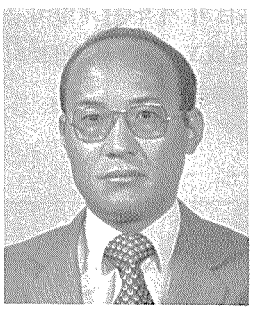
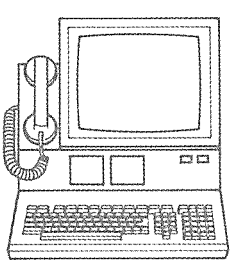


吳 吉 祿
韓國電子通信研究所
컴퓨터연구부장/工博

Multiprocessor Operating System



註 : 本稿는 ETRI 컴퓨터 개발부의 위탁 과제로 한양대학교 박용진 박사팀에서 수행한 다중운영체제에 대해 연구한 내용을 간추린 것이다.

Ⅰ. 序 論

최근의 급속한 반도체 기술의 발달은 컴퓨터 산업에 큰 변화를 가져왔다. 따라서 현대의 computer architecture분야의 최대 관심은 고속의 처리능력과 신뢰성이 높은 system의 개발에 있다. 이러한 요구를 만족시켜 주기 위해 여러개의 프로세서를 사용하여 처리능력을 크게 향상시키고 이들중 일부의 프로세서가 고장이 나더라도 나머지 프로세서들로써 정상적인 동작이 가능한 multiprocessor system에 많은 연구가 진행되어 오고 있다. 그러나 프로세서의 수에 비례해서 컴퓨터의 성능이 증가하지 않고 어느 수준에 이르러서는 오히려 성능이 감소되는 현상이 발생되었다. 이러한 현상은 프로세서간의 communication, scheduling 등 여러가지 요소에 의하여 발생되는데 이것은 컴퓨터의 성능을 향상시키려는 노력에 많은 어려움을 주고 있다.

여기서는 multiprocessor system의 O.S가 uniprocessor system의 O.S와 다른 점이 무엇이며 multiprocessor O.S의 설계와 system의 성능에 많은 영향을 주는 문제가 어떤 것이며 이들이 현재 어떻게 다루어지고 있는지 설명하겠다.

Ⅱ. Multiprocessor O.S와 Uniprocessor O.S와의 차이점

멀티프로그래밍 기능을 가진 uniprocessor system O.S에서는 다음의 기능들이 요구된다.

- 자원할당과 운영방식
- 기억장치와 데이타에 대한 보호기능
- 시스템 deadlock 방지기능

Multiprocessor system의 O.S는 위의 기능들

외에

- 프로세서간의 load-balancing 기능(scheduling)
- Parallelism detection기능
- 병행처리가 가능한 프로세스들간의 synchronization 기능

등이 더 요구된다.[1]

위의 3가지 외에 더 많은 기능들이 multiprocessor system O.S를 구성하는데 필요하지만 이들이 system performance에 미치는 영향이 매우 크므로 이 3가지에 대해서 간단히 설명하겠다. 그리고 현재까지 개발되었거나 개발중인 시스템과 이들을 개발하는 연구기관들이 표1에 나타나 있다.

표 1.

시스템명칭	연 구 기 관
NYU	New York Univ.
TRAC	Texas Univ.
MPP	NASA/Goddard Space Flight Center
Navier-Stock Computer	Princeton Univ.
CEDAR	Univ. of Illinois
Cm*	Carnegie-Mellon Univ.
CRAY	CRAY Research Inc.
D-825	Burroughs
7600	CDC
1100/80	UNIVAC
308X	IBM
HEP	Denelcor
System 6400	ELXSI
ECLIPSE MV/20000	Data General

Ⅲ. Parallelism

Performance를 높이기 위해서는 프로그램에서 병행처리를 할 수 있는 부분을 가능한 한 많이 찾아내어 처리하면 좋을 것이다. 그러나 병행처리가 가능한 부분을 schedule하고 synchronization 시키는데 많은 시간이 소모되므로 지나치게 프로그램을 분할시키면 오히려 performance가 감소된다. [2]

그림 1의 (a)에서 control-flow graph에는 7개의 노드가 있다. 이들을 schedule하고 synchronization시키는데 필요한 시간이 크다면 그림 1의 (a)의 노드 2와 3, 4와 5, 6과 7을 각각 하나의 노드 W, Y, Z으로 묶어 (b)와 같이 만들어 scheduling과 synchronization에 필요한 시간을 줄이는 것이 노드 5와 6을 병행처리하는 것보다 더 효과적일 수 있다.

따라서 병행처리를 해서 얻는 잇점과 scheduling과 synchronization에 소요되는 시간을 고려해서 프로그램을 적절히 분할하는 것이 매우 중요하다. 이 관계는 그림 2에 나타나 있다.

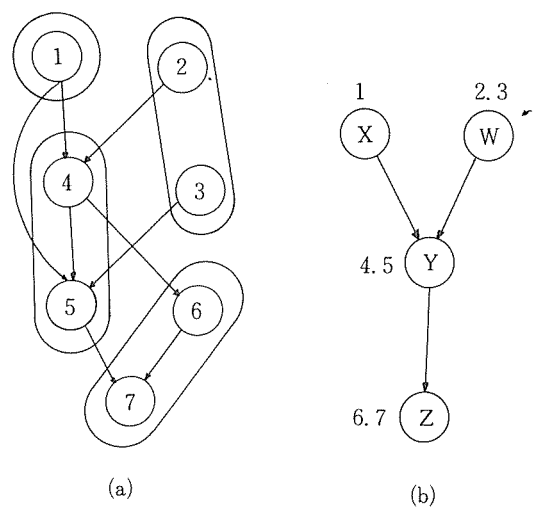


그림 1.

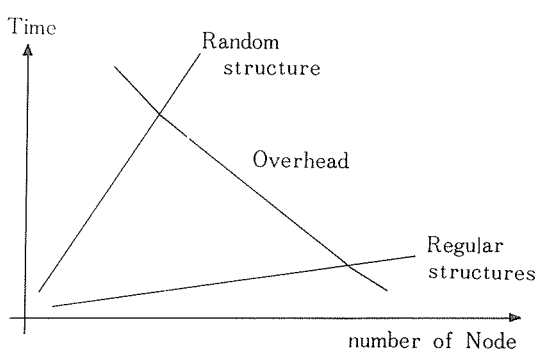


그림 2.

Ⅳ. Scheduling

Scheduling을 설명하기 전에 프로그램은 여러개의 테스크(task)로 구성되며 테스크는 여러개의 프로세스로 이루어졌다고 정의한다.

Scheduling에는 테스크 또는 프로세스가 프로그래머나 compile하는 과정에서 compiler에 의해 할당되는 정적인 방법과 프로그램이 실행될 때 machine에 의해 할당되는 동적인 방법이 있다. 첫번째 방법은 schedule 시간이 적은 반면 프로세서를 효과적으로 사용할 수 없으며, 두번째 방법은 schedule 시간이 조금 많더라도 프로세서의 load-balance를 이루며 이들을 효과적으로 이용할 수 있다는 장점이 있다. 현재 연구중인 시스템들이나 이미 개발된 시스템에서는 이 두가지 방법을 scheduling-level[1]에 따라 달리 적용한다. 이런 예가 표 2에 나타나 있다.

표 2. HEP CEDAR에서의 Schedule방식

Machine	Scheduling
HEP	Task; Dynamic Process; Dynamic self-scheduling
CEDAR	Task; Dynamic Procss; Static

HEP machine에서는 예측할 수 없는 delay로 인해 전체적인 실행시간이 길어지는 것을 방지하기 위해 self-scheduling 방식을 사용한다. 예를 들어 그림 3에서 HEP는 100개의 iteration을 10개의 프로세스가 나누어 처리하지만 각 프로세스가 처리하는 iteration의 수는 프로세스의 상황에 따라 달라진다. 프로세스들은 하나의 iteration을 실행한 후 $ NOI 값에 따라 그 다음 실행할 iteration을 가져오게 된다. 이렇게 함으로써 static하게 iteration들을 분할 했을 경우에 예기치 않았던 delay에 의해 실행시간이 더 길어지는 것을 방지할 수 있다. 그러나 self-scheduling방식은 iteration수가 프로세스의 수보다 매우 클 때 효과적이다.

CEDAR에서는 각 프로세스가 실행할 iteration의 수는 프로그램상에서 미리 결정된다. 그림 4. Cray XMP-2 컴퓨터에서도 이와 같은 방식을 사용하고 있다.[3,4]

Dynamic Scheduling 방식으로는 multiprogramming기능을 가진 시스템에서 사용되는 first-come-first-serve, least-service-time-first, random-choice, round-robin, preemptive 방법들이 multiprocessor system에서도 사용될 수 있다.

(a)
```
DIMENSION A(100), B(100), C(100), D(100)
N=100
DO 10 I=1, N
    A(I)=B(I)+C(I)
    D(I)=A(I)**2
10 CONTINUE
```

(b)
```
COMMAND A(100),B(100),C(100),D(100),N
N=100
NP=10
PURGE $ NOI, $ TOTAL
$ TOTAL=0
DO 10 I=1, (NP-1)
CREAT DOALL($ NOI, $ TOTAL)
10 CONTINUE
CALL DOALL($ NOI, $ TOTAL)
20 IF (VALUE($ TOTAL). LT. N)GOTO 20
SUBROUTINE DOALL ($ NOI, $ TOTAL)
MON A(100), B(100), C(100), D
(100), N
100 I=$ NOI
$ NOI=I+1
IF(I. GT. N) GOTO 200
A(I)=B(I)+C(I)
D(I)=A(I)**2
$ TOTAL=$ TOTAL+1
GOTO 100
200 RETURN
END
```

(a) Serial source code
(b) Parallel self-scheduling code

그림 3. Self-scheduling in HEP

Multiprogramming과 multiprocessing이 동시에 이루어지는 경우 multiprocessing을 하는 테스크 또는 프로세스들이 다른 프로그램들과 프로세서를 할당받기 위해 경쟁을 할 수 있다. 이

때는 multiprocessing을 하는 프로세스나 테스크에 priority를 줌으로써 multiprocessing의 잇점을 살릴 수 있다.[3]

```
DO 102 I= 1, 10000
       F(I)=ABS(F(I))
102 IF  (G(I). LT. 0) F(I)= -F(I)
```

(a) Serial code

```
C : N=10
    local private integer tasknum(N), T, J
    global shared integer I
    I= 0
    do j=i, N
       T=creat-task( )
       tasknum(J)=T
       call start-task(T, CC)
    end do
    do J= 1, N
       T=tasknum(J)
       call wait-task(T)
       call delete-task(T)
    end do
CC : local private integer J, K
    dowhile(sync(1<100, J=++1))
       J=J*100-99
       do K=J, J+99
            F(I)=abs(F(I))
            if(G(I). it. 0)F(I)=-F(I)
    end do
    end while
    call end-task( )
```

(b) Parallel code

그림 4. Scheduling in Cedar

V. Synchronization

Multiprocessor system에서의 synchronization은 multiprocessing을 하는 테스크 또는 프로세스간의 synchronization을 말한다. Synchronization은 여러 프로세스 또는 테스크가 shared variable을 가지고 있거나 어느 한 프로세서에서 실행되는 프로세서가 처리한 결과를 필요로 할 때 발생한다.

여기서는 아래의 3가지 방법을 소개한다.

- Test & Set
- Bit map synchronization method
- Message passing method

1. Test & Set

IBM 360/370에서 사용하는 방법으로 shared variable에 대한 mutual access를 가능하게 한다. 그림 5에 test & set의 기능이 나타나 있다. 그림 5에서 shared variable을 access하려는 프로세스는 a=0일 때 access를 할 수 있다. 따라서 a=0일 때 프로세스는 a=1로 하여 자기가 access하는 동안 다른 프로세스들이 이 shared variable을 access할 수 없게 한 다음 access한다. 그리고 필요한 목적을 달성한 다음에는 a값을 다시 a=0로 하여 다른 프로세스가 access할 수 있게 한 다음 빠져 나온다.

2. Bit-map Method

Cedar에서 사용하는 방법이다.

```
PROCESS Mutual(a);
    WHILE Test $ Set (a) 〈〉 0 DO
          Nothing;
    Access The Shared Variable;
    a= 0 ;
    END
```

그림 5. Mutual execlusion implementented by test & set

READ/WRITE, Address, Mask가 synchronization instruction으로써 implement 되어 있다. Mask는 F/E bit를 test하고 set시키기 위해 사용된다.

READ/WRITE에 대한 sequence는 다음과 같이 정의된다.

READ

$$\bigwedge_{F=1}^{n} (\text{Maski OR Key i});$$

Register←Memory(Address);

Key←Mask AND Key

WRITE

$$\neg\left(\bigvee_{i=1}^{m} (\text{Mask i AND Key i})\right);$$

Memory(Address)←Register

Key←Mask OR Key

예를 들어 R_1, R_2, W_3, R_4, W_5의 순으로 같은 Memory location을 access할 때 이에 대한

Bit-map table은 표 3에 나타나 있다. 표의 SYNC value와 R_1의 Mask value를 가지고 위에서 보여준 식을 적용시키면 Table의 값을 얻을 수 있다.

표 3. Bit-map synchronization example

	SYNC OR MASK VALUE					
	W_5	R_4	W_{32}	W_{31}	R_2	R_1
Initial SYNC value	1	0	1	1	1	1
MASK of R_1	1	1	1	0	1	0
SYNC after R_1	1	0	1	0	1	0
MASK of R_2	1	1	0	1	0	1
SYNC after R_2	1	0	0	0	0	0
MASK of W_3	0	1	1	1	0	0
SYNC after W_3	1	1	1	1	0	0
MASK of R_4	0	0	1	1	1	1
SYNC after R_4	0	0	1	1	0	0
MASK of W_5	1	0	0	0	1	1
SYNC after W_5	1	0	1	1	1	1

3. Message Passing 방식

Read, Write 하는 대신 프로세스가 message를 보내고 받고 함으로써 synchronization이 이루어질 수 있다. Sender 프로세스가 message를 보낸 후에 수신측 프로세스가 message를 받을 수 있기 때문에 두 프로세스간에 실행순서가 정해진다. 필요한 message를 받을 때까지 실행을 일시 중지하고 wait 상태에 있다가 message가 도착한 다음에 다시 실행을 할 수 있도록 하기 위해서는 event mechanism이 이용된다. Star OS[6]에서는 프로세스간에 message를 교환하기 위해 mail box라는 mechanism을 사용한다.

위에서 설명한 방법외에 NYU Ultracomputer에서 사용하는 F & A(v, e), [5] C. mmp의 lock instruction 등 다른 방법들이 있다.

Ⅵ. 結 論

Multiprogramming system의 O. S에서 필요로 하는 기능들 외에 multiprocessor O. S를 구성하기 위해 어떤 기능들이 더 필요한가를 살펴보고 이들이 각 시스템에서 어떻게 다루어지는지 간단하게 소개했다.

물론 여기서 소개된 것들이 전부는 아니지만 이들이 multiprocessor system의 O. S에서 차지하는 비중은 매우 크다. 따라서 이부분에 많은 연구가 진행중에 있다.

參 考 文 獻

[1] H. M. DEITEL, "An Introduction to Operating System," 1983, Addison Wesley.

[2] Daniel D. Gajski and Jih-Kwon Peir, "Essential Issue Multiprocessor Systems," IEEE Computer, vol. 18, no. 6, June, 1985, pp. 9-27.

[3] John L. Larson, "Multitasking on the Cray X-MP-2 Multiprocessor" IEEE Computer, vol. 17, no. 7, July, 1984, pp. 62-69.

[4] Perry Emar, "Xylem; An Operating System for the Cedar Multiprocessor," IEEE Software, July, 1985, pp. 30-37.

[5] A. Gottlieb et al, "The NYU Ultracomputer-Desiging and MIMD Shared Memory Parallel Computer," IEEE Trans. Computer, vol. C-32, no. 2, Feb, 1983, pp. 175-189.