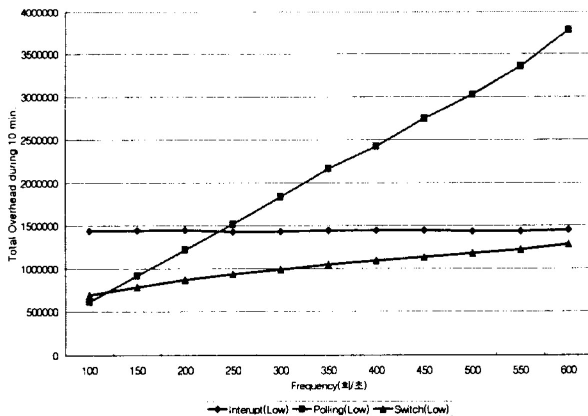
폴링을 수행했으나 도착한 패킷이 없고, history를 고려한 결과 이전에 패킷이 오랫동안 도착 안한 경우로 인터럽트로 전환을 한다.

```
Shift History, 1              4 clock : Move history window
IN acc, port                  8 clock
CMP acc, 0                    1 clock
JE End_of_Body -- Taken       3 clock
CMP History, 0                2 clock
JNE            -- Not Taken   1 clock
MOV acc, 0                    1 clock
OUT Port_No, acc              10 clock

Total Polling Overhead        30 clock
```

4. Simulation Results

본 절에서는 네트워크 인터페이스와 같이 그 발생 빈도가 빈번한 입출력 장치에 있어서의 인터럽트와 폴링의 성능을 비교하고, 본 논문에서 새로 IPSwitch를 사용할 경우 인터럽트와 폴링 방식에 비해 폴링 주기의 변화에 따라 얼마나 효율적인지 비교를 한다. 또한 History window의 크기 변화에 따른 IPSwitch의 성능 변화에 대해 알아보고자 한다.



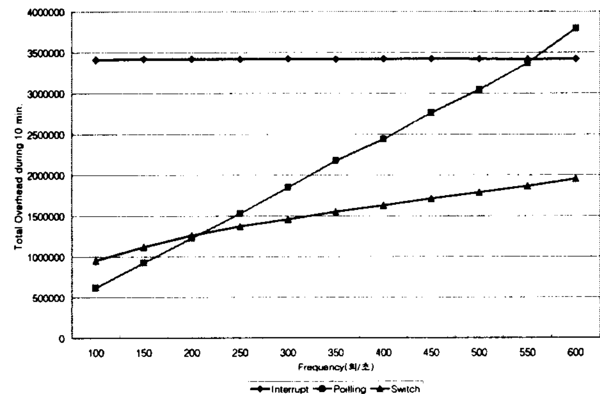
<그림 3> 폴링 주기에 따른 비교(low load, windows size=4)

4.1 인터럽트와 폴링, IPSwitch 방식의 비교

네트워크 인터페이스에 부하가 적게 걸린 상황일 때 인터럽트와 폴링, IPSwitch 알고리즘을 비교하고 있다. <그림 3>에서 알 수 있듯이 인터럽트는 오버헤드가 일정하나 폴링의 경우 폴링 주기가 240 이하일 경우는 인터럽트 보다 훨씬 좋은 결과를 내고 있으며 240 이상에서는 오버헤드가 인터럽트보다 훨씬 많음을 알 수 있다. 일반적으로 시스템에서의 폴링 주기는 시스템의 용도에 따라 다르지만 일반적으로 호스트와 같은 경우 초당 250개 이상의 패킷이 들어옴으로 최소한 300번 이상의 폴링은 수행해야 한다. 따라서 이와 같은 경우 인터럽트가 폴링 보다 우수하다.

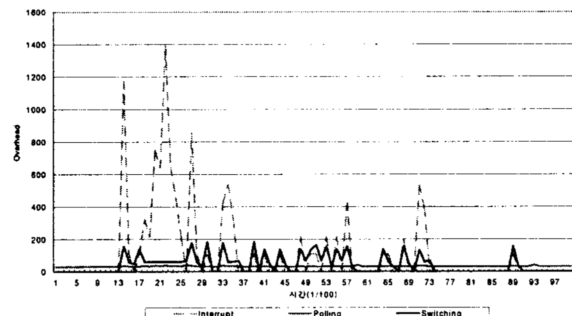
본 논문에서 새로 제안된 인터럽트와 폴링을 전환하는 알고리즘의 경우 폴링 주기가 600 이상이 되어도 인터럽트나 폴링에 의한 오버헤드보다는 훨씬 낮은 결과를 가지고 있음을 확인할 수 있다. 즉 이 그래프를 통해서만 보더라도 제안된 알고리즘의 네트워크 인터페이스에서 효율적으로 동작하며 우수한 성능을 내고 있음을 알 수 있다.

네트워크 인터페이스에 부하가 많이 걸린 상황에서 인터럽트와 폴링, IPSwitch 알고리즘을 비교하고 있다. <그림 4>에서 볼 수 있듯이 인터럽트는 입출력의 빈도가 많을 경우 수행해야 하는 코드의 크기가 크므로 오버헤드가 매우 높은 것을 알 수 있다. 이에 반하여 폴링 방식의 입출력은 폴링 주기가 증가함에 따라 계속적으로 오버헤드가 증가하나 폴링 주기가 550 이하일 경우 인터럽트보다 좋은 결과를 나타내고 있다.



<그림 4> 폴링 주기에 따른 비교 (high load, window size=4)

본 논문에서 제안된 IPSwitch 알고리즘의 경우 폴링 주기가 증가함에 따라 그 오버헤드의 증가 정도가 급격하지 않으며 인터럽트나 폴링에 비해 훨씬 우수한 성능을 나타냄을 알 수 있다. 즉 본 논문에서 제안된 알고리즘이 효율적임을 알 수 잇다. 여기서 폴링 주기가 200 이하일 경우는 폴링이 우수한 결과를 나타내고 있다. 이것은 폴링의 횟수가 너무 적어 그 오버헤드가 적기 때문에 당연한 결과이다.

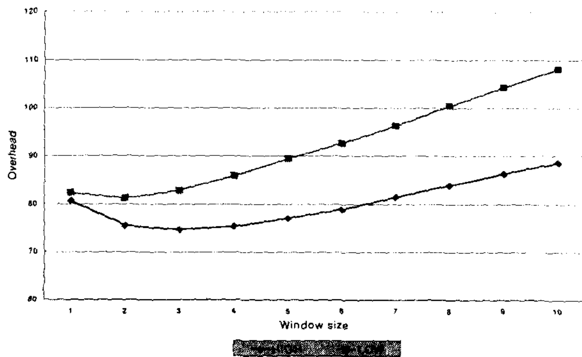


<그림 5> 시간에 따른 정책의 비교(주기=300, window size=2)

<그림 5>에서는 시간의 변화에 따른 인터럽트, 폴링, IPSwitch 의 오버헤드의 변화를 나타낸 것이다. 그림에서 알 수 있듯이 일반적으로 폴링은 그 오버헤드가 일정한데 비해서 인터럽트는 데이터가 들어왔을 경우 급격히 오버헤드가 증가하는 것을 알 수 있다. 이에 반하여 제안된 알고리즘은 데이터가 없을 경우 오버헤드가 전혀 존재하지 않다가 데이터가 들어왔을 때(가령, 13 초나 17 초) 약간의 오버헤드가 증가함을 알 수 있다. 또한 데이터의 입력이 없을 경우 다시 어느 정도의 시간이 지난 후 오버헤드가 없음(가령, 59 초나 74 초)을 알 수 있다.

4.2 IPSwitch 방식의 윈도우 크기에 따른 성능 비교

인터럽트와 폴링 전환 알고리즘에서는 각 기법의 전환을 위한 기준으로써 현재의 시점을 중심으로 이전의 폴링의 history 를 이용한다. 따라서 윈도우의 크기에 따라 제안된 알고리즘의 성능이 차이가 난다. 이 절에서는 윈도우의 크기가 변함에 따라 제안된 알고리즘의 성능의 변화를 알아보고자 한다.



<그림 6> window size 에 따른 IPSwitch 의 성능 비교 (주기=1000)

<그림 6>을 보면 알 수 있듯이 window size 와 성능이 비례하는 것이 아니라 최적의 값이 존재한다는 것이다. 이 그림을 통해서 윈도우의 크기가 커질수록 오히려 오버헤드가 점점 증가함을 알 수 있다. 이것은 윈도우의 크기가 커짐으로 인해서 알고리즘은 인터럽트 보다는 오히려 폴링 상태에 머물게 되고 이로 인하여 주기적으로 계속 폴링을 수행하기 때문이다. 그리고 너무 적으면 폴링 상태와 같게 된다.

5. 결론

네트워크의 속도가 빨라지고, 처리량이 증가함에 따라 네트워크를 통한 데이터의 발생 빈도가 높아지고 있다. 따라서 발생 빈도가 빈번하지 않았을 때의 입출력 데이터의 처리 기법인 인터럽트를 사용하는 것은 시스템의 성능을 향상 시키지 못한다. 따라서 본 논문에서는 발생 빈도가 빈번한 네크웍 인터페이스와 발생 빈도가 빈번하지 않은 키보드 데이터를 이용하여 인터럽트와 폴링의 효율성을 비교함으로써 입출력 데이터의 처리 기법의 변화가 필요함을 알 수 있었다. 또 인터럽트와 폴링 이외의 인터럽트와 폴링 전환 방식(IPSwitch)을 제시하고, 인터럽트와 폴링과의 비교를 통해서 제시된 알고리즘의 효율성을 증명하였다.

본 논문을 통해서 인터럽트가 폴링에 발생빈도가 빈번하지 않은 입출력 장치에 대해서는 우수하지만 발생 빈도가 빈번하며, 폴링 주기가 크지 않다면 오히려 폴링에 의한 방법이 더 우수함을 알 수 있었다. 또한 본 논문에서 제안된 인터럽트와 폴링 전환 방식(IPSwitch)과 같이 입출력 패턴의 시간적 지역성(Temporal locality)를 이용함으로써 시스템의 성능을 향상시킬 수 있음을 알 수 있었다.

7. 참고문헌

[1]. Pilgrim, Aubrey, Build your own 80486 PC save a bundle, Windcrest, 1991

[2]. Brey, Barry B, Intel microprocessor : 8086/8088, 80186, Merrill, 1991

[3]. 김병천, 80286 및 80386 어셈블리어, 자유아카데미, 1989

[4]. 오길록, M68000 어셈블리어, 홍릉과학사, 1991

[5]. Window NT 시스템 관리 툴, 정보시대, 1997

[6]. Hyde, Randall, Barkakati, Nabajyoti, 마이크로소프트 매크로 어셈블러 바이블, 인포북, 1993

[7]. http://developer.intel.com/design/

[8]. http://songgang.skku.ac.kr/snaps/