트랜스퓨터를 기반으로 하는 병렬컴퓨터 TiME을 위한 마이크로커널의 설계와 구현

Design and implementaion of microkernel for transputer - based parallel computer TiME

저자
(Authors)

오규봉, 최태영, 박찬익, 박찬모
Kyu-Bong Oh, Tae-Young Choe, Chan-Ik Park, Chan Mo Park

출처
(Source)

한국정보과학회 학술발표논문집 23(1A), 1996.4, 843-846 (4 pages)

발행처
(Publisher)

한국정보과학회
KOREA INFORMATION SCIENCE SOCIETY

URL

http://www.dbpia.co.kr/Article/NODE00624661

APA Style

오규봉, 최태영, 박찬익, 박찬모 (1996). 트랜스퓨터를 기반으로 하는 병렬컴퓨터 TiME을 위한 마이크로커널의 설계와 구현. 한국정보과학회 학술발표논문집, 23(1A), 843-846.

트랜스퓨터를 기반으로 하는 병렬컴퓨터 TiME을 위한 마이크로커널의 설계와 구현

오 규봉○ 최태영 박 찬익 박 찬모
포항공과대학교 전자계산학과/정보통신 연구소 시스템 소프트웨어 연구실

Design and implementaion of microkernel for transputer-based parallel computer TiME

Kyu-Bong Oh○ Tae-Young Choe Chan-Ik Park Chan Mo Park
System Software Lab.
Dept. of Computer Science and Engineering/PIRL
Pohang University of Science and Technology
email:kyubong@yoobee.postech.ac.kr

요 약

운영체제에는 파일 서비스, 프로세스 관리, 메모리 관리, 네트웍 통신 등 여러가지 기능을 지원한다. 그러나, 병렬컴퓨터를 위한 운영체제에서는 특정한 노드는 파일 서비스만을 제공해주고, 특정한 노드는 계산만을 하기 때문에 기존의 운영체제가 적당하지 않다. 그러므로, 모든 노드에서 필요한 기능들만을 모아서 운영체제를 만들고자 한다. 이 기능을 가진 운영체제는 minimum kernel을 형성하게 되며 microkernel[1]의 형태로서 디자인하는 것이 바람직하다. 이 microkernel은 T800계열의 트랜스퓨터를 기반으로해서 만들어진 병렬컴퓨터 TiME[6]에서의 각 노드에서 운영체제로 사용된다.

T800계열의 트랜스퓨터는 2개의 우선순위를 하드웨어적으로 제공해주나, 실시간 응용을 위해서는 다중 우선순위 스케쥴링이 필요하다. 또한, 다중 우선순위 스케쥴링을 위해서는 기존 language run-time system library로 제공되었던 여러 기능들을 assembly language차원에서의 변환이 필요하다.

따라서, 본 논문에서는 트랜스퓨터상에서 실시간 응용을 위해 다중 우선순위 스케쥴링뿐만 아니라 트랜스퓨터 assembly language차원에서 기존 language run-time system을 대체하여 process, channel, *semaphore management*를 지원하는 트랜스퓨터용 *microkernel*을 소개하고자 한다.

1. 서론

운영체제에 있는 여러가지 기능들 중에서 병렬컴퓨터 에서의 모든 노드에서 필요로 하는 기능들만을 모아서 microkernel을 구성하고 특히 디스크를 제어하는 노드의 경우에 필수적인 파일 서비스는 user process에서 수행되게 된다. 본 논문에서 설계,구현된 microkernel을 운영체제로 사용할 병렬컴퓨터는 T800계열의 트랜스퓨터를 기반으로 한 것으로써 계산을 위한 32개의 노드, 카메라 입력 제어을 위한 1개의 노드, 모니터 출력 제어를 위한 1개의 노드, 4개의 디스크 각각을 제어하기 위한 4개의 디스크 노드로 구성된다.

주어진 시간내에 결과를 출력하는 것이 필수적인 환경에서 사용되는 실시간 시스템에서는 다중 우선순위 스케쥴링이 기본적으로 요구된다. 그래서, 병렬컴퓨터 TiME을 위한 microkernel에서 기본적으로 제공되어야 할 기능이 다중 우선순위 스케쥴러이다. 그러나, T800계열의 트랜스퓨터는 2개의 우선순위만을 제공해 주므로 다중 우선순위 스케쥴링을 위해서는 별도의 소프트웨어 스케쥴러가 필요하다. 본 논문에서 제시하는 스케쥴러는 기본적으로 하드웨어 스케쥴러를 이용하며 사건 제어 방식을 적용한다. 사건 제어 방식 스케쥴러는 외부에서 특정한 사건이 발생했을 경우에만 호출되는 방식으로써 사건에는 크게 channel 통신, process의 생성/소멸, semaphore의 대기/종료가 있다.

다중 우선순위 스케쥴링을 위해서 위의 사건을 발생시키는 부분들을 트랜스퓨터 assembly language를 이용하여 새로 작성하였다. 또한, 스케쥴러의 내부에서 사용되는 명령어들도 assembly code로 직접 작성한 함수들을 사용함으로써 side effect의 가능성을 제거하였다. 또한, ANSI C language run-time library의 기능을 충실히 지원함으로써 기존 프로그램에 수정없이 다중 우선순위를 적용할 수 있다.

본 논문의 2장에서는 microkernel을 운영체제로 사용하게 되는 병렬컴퓨터 TiME의 구조를, 3장에서는 microkernel의 구조를, 4장에서는 microkernel의 성능평가를 다루고 5장에서 결론을 맺는다.

2. TiME의 구조

TiME은 트랜스퓨터를 기반으로 한 병렬컴퓨터이다. <그림1> 에서 보는 바와 같이 병렬컴퓨터 TiME은 다음 4가지의 각기 다른 목적의 노드들로 구성되어 있다.

- 계산을 위한 32개의 노드(그림1의 가장 위쪽에 있는 32개의 노드)
- 카메라 입력을 처리하기 위한 노드(그림1에서 TTGF라고 명명된 노드)
- 모니터 출력을 처리하기 위한 노드(그림1에서 TTG3라고 명명된 노드)
- 4개의 디스크 각각을 처리하기 위한 4개의 노드(그림1에서 DISK라고 명명된 노드)

병렬컴퓨터 TiME는 호스트 컴퓨터로 사용하고 있는 펜티엄 PC와 연결되어 있다. 병렬컴퓨터로의 프로그램 loading은 펜티엄 PC

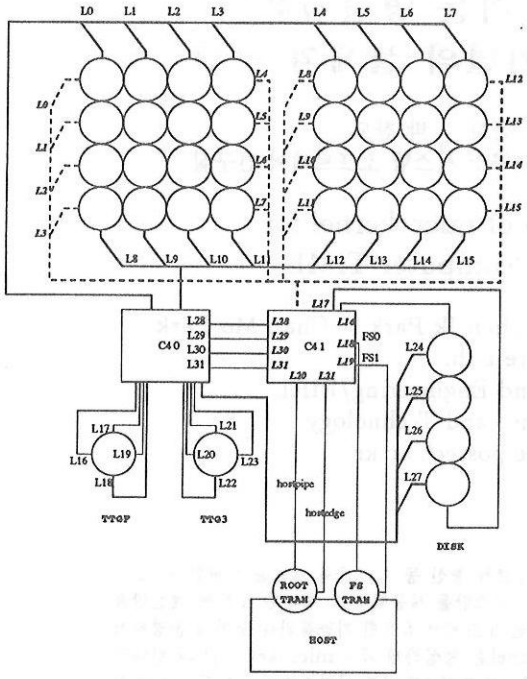


그림 1: 트랜스퓨터를 기반으로 한 병렬컴퓨터 TiME의 구조

에 설치된 두개의 보드중 그림1에서의 ROOT TRAM이 담당한다. 그리고, 호스트 컴퓨터에 설치된 또 다른 보드(그림1에서의 FS TRAM)는 TiME에서의 디스크와 직접 연결되어 있다.

3. Microkernel의 구조

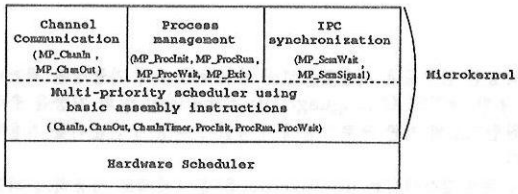


그림 2: Transputer-based microkernel의 구조

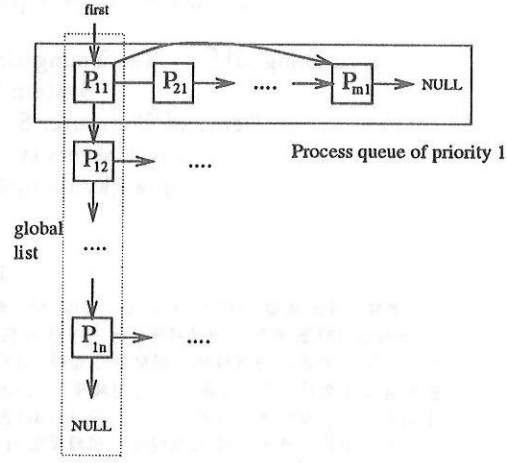
그림2은 하드웨어 스케쥴러와 다중 우선순위 스케쥴러, 그리고 다중 우선순위로 사건들을 발생시켜 특정한 기능을 수행하는 library의 관계를 나타낸다.

다중 우선순위 스케쥴러을 위해서 사건들을 발생시키는 부분들을 트랜스퓨터 assembly language를 이용하여 새로 작성하였다. 또한, 스케쥴러의 내부에서 사용되는 명령어들은 side effect의 제거를 위해서 트랜스퓨터 assembly language로 직접 함수들을 작성하여 사용하였다.

트랜스퓨터의 하드웨어 스케쥴러에서는 2-단계의 우선순위를 제공해 주며 2개의 프로세스 큐(HIQ, LOWQ)를 관리한다. High-level 우선순위를 가지는 프로세스는 HIQ에서 비선점형 방식으로 실행되며, low-level 우선순위를 가지는 프로세스는 LOWQ에서 round-robin 방식으로 실행된다. 다중 우선순위 스케쥴러는 하드웨어 스케쥴러 상에서 high-level 우선 순위를 가지며 microkernel의 각 기능(channel communication, process management, IPC synchronization)에서 사건을 발생시켰을 때 실행된다.

Microkernel의 기능을 실제로 사용하는 사용자 프로세스에게 주어진 다중 우선순위는 스케쥴러가 실제로 실행할 응용 프로세스를 선택할 때 고려되며 이 응용 프로세스는 하드웨어 스케쥴러상에서 low-level 우선순위를 가지고 실행된다.

(a) List of processes blocked by process wait, semaphore, internal channel, external channel



(e) List of processes ready for execution

그림 3: Microkernel에서 스케쥴러와 관련되어서 사용되는 자료구조

Microkernel에서 스케쥴러와 관련해서 사용되는 자료구조로는 그림3에서 보는 바와 같이 스케쥴러 내부에 유지되는 자료구조와 microkernel의 각 기능 내부에서 사용되는 자료구조가 있다.

스케쥴러 내부 자료구조는 실행 준비가 된 프로세스들의 리스트이다. 이 자료구조는 선점에 의해서 실행이 중단된 프로세스들, channel 통신이나 세마포어에서 block된 상태였다가 다시 실행 준비가 된 프로세스들을 보관하는 자료구조로써 같은 우선순위를 가지는 프로세스들이 하나의 하위 리스트를 구성하며 전체 리스트는 이들 하위 리스트들의 리스트로 구성된다. 전체 리스트는 우선순위의 순서로써 하위 리스트들을 배열하며, 하위 리스트에서의 프로세스들은 삽입된 순서에 따라서 나열되게 된다. 리스트에 새로 추가되는 프로세스는 우선 순위의 순서에 해당하는 위치의 하위 리스트까지 이동한 후에 그 리스트에 삽입이 되고, 리스트에서 삭제되는 프로세스는 항상 우선순위가 가장 높은 프로세스가 그 대상이 되므로 가장 위의 하위 리스트에 있는 프로세스들이 삭제된다.

다중 우선순위 스케쥴러와 관련해서 microkernel의 각 기능 내부에서 사용되는 자료구조는 MP_ProcWait 명령(일정시간 동안 프로세스를 idle시키는 명령어)에 의해서 대기중인 프로세스들의 리스트, 세마포어에 대기중인 프로세스들의 리스트, 내부 channel에 대기중인 프로세스들의 리스트, 외부 channel에 대기중인 프로세스들의 리스트가 있다. 외부 channel을 통한 통신은 내부 channel을 통한 통신과는 달리 다른 프로세서와 연결된 hardwire 링크를 통해서 통신을 하기 때문에 트랜스퓨터의 각 링크마다 하나의 background 프로세스를 실행시킨다. 이 프로세스들은 하드웨어 스케쥴러상에서 high-level 우선순위를 가지고 실행된다. 그러므로 이 프로세스들은 다중 우선순위 스케쥴러의 스케쥴 대상이 되지 않는다.

본 논문에서 제안한 microkernel내의 다중 우선순위 스케쥴러는 사건 제어방식으로써 그 동작원리는 다음과 같다. 사용자 프로세스가

microkerl의 기능을 호출하면 microkernel의 각 기능 내부에서 스케쥴러로 사건을 발생시킨다. 그 사건을 받은 스케쥴러는 사건의 종류에 따라서 조치를 취하게 된다. 발생시키는 사건들은 microkernel의 기능에 따라서 다음 세가지로 나뉜다.

- Channel communication에서 발생시키는 통신 종료, 통신 종료 대기.
- Process management에서 발생시키는 프로세서 생성, 프로세스 소멸, 프로세스 대기 그리고, process management가 발생시키지는 않지만 프로세스 대기와 관련해서 다중 우선 순위 스케쥴러 내부에서 발생시키는 프로세스 대기 종료.
- IPC communication에서 발생시키는 세마포어 대기, 세마포어 대기 종료.

```
1. If a process is to blocked
   a. If blocking reason == process wait blocked process queue
      (1) insert the process into process wait
   b. else if blocking reason == semaphore
      (1) insert the process into semaphore blocked process queue
   c. else if blocking reason == internal communication
      (1) insert the process into internal communication
   d. else if blocking reason == external communication
      (1) intert the process into external communication
   e. If the number of processes in LOWQ == 0
      (1) If there is a preempted process for processes of all processes
          of the software process queue of the highest priority
          . restore FPU computation and status registers
          . restore Interrupt Save Location
      (2) Inserts all processes of the same priority level into LOWQ
2. If a process is to be ready-to-run from blocked or a new process
   is to be created
   a. remove the process from the corresponding blocked process queue.
      /* the priority of the process = Pnew */
      /* the priority of currently ready-to-run process == Pcur */
   b. If Pnew < Pcur
      (1) insert the process into the software process queue of Pnew
   c. else if Pnew == Pcur
      (1) attach the process to the end of LOWQ
   d. else if Pnew > Pcur
      (1) insert all processes in LOWQ into the software process queue
          of Pcur and clear Pnew
      (2) save the content of Interrupt Save Location to the corresponding
          save area for interrupt process context information
      (3) clear Interrupt Save Location memory area
      (4) save FPU information such as FPU registers and status.
      (5) insert the process to the end of LOWQ.
3. If a process is to be terminated.
   a. do the same execution steps in 1-d.
```

그림 4: Microkernel에서의 스케쥴러 알고리즘

그림4은 다중 우선순위 스케쥴러 알고리즘의 개관을 보여준다.

스케쥴링 알고리즘은 microkernel의 각 기능에서 사건을 발생시면 사건의 종류을 조사하고 그 사건에 따라서 조치를 취한다. 각각의 프로세스는 ready-to-run 상태, block된 상태, terminated 상태중 하나의 상태를 가진다. 프로세스가 block된 상태일 경우에는 block된 이유에 따라서 block된 프로세스를 앞에서 설명한 그림3의 (a) 리스트들 중 하나에 저장한다. 이 프로세스가 block된 상태가 됨으로써 LOWQ에서 실행중인 프로세스가 없을 경우에는 ready-to-run 상태가 된 프로세스들을 저장하고 있는 리스트(그림3의 (b))에서 가장 우선순위가 높은 프로세스들을 LOWQ에 삽입시켜 실행시키게 되는데, 실행에 앞서 이 프로세스들 중 선점된 프로세스가 있다면 그 프로세스가 선점되기 전의 상태로 복구하기 위해서 선점될 당시의 register들(Floating Point Unit registers, Interrupt Save Location register)을 복구한다.

프로세스가 ready-to-run 상태가 되는 데는 두가지 가능성이 존재하는데 그 프로세스가 block된 상태에서 ready-to-run 상태로 바뀌었을 경우와 새로 만들어진 프로세스일 경우이다. Block된 상태에서 ready-to-run 상태가 된 프로세스의 경우에는 block된 프로세스들이 저장된 리스트(그림3의 (a))로부터 그 프로세스를 삭제한다. 그리고, ready-to-run 상태가 된 모든 프로세스에 대해서 그 프로세스의 우선순위와 현재 LOWQ에서 실행중인 프로세스들의 우선순위를 비교한다. 비교후 ready-to-run 상태가 된 프로세스의 우선순위가 작을 경우에는 ready-to-run 상태가 된 프로세스를 그림3의 (b)리스트에 삽입시키고, 같을 경우에는 LOWQ에서 그 프로세스를 실행시키고, 클 경우에는 현재 LOWQ에 있는 모든 프로세스들을 선점해서 LOWQ에 있는 프로세스들을 그림3의 (b)리스트에 저장한 후 ready-to-run 상태인 프로세스를 LOWQ에서 실행시킨다.

프로세스가 terminated 상태일 경우에는 이 프로세스가 종료됨으로써 LOWQ에서 실행중인 프로세스가 없으면 ready-to-run 상태가 된 프로세스들을 저장하고 있는 리스트(그림3의 (b))에서 가장 우선순위가 높은 프로세스들을 LOWQ에 삽입시켜 실행시키게 되는데, 실행에 앞서 이 프로세스들 중 선점된 프로세스가 있다면 그 프로세스가 선점되기 전의 상태로 복구하기 위해서 선점될 당시의 register들(Floating Point Unit registers, Interrupt Save Location register)을 복구한다.

4. 성능평가

(단위 : μs)

Microkernel의 기능	scheduling overhead
MP_ProcRun	43.17
MP_Exit	38.90
MP_ProcWait	103.30
MP_SemWait	48.06
MP_SemSignal	37.03
MP_Chan(In/Out)	100.73

표 1: 각 microkernel 기능에서의 scheduling overhead

성능 평가는 다중 우선순위 스케쥴러를 사용하는 microkernel의 기능들을 대상으로 해서 그 기능들의 실행시간 중에서 다중 우선순위 스케쥴러에서 소비되는 시간(scheduling overhead)을 측정하였다. 따라서, scheduling overhead에는 rescheduling 시간과 descheduling 시간 모두 포함되어 있다. Microkernel의 기능들중 지정된 우선순위의 새로운 프로세스를 하나 만드는 함수 MP_ProcRun, 수행이 끝난 프로세스를 종료시키는 함수 MP_Exit, 프로세스를 일정 시간 동안 idle 시키는 함수 MP_ProcWait, 내부 채널 통신 함수 MP_ChanIn과 MP_ChanOut, 세마포어 대기 함수 MP_SemWait, 그리고 세마포어 대기 종료 함수 MP_SemSignal을 성능평가의 대상으로 했다.

실험 환경은 T805-25MHz 트랜스퓨터 상에서 INMOS-C 컴파일러로 실행화일을 만들었다. 성능 측정을 위해 30개의 프로세스가 각각의 실행시간과 주기로 실행된다. 이때, rate monotonic scheduling [4] 알고리즘을 적용시켜 각 프로세스들은 5개의 우선순위 단계를 가지며 CPU 부하는 50%로 두었다. 다음 표4.는 성능평가의 대상이 된 각 함수들의 scheduling overhead를 측정한 것이다.

5. 결론

트랜스퓨터를 기반으로 한 병렬컴퓨터 TiME의 모든 노드에서 필요한 기능들만을 모아서 microkernel을 설계 및 구현하였다. Microkernel내의 다중 우선순위 스케줄러는 실시간 시스템에의 병렬컴퓨터 적용을 위해서 2-단계의 우선순위만을 제공해주던 트랜스퓨터상의 하드웨어 스케줄러를 확장하였다. 그리고, 다중 우선순위 스케줄링을 위해서 기존의 ANSI C language run-time library로 제공되었던 모든 기능들을 새로이 디자인하였다. 본 논문에서 구성된 microkernel을 바탕으로 해서 병렬컴퓨터 TiME에 있는 계산전용의 여러 노드간상에 상위 수준의 메시지 전달, 노드간 동기화 등을 제공하는 기능과 디스크 노드에서의 파일 시스템등을 구현하고 한다.

참고문헌

[1] Jean Bacon, Concurrent systems, *Addison-Wesley Company,*. 1992.

[2] INMOS C toolset user manual, *INMOS*, 1990.

[3] TRANSPUTER INSTRUCTION SET, *Prentice Hall*, 1988.

[4] C.L. Liu, and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *Journal of the ACM, 1(1)*, January 1973, pp.46-61

[5] 류경진 외 5인, "영상처리용 병렬 컴퓨터 TiME을 위한 커널 및 파일시스템," 한국정보과학회 가을학술발표논문집(제20권, 2호), 1993.10, pp.1073-1076.

[6] 박찬모 외 4인, "병렬 컴퓨터 구조에 관한 연구(영상처리용 고도 병렬 컴퓨터에 관한 연구)," 국방과학연구소 ADD-90-4-005, 1991.12.

[7] 최태영, 박찬익, "트랜스퓨터를 사용하는 실시간 시스템을 위한 다중 우선순위 스케줄러," 한국 트랜스퓨터 사용자 그룹 워크 숍, 1994.