Mach 커널의 재구성을 위한 확장된 스케줄 가능성 검사를 수행하는 실시간 스케줄러

류 진 열[†] · 김 광[††] · 허 신[†††]

요 약

본 논문에서는 실시간 스케줄링이 가능하도록 Mach 커널을 재구성하기 위해 확장된 스케줄 가능성 검사를 수행하는 실시간 스케줄러를 구현한다. 첫째, 실시간 운영체제의 요구사항에 따른 구성요소들을 제시하고 기존의 실시간 스케줄링 알고리즘에 대하여 분석한다. 둘째, Mach 커널 재구성을 위하여 Mach 커널의 실행 환경과 스케줄링 부분에 대한 분석을 통해 수정된 자료구조를 제시한다. 셋째, 기존 실시간 스케줄링 정책에서의 스케줄가능성 검사에서 확장된 스케줄가능성 검사 방법을 제시한다. 넷째, 확장된 스케줄가능성 검사를 거쳐 마감시간이 보장된 쓰레드들을 마감시간 우선 스케줄링 방식과 비율단조 스케줄링 방식으로 스케줄하는 스케줄러를 구현한다.

Real-Time Scheduler with Extended Schedulability Testing for Mach Kernel Reconfiguration

Jin-Yeol Ryu[†] · Kwang Kim[††] · Shin Heu[†††]

ABSTRACT

In this paper, we implement the real-time scheduler which performs extended schedulability testing, to reconfigure Mach kernel in which Real-Time scheduling is possible. For this purpose, first, we propose the configuration factors according to requirements of Real-Time operating systems and we analyze a Real-Time scheduling algorithm. Second, for the reconfiguration of Mach kernel, we propose the modified data structure through the analysis of Mach kernel environments and scheduling. Third, we suggest the extended scheduling method by analyzing conventional Real-Time scheduling policies. Fourth, we implement the scheduler which executes tasks according to the Earliest-Deadline-First scheduling and the Rate Monotonic scheduling.

1. 서 론

컴퓨터의 응용분야가 다양해지고 확장됨에 따라 각 응용분야에서 요구하는 사항들도 다양하게 변화되고 있다. 특히, 이러한 다양한 요구들을 충족시켜주기 위한 응용분야 중 외부에서 발생한 사건에 대하여 제한된 시간 내에 처리를 요구하는 시스템이 실시간 시스템이다[1].

기존의 UNIX 같은 비실시간 운영체제의 스케줄링 방식으로는 마감시간 내에 외부의 비동기적인 사건에 응답해야하는 실시간 시스템의 시간 제약조건을 만족시키지 못하기 때문에 실시간 응용에는 부적합하다. 따라서, 기존의 비실시간 운영체제의 실시간 기능 확

† 준 회 원 : LG정보통신 정보시스템연구소 미디어 S/W실
†† 정 회 원 : 삼일데이타시스템(주) 부설 기술연구소
††† 정 회 원 : 한양대학교 전자계산학과 교수
논문접수 : 1999년 7월 16일, 심사완료 : 2000년 1월 6일

장이 요구되며 이에 대한 많은 연구가 필요하다. 그러나, 기존의 UNIX에 실시간 기능을 추가한 실시간 운영체제의 경우 기존의 UNIX 운영체제와 개발 환경을 공유할 수 있는 유연성은 보장되지만, 대부분의 실시간 운영체제에서는 실시간 스케줄링을 위한 메커니즘만을 보완했을 뿐 그를 이용한 스케줄링 정책은 지원하지 못해서 완벽하게 마감시간을 보장하지 못하는 경우가 많았다.

본 논문에서는 기존의 비실시간 운영체제 중 Carnegie Mellon 대학에서 연구 발표되어 커널 코드의 완전한 공개로 운영체제를 연구하는 여러 분야에서 사용되고 있는 Mach 커널을 선택하여, 이 Mach 상에서 실시간 스케줄링이 가능하도록 커널을 재구성하고자 한다. 이를 위해서 실시간 운영체제의 요구사항에 따른 구성요소를 제시하고, 기존 실시간 스케줄링 알고리즘과 Mach 커널에 대한 분석을 통해서 실시간 스케줄링을 위해 가장 중요한 부분인 스케줄러를 설계하고 구현한다. 구현한 스케줄러는 모의 실험환경으로 X 윈도우 상에서 사용자 인터페이스를 통해 입력받은 태스크의 속성을 가지고 태스크들을 스케줄하는 모습을 보여주면서 이를 평가한다. 이렇게 구현된 스케줄러는 Mach 커널에 적용하여 Mach 상에서 실시간 스케줄링이 가능하게끔 할 수 있다.

본 논문의 2장에서는 실시간 운영체제에서 요구되는 기능과 실시간 스케줄링 알고리즘에 대해서 설명하고, 3장에서는 Mach 커널 재구성을 위해서 기존의 Mach 커널의 실행 환경과 스케줄링에 대한 분석을 통해 실시간 스케줄링이 가능하도록 수정된 자료구조를 제시한다. 4장에서는 실시간 스케줄링 정책을 설계하고 실시간 스케줄러를 구현한다. 5장에서는 구현된 실시간 스케줄러의 시뮬레이션 모델과 결과를 기술하고, 6장에서는 결론 및 향후 연구방향을 제시한다.

2. 실시간 운영체제

현재에는 실시간 계산의 폭넓은 다양성으로 인해 실시간 운영체제의 특성과 설계도 매우 광범위해졌으며, 더우기 복잡한 프로그램 규칙과 실시간 요구조건을 만족시키는 실시간 운영체제를 구현하는 일도 매우 어려운 일이 되었다. 이러한 실시간 운영체제에 대한 요구조건들에 대해서는 다음 절에서 설명하고자 한다.

2.1 실시간 운영체제의 요구조건

본 절에서는 현재의 실시간 운영체제의 특성[2, 3]을 포함해서 향후 실시간 운영체제가 갖추어야 하는 요구조건들을 제시한다.

2.1.1 실시간 태스크 스케줄링

기존의 UNIX 운영체제의 스케줄링 방식은 시분할 환경에 적합하도록 고안되었으며 스케줄러는 프로세스들의 우선순위를 주기적으로 재계산하여 각 프로세스들이 CPU를 공평하게 공유할 수 있도록 하였다. 그러나, 실시간 시스템에서는 이러한 비실시간 스케줄링 방법으로는 시간 제약성을 만족시킬 수 없다. 실시간 시스템의 스케줄링은 시간 제약성과 정확성을 보장하기 위해서 우선순위에 따르는 선점 스케줄링(preemptive scheduling)이 필요하다[4]. 실시간 태스크들은 마감시간 내에 수행되는 것이 보장되어야 하며, 수행 가능성을 예측할 수 있어야 한다. 이런 마감시간을 만족시키기 위해서는 우선순위에 의한 태스크 스케줄링을 하고 이에 따라서 수행시간을 정확히 예측할 수 있어야 한다.

2.1.2 실시간 태스크간 통신

실시간 시스템에서는 여러 태스크들이 병행적으로 수행되어야 한다. 여러 태스크들이 병행적으로 수행되기 위해서는 메시지 전달과 같은 태스크간의 통신이 필요하다. 실시간 시스템에서 이러한 태스크간의 통신은 각 메시지들이 마감시간을 갖고 이 마감시간 내에 전달되어야 한다는 제약사항을 가지고 있다. 기존의 UNIX 운영체제와 같은 범용 운영체제는 파이프(pipe)나 소켓(socket) 등의 프로세스간의 통신 방법이 제공된다. 그러나, 이런 방법들은 메시지에 우선순위나 시간 제약성 같은 것이 없고 선입선출(FIFO) 방식으로 처리되므로 메시지들이 마감시간 안에 전달되는 것을 보장하지 못한다. 실시간 운영체제에서는 우선순위, 마감시간들을 명시하여 마감시간 내에 전달이 보장되는 실시간 태스크간 통신 방법이 요구된다.

2.1.3 보증된 인터럽트 응답

실시간 운영체제는 사건(event) 발생을 빠르게 인식해야하며 발생한 사건을 예측할 수 있는 시간 안에 처리해야 한다. 또한, 하드웨어 인터럽트와 소프트웨어 인터럽트 모두 다 처리할 수 있어야 한다. 특히 높은 우선순위의 실시간 태스크가 "수행 대기" 상태로 되는

현상을 위하여 보장된 시간 안에 문맥교환이 이루어져야 한다.

2.1.4 고해상도의 타이머 및 동기화

실시간 시스템의 태스크들은 마감시간을 가지고 있으며, 또한 수행되는 동안 정확한 시간의 측정이 필요하다. 그러므로, 실시간 운영체제에서는 정확한 수행시간의 측정 등과 같은 시간에 대한 기능을 제공하기 위해서 고해상도의 타이머가 필요하다. 또, 실시간 시스템은 어떤 목적을 이루기 위하여 여러 태스크들이 병행적으로 수행된다. 이렇게 태스크가 병행적으로 수행되는 환경에서는 각 태스크들의 수행되는 순서를 제어하여야 하는 경우가 있다. 이런 제어를 하는 것이 동기화 방법이며 세마포어(semaphore) 같은 방법이 사용된다. 전통적으로 이런 동기화 방법들은 도착되는 순서대로 처리되는 선입선출(FIFO) 방식으로 관리되므로 실시간 시스템에서는 이런 기존의 처리 방식으로는 태스크의 마감시간을 만족시키기가 어렵다.

2.1.5 시스템 자원의 사용자 제어

실시간 시스템에서는 CPU, 메모리, I/O 자원들을 비롯한 시스템 자원을 사용자가 제어할 수 있어야 한다. 사용자가 사용자 태스크에 우선순위를 부여하는 우선순위 기반 스케줄링 기법으로 CPU의 제어가 가능하며 인터럽트가 발생할 때 빠른 문맥 교환을 위해 메모리에 있는 프로그램 전체나 일부를 사용자가 잠금(lock)으로서 메모리를 제어할 수 있다.

이러한 실시간 운영체제의 여러 요구조건 중, 본 논문에서는 Mach 커널에서 실시간 스케줄링이 가능하도록 하기 위해서 스케줄링 부분에 대한 중점적인 연구를 진행한다.

2.2 실시간 스케줄링 알고리즘

본 절에서는 기존의 실시간 스케줄링 알고리즘 중 가장 널리 알려져 연구되고 있는 네 가지 스케줄링 알고리즘에 대해서 기술한다[5].

2.2.1 비율 단조(Rate-Monotonic) 알고리즘

최적의 정적 알고리즘으로 알려졌으며, 주기를 기초로 태스크에 정적 우선순위를 할당하고 짧은 주기의 태스크에 높은 우선순위를 할당한다. 태스크가 도착할 때 우선순위가 할당되므로 우선순위는 재계산될 필요가 없다[6].

2.2.2 마감시간 단조형(Deadline-Monotonic) 알고리즘

정적 알고리즘이며 태스크는 상대적 마감시간에 따라서 우선순위가 할당된다. 즉, 상대적 마감시간이 짧을수록 높은 우선순위가 할당된다. 이것은 태스크들이 주기보다 짧거나 같은 마감시간을 갖는 경우에 최적인 알고리즘이다[7]. 특히, 마감시간이 주기와 같거나 주기에 비례해서 작은 경우에 생성되는 스케줄은 비율 단조 알고리즘과 같다. 그러므로, 이 경우에는 비율 단조 알고리즘과 마감시간 단조형이 모두 최적의 알고리즘이다.

2.2.3 마감시간 우선(Earliest-Deadline-First) 알고리즘

동적 알고리즘으로 선점 알고리즘들 중에서 최적인 것으로 알려져 있다. 태스크의 우선순위는 마감시간에 따라서 할당된다. 즉, 마감시간이 짧을수록 높은 우선순위가 할당되므로 임의의 순간에 실행되는 태스크는 실행이 완료되지 않은 태스크들 중에서 마감시간이 가까운 것이 선택된다. 또, 정적 알고리즘과 달리 태스크의 우선순위가 시간에 따라 변하게 된다[8].

2.2.4 최소 유휴시간(Least-Laxity-First) 알고리즘

동적 우선순위 할당 알고리즘으로 유휴시간(Laxity time : 마감시간을 만족시킬 때까지 기다릴 수 있는 시간)이 가장 작은 쓰레드가 먼저 선택되어 수행되게 된다. 이 유휴시간이 0보다 작은 값을 가지게 되면 해당 쓰레드는 마감시간을 놓친 것을 뜻한다.

3. Mach 커널의 재구성

3.1 Mach 커널

Mach 커널은 Carnegie Mellon 대학에서 연구되어 발표된 마이크로커널로 커널 모드에서 수행되는 코드의 크기를 최소로 하고 사용자 모드에서 응용프로그램으로 수행되는 코드의 양을 최대화하여 커널 모드에는 제한된 기능만을 제공해준다. 이 커널은 코드의 완전한 공개로 운영체제를 연구하는 여러 분야에서 사용되고 있다. 본 논문에서는 Mach 상에서 실시간 스케줄

링 가능하도록 커널을 재구성하기 위해서 Mach의 실행 환경과 스케줄링 방식을 분석하고 수정된 자료구조를 제시한다.

3.1.1 Mach의 실행 환경

Mach 운영체제는 UNIX 운영체제의 프로세스 모델[9]을 태스크와 쓰레드로 분리하였다[10]. 태스크는 자원의 집합으로 각 실행 단위인 쓰레드가 공유하는 가상 기억공간, 프로세스간 통신을 위한 포트(port) 등의 자원들과 쓰레드들을 포함하고 있다. 쓰레드는 실행 및 스케줄링의 단위이며 각자의 스택과 태스크의 자원을 가지고 병행적으로 수행된다. 이 구조는 UNIX의 프로세스에 비하여 문맥교환시 적은 오버헤드를 가지며 효율적인 병렬화와 자원공유 등의 여러 가지 이점을 가지며, 또한 다중프로세서 시스템에 적용하기에 적합하다. 또, 쓰레드간의 통신을 위해서 포트라는 도구를 제공해서 커널 내 모든 서비스, 자원, 기능 등은 포트에 의하여 표현되며 해당 포트에 메시지를 보냄으로서 조작될 수 있다[11].

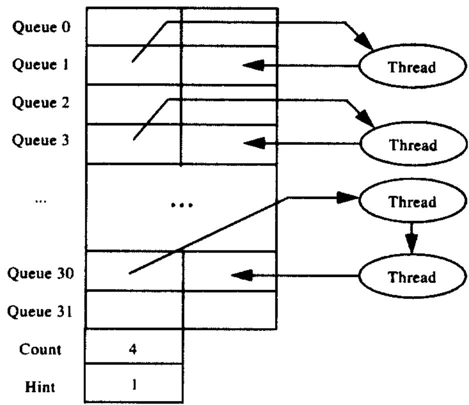
실행을 기다리는 쓰레드들은 실행 대기열(run queue)에서 대기한다. 실행 대기열은 우선순위에 따라 32개의 큐들로 구성되어 있으며 실행 대기열의 자료구조에는 'Count'와 'Hint' 변수가 있다[12]. Count는 실행 대기열의 총 쓰레드 수를 나타내고, Hint는 가장 높은 우선순위의 쓰레드가 있는 실행 대기열의 위치를 나타낸다. 실행 대기열에서 우선순위가 가장 높은 쓰레드를 찾을 때에는 Hint가 가리키는 곳부터 찾기 시작한다.

(그림 1)은 4개의 쓰레드를 갖는 실행 대기열의 구조에 대해서 보여주고 있다. 그림에서 보는 것과 같이 Hint가 1을 나타낸다는 것은 실행 대기열 0에는 쓰레드가 없으므로 이곳에서는 가장 높은 쓰레드를 찾기 위해서 탐색을 하지 않는다.

3.1.2 Mach의 스케줄링

Mach 운영체제에서는 쓰레드를 스케줄링하기 위해서 쓰레드간의 프로세서 시간을 공평하게 분배하는 시분할 방식과 쓰레드에 고정된 우선순위를 부여하는 고정 우선순위 방식 이렇게 두 가지 스케줄링 방식을 사용한다.

각 쓰레드는 우선순위를 가지고 있으며 이 우선순위는 그 쓰레드의 중요도를 나타낸다. 우선순위에는 기본 우선순위(base priority)와 스케줄러 우선순위(scheduler priority)가 있는데 두 가지 모두 0에서 31까지의 정수 값을 갖는다. 이 때, 0이 가장 높은 우선순위를 나타낸다. 각 쓰레드의 기본 우선순위는 고정되어 있어서 실행 중에도 값이 변하지 않는다. 반면 스케줄러 우선순위는 실제로 스케줄러가 사용하는 것으로서 일반적으로 실행 중에 값이 변화한다.



(그림 1) 실행 대기열의 구조

시분할 정책의 목적은 쓰레드들이 프로세서를 공평하게 공유하도록 하는 것이다. 이 정책에서 쓰레드의 스케줄러 우선순위의 값은 기본 우선순위 값에 최근에 프로세서 사용시간 등에 따른 가변적인 수치를 더한 값으로 나타낸다. 따라서, 프로세서 사용시간이 긴 쓰레드는 시간이 지남에 따라 스케줄러 우선순위가 낮아지게 된다. 고정 우선순위 정책은 특정 쓰레드에 고정된 우선순위를 부여하여 우선적으로 처리될 수 있도록 해준다. 이 정책에서 스케줄러 우선순위는 프로세서 사용시간에 관계없이 언제나 기본 우선순위의 값을 가진다.

우선순위를 계산하기 위해서는 쓰레드의 프로세서 사용시간과 프로세서 집합의 부하값(sched_load)을 알아야 한다. 프로세서 사용시간은 쓰레드 자료구조에 기록되어 있고 프로세서 집합의 sched_load는 프로세서 집합 안에 있는 총 쓰레드 수를 프로세서 집합의 프로세서 수로 나눈 값으로 주기적으로 구해진다. 즉, 프로세서 하나가 처리해야하는 쓰레드의 수를 나타낸다. sched_load 값이 급격하게 변하는 경우, 이를 사용하는 우선순위의 변화도 크기 때문에 이를 방지해서 원만한 변화 값을 나타내기 위해서 계산시 특정 값을 곱해

준다. sched_load 값을 구하는 방법은 다음과 같다.

```
load_now = (nthread << 7)/ncpus    (nthread > ncpus)
         = 128                     (nthread <= ncpus)
pset→sched_load = (pset→sched_load + load_now)>>1
```

먼저, 현재의 부하값을 나타내는 load_now 값을 구한다. 여기서 nthread는 프로세서 집합내의 쓰레드 수이고, ncpus는 프로세서 집합의 프로세서 갯수이다. pset->sched_load는 프로세서 집합의 부하값을 나타낸다.

우선순위의 재계산은 쓰레드의 매 시간 할당량(time quantum)이 끝나는 경우나 에이징(aging)시 이루어진다. 에이징은 높은 우선순위를 갖는 쓰레드들이 CPU를 독점하게 되는 현상을 방지하게 하기 위해서 행해진다. 우선순위의 재계산 방법은 쓰레드의 스케줄링 정책이 무엇인가에 따라 다르다. 시분할 정책의 경우, 프로세서 사용시간(sched_usage)은 다음의 방식으로 계산한다.

```
thread→sched_usage = thread→sched_usage + delta ×
                    쓰레드가속한 프로세서 집합의 sched_load
thread→sched_usage = thread→sched_usage × (5/8)ticks
```

여기서 delta는 쓰레드가 이번 할당 동안에 커널 모드 및 사용자 모드에서 프로세서를 사용한 시간으로 단위는 10^{-6}이다. ticks는 쓰레드의 우선순위가 갱신된 최근의 시간과 현재 시간의 차이 값을 나타낸 값이다. 시분할 정책에서는 스케줄링 우선순위(sched_pri) 값은 sched_usage 값을 이용하여 쓰레드가 생성될 때 지정되는 기본 우선순위 값에 더한 값으로 나타낸다.

```
thread→sched_pri = thread→priority+(thread→sched_usage>>25)
```

고정 우선순위 정책에서의 스케줄링 우선순위는 sched_usage 값과는 관계없이 언제나 기본 우선순위(thread→priority) 값만을 가지고 구한다.

```
thread→sched_pri = thread→priority
```

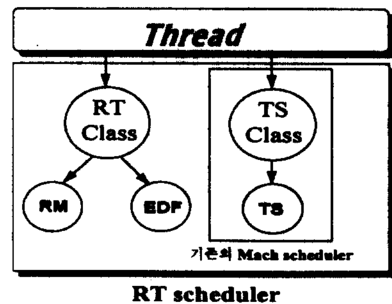
이러한 기존 Mach의 시분할 정책의 우선순위 할당 방식으로는 마감시간안에 실시간 쓰레드의 수행을 보장받지 못하므로 본 논문에서는 실시간 스케줄링이 가능하도록 Mach 커널을 다음과 같은 방식으로 재구성하도록 제안한다.

3.2 수정된 커널 구조

3.2.1 수정된 자료구조

본 논문에서는 기존에 있던 시분할 쓰레드와 추가하고자 하는 실시간 쓰레드가 공존하는 것으로 가정하였다. 이를 위해서 쓰레드에 대한 클래스 개념을 사용하였다. 실시간 클래스의 쓰레드들은 독자적인 우선순위를 가지고 실행 대기열을 가진다. 또, 시분할 쓰레드보다는 높은 우선순위를 가진다.

수정된 Mach 커널의 스케줄링 정책은 기존의 쓰레드들을 위한 시분할 정책에 실시간 쓰레드들을 위한 실시간 스케줄링 정책 중 비율 단조 방식과 마감시간 우선 방식을 추가 하였다. 스케줄러는 기존의 스케줄러에 실시간 쓰레드들을 스케줄하는 실시간 스케줄러를 추가하였다. 실시간 스케줄러는 우선순위가 높은 실시간 쓰레드들을 스케줄하다가 더 이상의 실행시킬 실시간 쓰레드가 없는 경우, 기존의 스케줄러가 시분할 쓰레드들을 스케줄하도록 고안하였다. 고안된 방식을 그림으로 도식화하면 다음과 같다.



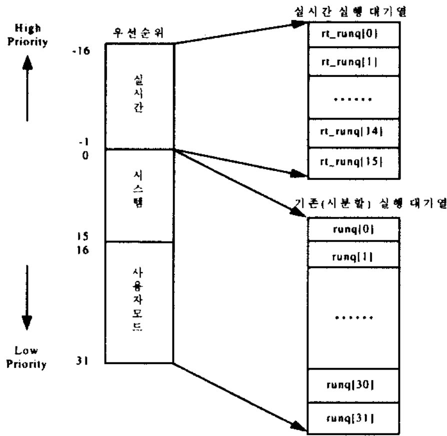
(그림 2) 실시간 스케줄링을 위한 수정된 Mach 커널의 구조

다음은 실시간 쓰레드를 구현하기 위한 수정된 자료구조를 보여주고 있다.

```
typedef struct rt_thread {
    char      p_class;      /* 실시간 클래스 */
    u_int     period;       /* 쓰레드의 주기 */
    u_int     priority      /* 우선순위 */
    u_int     exec_time;    /* 수행시간 */
    u_int     qtm_left;     /* 남은 수행시간 */
    u_int     deadline;     /* 마감시간 */
    u_int     p_cpu;        /* CPU 사용량 */
    boolean   schedulable;  /* 스케줄링 가능 여부 */
}
```

(그림 3) 실시간 쓰레드의 자료구조

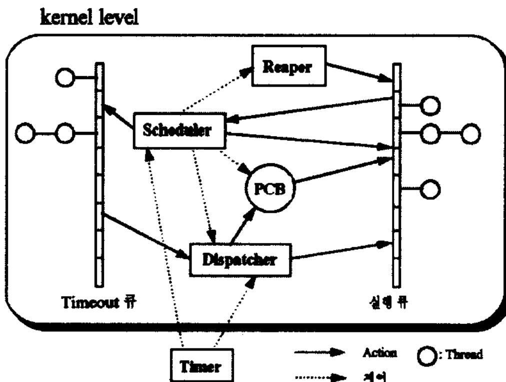
실시간 클래스를 위해서는 시분할 클래스보다 높은 16개의 우선순위를 추가하였고 실시간 실행 대기열을 만들었다. (그림 4)는 우선순위와 실행 대기열의 관계를 보여주고 있다.



(그림 4) 우선순위와 실행 대기열의 관계

3.2.2 실시간 스케줄러

본 논문에서 구현된 스케줄러는 (그림 5)와 같이 크게 네 부분으로 구성이 되는데, 쓰레드를 스케줄해주는 스케줄러, 각 주기마다 쓰레드들을 활성화 시켜주는 디스패처(dispatcher), 스케줄러와 디스패처에 타이머 서비스를 하기 위한 타이머, 그리고 수행이 완료된 쓰레드를 종료시켜주는 리퍼(reaper)로 구분된다.



(그림 5) 실시간 스케줄러의 구성

스케줄러는 0.1초 간격으로 타이머의 신호를 받아 활성화되고 실행 대기열에서 쓰레드를 선택하여 수행시켜 준다. 수행 중인 쓰레드가 아직 끝나지 않았으면 수행 중인 쓰레드를 선점하여 실행 대기열에 다시 삽입하고, 수행을 마쳤으면 time_out 큐에 삽입해 준다. 디스패처는 0.1초마다 타이머의 신호에 의해 활성화되고, 대기 상태에 있는 쓰레드들을 검사하여 실행하여야 할 주기적인 쓰레드들을 time_out 큐에서 실행 대기열과 PCB(Process Control Block)에 삽입시켜 준다. PCB는 수행중인 실시간 쓰레드들에 대한 정보를 갖는다. 이 정보에는 각 실시간 쓰레드들의 시간 제약사항이 포함된다. (그림 6)은 PCB의 자료구조를 보여주고 있다.

```
struct PCB{
   thread_t    thread_id;
   u_int       period;      /* 주기 쓰레드의 주기 */
   u_int       priority;          /* 우선순위 */
   u_int       c_time;      /* 수행시간 */
   u_int       left_time;   /* 잔여 수행시간 */
   u_int       p_cpu;       /* CPU 사용시간 */
   u_int       deadline;    /* 마감시간 */
   }
```

(그림 6) PCB의 자료구조

타이머는 스케줄러와 디스패처에 시간에 관련된 정보를 제공해준다. 이 정보는 매 0.1초마다 신호로 전달되어 스케줄러와 디스패처를 활성화시킨다. 리퍼(reaper)는 수행이 완전히 종료된 쓰레드를 PCB에서 제거시켜주는 역할을 한다.

4. 실시간 스케줄러의 설계 및 구현

실시간 운영체제에서 태스크의 스케줄링은 매우 중요한 역할을 수행한다. 실시간 시스템에서는 각각의 태스크들마다 마감시간이 주어져 있고 이 태스크들이 마감시간 내에 수행되지 못하는 경우에는 큰 피해를 입을 수 있으므로 반드시 지켜져야 한다. 지금까지 마감시간을 보장하기 위한 여러 스케줄링 방법[5]들이 연구되어 왔으며 본 논문에서는 이런 방법 중 우선순위가 고정적으로 주어진 주기 태스크 집합들에 대한 정적 스케줄링 알고리즘들 중 최적 알고리즘이라고 알려진 비율 단조 스케줄링 알고리즘과 태스크의 수행 중에 마감시간에 가까울수록 우선순위를 높게 해주는 동적 우선순위 할당 방법을 사용하는 마감시간 우선 스케줄링 알고리즘에 대해서 기술한다. 또, 기존의 마감

시간 우선 스케줄링 알고리즘이나 비율 단조 스케줄링 알고리즘에서 실시간 태스크들의 집합이 마감시간을 보장받는지를 검사하는 방식을 확장하여 좀 더 정확한 스케줄링 검사 방법을 제시하며, Mach 운영체제 상에서 확장된 스케줄가능성 검사를 사용하여 마감시간 우선 알고리즘과 비율 단조 알고리즘을 기반으로 스케줄하는 실시간 스케줄러를 설계하고 구현한다.

4.1 실시간 스케줄링 정책의 설계

4.1.1 마감시간 우선 스케줄링 정책

실시간 스케줄링은 주로 마감시간 우선(Earliest Deadline First) 스케줄링 알고리즘과 비율 단조(Rate Monotonic) 스케줄링 알고리즘을 근간으로 발전하여 왔다. 마감시간 우선 알고리즘에서는 각 태스크에게 동적(dynamic)으로 우선순위가 부여되는데 매 스케줄링 시점에서 볼 때 마감시간까지의 거리가 가장 짧은 태스크에게 높은 우선순위가 주어지는 선점 가능 스케줄링 알고리즘이다.

실시간 시스템에서 어떤 스케줄링 알고리즘에 의해 모든 응용 태스크가 마감시간을 보장받을 수 있다면 태스크 집합은 "스케줄 가능하다(schedulable)"고 하고, 그렇지 않은 경우 "스케줄 가능하지 않다"고 한다.

마감시간 우선 스케줄링 알고리즘에서는 다음의 조건이 만족되면 주기적으로 발생하는 독립적인 n개의 태스크들은 스케줄 가능하다고 할 수 있다. 이 때, 태스크의 주기는 T_i로 수행 시간은 C_i로 표기한다($1 \le i \le n$).

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \ldots\ldots + \frac{C_n}{T_n} \le 1 \qquad (1)$$

4.1.2 비율 단조 스케줄링 정책

비율 단조 스케줄링 알고리즘에서는 각 태스크들이 독립적이라고 가정되며 각 태스크에게 정적(static)으로 우선순위를 부여하고 이 때 주기가 짧은 태스크일수록 높은 우선순위가 주어지는 선점가능한 스케줄링 알고리즘이다. 이 알고리즘은 우선순위가 고정적으로 주어진 태스크 집합들에 대한 스케줄링 알고리즘들 중 최적 알고리즘으로 알려져 있다[13].

단일 프로세서 환경에서 비율 단조 스케줄링 알고리즘을 사용하는 경우 태스크 집합의 스케줄가능성은 주어진 주기 태스크들의 집합의 프로세서 이용률(utilization, U)이 $n(2^{1/n} - 1)$ (n은 태스크의 수)보다 작거나 같으면 스케줄링이 가능하다고 확인되며 이에 의해 확인되지 못하는 경우는 동시에 시작된 각 태스크의 첫 번째 job의 수행시간(completion time)을 구한 값이 마감시간보다 작은가의 여부로 확인될 수 있다.

Liu와 Layland는 비율 단조 스케줄링에서의 주기 태스크들의 스케줄가능성을 제안하였다[14]. 그 내용은 다음과 같다.

주기 태스크들의 집합 $\tau = \{\tau_1, \tau_2, \ldots\ldots, \tau_n\}$가 고정 우선순위를 사용하는 알고리즘에 의해 스케줄링된다고 할 때 τ_1의 우선순위가 가장 높고 이하 우선순위는 내림차순으로 태스크들이 정렬이 되어있다고 가정한다.

τ_i의 주기는 T_i로, 수행 시간(completion time)은 C_i로 표기하면 태스크의 프로세서 이용률(utilization, U_i)은 C_i / T_i으로 표시될 수 있고, 태스크 집합의 프로세서 이용률은

$$U = \frac{C_1}{T_1} + \frac{C_2}{T_2} + \ldots + \frac{C_i}{T_i} + \ldots + \frac{C_n}{T_n} \qquad (2)$$

로 표시될 수 있다.

이 때, 다음과 같은 조건이 성립하면 독립적인 주기 태스크들의 집합은 각 태스크의 마감시간을 만족할 수 있다.

$$U \le U(n) = n(2^{1/n} - 1) \qquad (3)$$

그러나, 태스크 집합의 프로세서 이용률(U)이 U(n)보다 크고 1보다 작으면 태스크 집합이 스케줄링 가능한지 판단을 할 수 없다. 이 때에는 좀 더 정확한 측정 방법이 사용되어야 한다.

최악의 경우의 수행 시간은 Tindell 등에 의해 제안[15]된 다음과 같은 점화식으로 계산할 수 있다.

τ_i의 최악의 경우의 수행 시간(completion time, W_i)은 다음과 같은 점화식으로 계산할 수 있다.

$$W_i(n+1) = C_i + \sum_{j<i} \left\lceil \frac{W_i(n)}{T_j} \right\rceil C_j, \quad W_i(0) = 0 \qquad (4)$$

이 때, $W_i(n+1) = W_i(n)$이고 $W_i(n+1) \le T_i$이면 τ_i는 스케줄링이 가능하고 그렇지 않은 경우에는 τ_i는 마감시간을 놓치게된다.

Liu와 Layland, Tindell이 제안한 방식에서는 다음과 같은 제한된 사항을 가지고 있다.

- 태스크들의 전환시 발생하는 오버헤드는 0
- 마감시간은 항상 주기의 끝
- 우선순위 역전 현상(priority inversion)은 고려치 않음

따라서, 좀 더 정확한 스케쥴가능성 검사를 하기 위해서는 위의 사항들도 모두 고려되어야 한다. 본 논문에서는 위에 나타난 제한된 사항들을 모두 고려한 확장된 스케쥴가능성 검사를 제안한다.

4.1.3 개선된 스케쥴가능성 검사

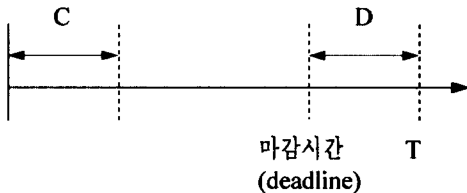
태스크 전환시 발생하는 오버헤드(S)를 고려한다면 한 태스크가 낮은 우선순위의 태스크를 선점하고자 하는 경우, 낮은 우선순위 태스크의 수행 상태를 저장할 때 한 번, 높은 우선순위 태스크를 실행시킬 때 한 번 이렇게 두 번의 태스크 전환시 발생하는 오버헤드를 수행 시간에 더해주어야 한다. 따라서, 태스크 집합의 프로세서 이용률(U)은 다음과 같이 수정되어야 한다.

$$U = \sum_{1 \le i \le n} \frac{C_i + 2S}{T_i} \quad (5)$$

또, 태스크가 주기 이전에 마감시간을 가진다면 다음조건을 만족시키게 된다.

$$C_i \le T_i - D_i \text{ 또는 } C_i + D_i \le T_i \quad (6)$$

이 때, 마감시간부터 주기까지의 시간 D_i는 "dormant time"이라 한다(그림 7).



(그림 7) 마감시간이 주기와 다른 경우의 timing chart

따라서, 태스크 집합의 프로세서 이용률(U)은

$$U = \frac{C + 2S + D}{T} \quad (n=1)$$

$$U = \sum_{0 < j < i} \frac{C_j + 2S}{T_j} + \frac{C_i + 2S + D_i}{T_i} \quad (2 \le i \le n) \quad (7)$$

이 된다.

우선순위 역전현상은 높은 우선순위 태스크의 수행이 낮은 우선순위 태스크에 의해 지연되는 현상을 말하는데 가령 인터럽트 처리는 비록 긴 주기를 갖고 있어서 비율 단조 우선순위는 다른 일반적인 태스크보다 낮아도 실행 우선순위는 높아서 먼저 실행되어야 하므로 짧은 주기를 갖는 높은 우선순위의 태스크의 실행을 지연시킬 수 있다. 이런 우선순위 역전현상으로 태스크가 마감시간을 놓치는 경우가 발생하게 된다. 그러므로, 좀 더 정확한 스케쥴가능성을 검사하기 위해서는 이런 우선순위 역전현상으로 생기는 지연 시간(Blocking time, B)을 고려하여야 한다.

따라서, 우선순위 역전 현상을 고려한 프로세서 이용률은 다음의 세 가지 시간을 모두 계산해 주어야 한다.

1) 자신의 우선순위보다 높은 우선순위의 태스크가 수행한 시간

$$\sum_{i=1}^{k-1} \frac{C_i}{T_i}$$

2) 태스크 자신의 수행시간

$$\frac{C_k}{T_k}$$

3) 자신의 우선순위보다 낮은 우선순위의 태스크의 실행으로 지연된 시간

$$\frac{B_k}{T_k}$$

이렇게 세 가지 시간을 모두 고려하고 제한된 사항을 모두 고려한 확장된 스케쥴가능성 검사를 위한 프로세서 이용률은 다음과 같다.

UB_Test :

$$U = \frac{C + 2S + D}{T} + \frac{B}{T} \quad (n=1) \quad (8)$$

$$U = \sum_{0 < j < i} \frac{C_j + 2S}{T_j} + \frac{C_i + 2S + D_i}{T_i} + \frac{B}{T_i} \quad (2 \le i \le n)$$

위의 계산으로 얻어진 프로세서 이용률(U)이 $n(2^{1/n} - 1)$ (n은 태스크의 수)보다 작거나 같으면 스케쥴링이 가능하다고 할 수 있다.

또, Tindell이 제안한 방식은 마감시간이 주기 이전 시간이고 우선순위 역전 현상으로 발생하는 지연 시간을 고려한다면 다음과 같이 수정되어야 한다.

CT_Test :

$$W_i(n+1) = B_i + C_i + \sum_{j \in I} \left\lceil \frac{W_i(n)}{T_j} \right\rceil C_j \qquad (9)$$
$$W_i(0) = 0$$

이 때, $W_i(n+1) = W_i(n)$이고 $W_i(n+1) \le T_i - D_i$이면 태스크 τ_i는 스케쥴링이 가능하다고 할 수 있다.

4.2 실시간 스케쥴러의 구현

본 논문에서는 3.1절에서 제시한 확장된 스케쥴가능성 검사를 거쳐 마감시간 안에 쓰레드의 실행이 보장된 실시간 쓰레드를 비율 단조 알고리즘과 마감시간 우선 알고리즘에 의해서 스케쥴링을 행하는 실시간 스케쥴러를 설계하고 구현한다.

실시간 시스템에서는 마감시간이 주어지지 않은 비실시간 쓰레드들이 존재할 수 있다. 따라서, 기존의 Mach 상에서의 시분할 쓰레드들에 대한 수행도 함께 고려하여야 한다. 이들 비실시간 쓰레드들은 실시간 쓰레드들이 수행되고 있지 않을 때 수행될 수 있다. 구현한 스케쥴러는 주기적으로 실행되는 실시간 쓰레드들을 중심으로 스케쥴하고, 실시간 쓰레드들이 수행되지 않는 동안에 비실시간 쓰레드들이 기존의 Mach 상의 스케쥴링 정책으로 수행되도록 하였으며 실시간 쓰레드의 새로운 주기가 되어서 실시간 쓰레드를 스케쥴할 필요가 있을 때에는 즉시 커널에 이 사실을 알려야한다. 또, 실시간 쓰레드들을 위한 실행 대기열과 종료를 마친 쓰레드들이 다음 주기에서의 실행을 기다리는 time_out 큐를 설계하였다. 스케쥴링 방식은 프로세서 집합 단위로 마감시간 우선 스케쥴링 방식과 비율 단조 스케쥴링 방식을 선택적으로 사용 가능하게 하였다. 실시간 쓰레드들은 스케쥴하기 전에 마감시간을 만족하는지 스케쥴가능성 검사를 거쳐 마감시간을 만족하는 것이 보장되어야지 스케쥴이 가능하게 된다.

비율 단조 방식과 마감시간 우선 방식에서의 확장된 스케쥴가능성 검사를 위한 알고리즘은 다음과 같다.

```
RM_sched_check()
{
    n = 1;

    REPEAT{
            UB = UB_Test(n);
            if ((0 <= UB)&&(UB <= U(n)))
                  /* U(n) = n*(21/n - 1) */
                  rt_thread[n].schedulable = TRUE;
            else if (UB <= 1){
                  W = CT_Test(n);
                  /* W는 최악의 경우의 수행시간 */
                  if (W <= T[n] - D[n])
                        rt_thread[n].schedulable = TRUE;
                  else
                        rt_thread[n].schedulable = FALSE;
            }else
                  rt_thread[n].schedulable = FALSE;
            n++;
    }UNTIL (n != task_set_no);
}
```

(그림 8) 비율 단조 방식에서의 스케쥴가능성 검사 알고리즘

```
EDF_sched_check()
{
    n = 1;

    REPEAT{
            UB = UB_Test(n);
            if ((0 <= UB)&&(UB <= 1))
                  rt_thread[n].schedulable = TRUE;
            else
                  rt_thread[n].schedulable = FALSE;
            n++;
    }UNTIL (n != task_set_no);
}
```

(그림 9) 마감시간 우선 방식에서의 스케쥴가능성 검사 알고리즘

스케쥴가능성 검사를 마쳐 마감시간이 보장된 쓰레드들은 스케쥴러에 의해서 스케쥴된다.

본 논문에서 제안된 새로운 실시간 스케쥴러의 알고리즘은 다음과 같다.

```
schedule_thread
{
    if (--qtm_left == 0){ /* 현재 쓰레드의 수행 시간 완료 */
          /* 현재 쓰레드를 실행 대기열에서 제거 */
          dequeue(rt_runq, c_thread);
/* 수행했던 쓰레드를 time_out 큐에 삽입 */
enqueue(timeout_q, c_thread));
    }

    if (실시간 실행 대기열이 empty)
          시분할 쓰레드를 스케쥴;
    else {
          hint 변수를 참조하여 다음 수행시킬 쓰레드를 실행
             대기열에서 선택;
          /* EDF인 경우에는 마감시간에 가장 가까운 쓰레드
          가 선택되고 RM인 경우에는 주기가 가장 짧은 쓰레
          드가 선택됨 */
    }
    수행 중인 쓰레드의 CPU 사용량을 증가(p_cpu++);
}
```

(그림 10) 실시간 스케쥴러 알고리즘

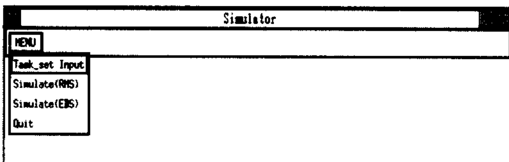
5. 시뮬레이션

5.1 시뮬레이션 모델

본 논문에서는 실시간 스케줄링이 가능하도록 Mach 커널을 재구성하기 위해서 스케줄링에서 가장 핵심적인 부분인 스케줄러를 구현하였다. 이를 위해서 Mach 상에서 비율 단조 알고리즘과 마감시간 우선 알고리즘 방식을 기반으로 확장된 스케줄가능성 검사를 거친 실시간 쓰레드들을 스케줄하는 실시간 스케줄러를 설계 및 구현하였다. 구현된 스케줄러에 의해 쓰레드들이 스케줄되는 모습을 보이기 위한 시뮬레이션 환경을 X 윈도우상에서 GUI 환경으로 설정하고, 모의 실험을 거쳐 Mach 운영체제에 적용시킨다.

시뮬레이션을 위해서 실시간 쓰레드들에 대한 속성을 입력하기 위한 사용자 인터페이스를 구현하였고, 확장된 스케줄가능성 검사를 거쳐 마감시간을 만족하는 실시간 쓰레드들을 스케줄하는 모습을 보여주고 있다.

(그림 11)은 시뮬레이터의 초기화면으로 'Task_set Input' 메뉴에서는 실시간 쓰레드들의 속성을 입력하게 되어있다. 시뮬레이터에서는 태스크들을 최대 5개까지 스케줄할 수 있게 하였고, 입력된 속성들의 단위는 0.1초이며 태스크들의 전환시 발생하는 오버헤드는 0.001초로 가정하였다. 'Simulate(RMS)'와 'Simulate(EDS)' 메뉴에서는 입력된 태스크 집합을 가지고 각 알고리즘 방식대로 스케줄링하는 모습을 보여주고 있다.



(그림 11) 시뮬레이터의 초기 화면

5.2 시뮬레이션 결과

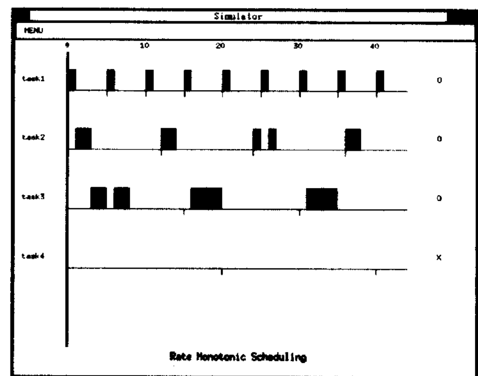
(그림 12)는 입력받은 태스크 집합의 속성을 나타내고 있다. 'Period' 항목은 주기를 나타내고 'Completion Time' 항목은 수행 시간을 나타내고 'Dormant Time'은 주기이전에 마감시간을 갖는 태스크의 마감시간부터 주기까지의 시간이고 'Interrupt process' 항목에 1이라고 표시되면 다른 태스크보다 먼저 실행되어야 하는 interrupt 처리를 요구하는 태스크를 말한다.

input dialog popup

	Period	Completion Time	Dormant Time	Interrupt process
task 1	5	1		
task 2	12	2	1	
task 3	15	4	2	
task 4	20	5		
task 5				

OK QUIT

(그림 12) 태스크 집합의 입력 데이터 (I)



(그림 13) 비율 단조 스케줄링 시뮬레이션 결과 (I)

(그림 13)은 주어진 속성을 갖는 태스크 집합을 비율 단조 스케줄링 방식으로 스케줄하는 모습을 보여주고 있는데 오른쪽의 ○, ×는 각 태스크가 스케줄이 가능한지 아닌지를 나타내고 있다.

〈표 1〉 태스크 집합의 프로세서 이용률 (I)

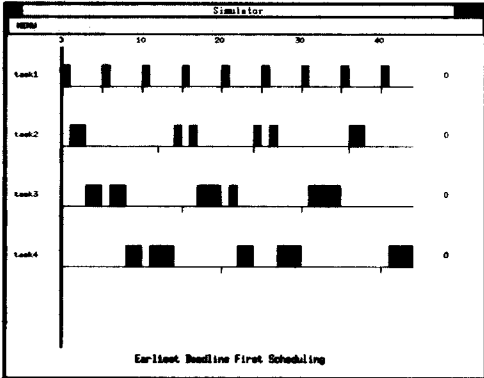
	프로세서 이용률	U(n)
태스크 1	0.2	1
태스크 2	0.451	0.828
태스크 3	0.767	0.78
태스크 4	0.884	0.757

태스크 1, 태스크 2, 태스크 3은 스케줄링 가능성 검사를 통해서 마감시간이 보장되어서 스케줄이 되고 있지만 태스크 4는 프로세서 이용률이 U(4)보다 크고 최악의 수행시간(이 태스크는 22)을 계산해보아도 주기(20)보다 크기 때문에 마감시간을 만족시킬 수 없다.

하지만 마감시간 우선 스케줄링 방식으로는 모든 태스크들을 스케줄링하는 것을 볼 수 있다.

비율 단조 시뮬레이션 결과에서 볼 수 있듯이 높은 우선순위를 갖는 상위 태스크의 새 주기가 수행될 때

마다 하위 태스크의 수행이 중단되는 것을 볼 수 있다. 시분할 태스크들은 실시간 태스크가 수행되지 않는 동안에 기존의 Mach의 스케줄링 방식으로 수행된다.



(그림 14) 마감시간 우선 스케줄링 시뮬레이션 결과 (I)

(그림 15)는 interrupt 처리를 요구하는 태스크(태스크 3)가 있는 경우의 태스크 집합의 입력 데이타를 보여주고 있다.

input_dialog_popup

	Period	Completion Time	Dormant Time	Interrupt process
task 1	7	1		
task 2	14	2	2	
task 3	18	5		1
task 4	26	7		
task 5	35	12	1	

OK QUIT

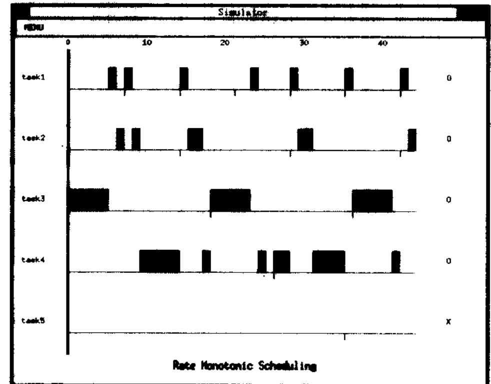
(그림 15) 태스크 집합의 입력 데이터 (II)

이 경우에는 태스크 3이 다른 태스크보다 먼저 실행이 되면서, 태스크 5를 제외한 모든 태스크들이 비율 단조 스케줄링 방식으로 스케줄하든지 마감시간 우선 스케줄링 방식으로 스케줄하든지 마감시간 안에 수행을 마칠 수 있는 것을 볼 수 있다.

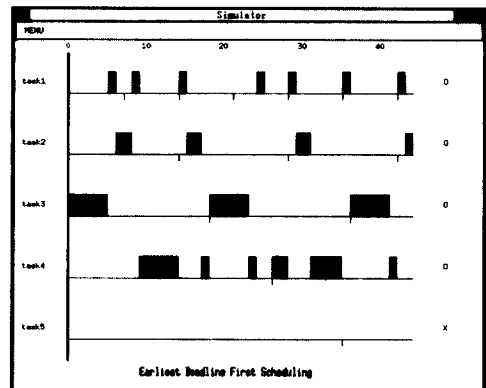
〈표 1〉 태스크 집합의 프로세서 이용률 (II)

	프로세서 이용률	U(n)
태스크 1	0.857	1
태스크 2	0.786	0.828
태스크 3	0.278	1
태스크 4	0.833	0.757
태스크 5	1.205	0.743

태스크 4의 경우에는 프로세서 이용률이 U(4) 보다는 크지만 최악의 수행 시간(이 경우에는 25)이 마감시간(주기에서 dormant 시간을 뺀 시간, 26)보다 작기 때문에 스케줄 가능하다는 것을 볼 수 있다.



(그림 16) 비율 단조 스케줄링 시뮬레이션 결과 (II)



(그림 17) 마감시간 우선 스케줄링 시뮬레이션 결과 (II)

6. 결 론

본 논문에서는 실시간 스케줄링이 가능하도록 Mach 커널을 재구성하는데 있어서 필요한 실시간 스케줄러를 구현하였다. 이를 위해서 첫째, 실시간 운영체제의 요구사항에 따른 구성요소들을 제시하였고 기존의 실시간 스케줄링 알고리즘에 대한 분석을 하였다. 그 결과로 비율 단조 스케줄링 방식과 마감시간 우선 스케줄링 방식을 채택하였다. 둘째, Mach 커널 재구성을 위해서 Mach의 실행 환경과 스케줄링 부분에 대한 분

석을 하였고 시분할 쓰레드만을 지원하는 Mach 커널에 실시간 쓰레드도 스케줄할 수 있도록 수정된 자료구조를 제시하였다. 셋째, 실시간 쓰레드를 스케줄하기 위해서는 마감시간 안에 쓰레드의 수행이 보장되는지를 검사하기 위해서 스케줄가능성 검사를 하는데, 기존 스케줄링 정책에서의 스케줄가능성 검사는 태스크들의 전환시 발생하는 오버헤드를 고려치 않았고, 마감시간은 항상 주기의 끝만을 고려했으며, 우선순위 역전 현상(priority inversion)은 고려하지 않아 정확한 스케줄가능성 검사를 할 수 없었다. 따라서, 본 논문에서는 이를 보안하여 좀더 정확한 검사를 위한 확장된 스케줄가능성 검사 방식을 제시하였다. 넷째, 확장된 스케줄가능성 검사를 통해 마감시간이 보장된 쓰레드들을 마감시간 우선 스케줄링 방식이나 비율 단조 스케줄링 방식으로 스케줄하는 실시간 스케줄러를 구현하였다.

구현된 스케줄러는 모의 실험환경으로 X 윈도우상에서 사용자 인터페이스를 통해 입력받은 태스크들의 속성을 갖고 태스크들을 스케줄하는 모습을 보여주면서 평가를 행했다. 이렇게 구현된 스케줄러는 Mach 커널에 적용하여 Mach 상에서 실시간 스케줄링 가능하도록 하였다.

하지만, 실시간 쓰레드를 좀 더 예측 가능하도록 서비스하기 위해서는 인터럽트에 대한 빠른 처리와 중단가능한 커널, 또한 비동기적인 I/O도 구현하여야 한다. 또한, 현재 ms 단위로 되어있는 시간 단위를 실시간 시스템에 맞추어 us 단위로 낮추는 문제도 고려되어야 한다. 그리고, 본 논문에서는 주기적인 실시간 쓰레드만을 고려하였는데 비주기 실시간 쓰레드들도 함께 고려되어야 하며, 분산처리 시스템에서도 적용시킬 수 있는 방안도 연구되어야 할 것이다.

참 고 문 헌

[1] John A. Stankovic, "Misconceptions About Real-Time Computing," IEEE Computer, pp.10-19, Oct. 1988.

[2] Borko Furht et al., 'Real-Time UNIX Systems : Design and Application Guide,' Kluwer Academic Publishers, 1991.

[3] Krithi Ramamrithm and John A. Stankovic, "Scheduling Algorithm and Operating Systems Support for Real-Time Systems," Proc. of IEEE Vol.82, No.1, pp.55-67, Jan. 1994.

[4] W. Zhao, K. Ramamritham, John. A. Stankovic, "Preemptive Scheduling Under Time and Resource Constraints," IEEE Computers, Vol.C-36, No.8, pp.949-960, Aug. 1987.

[5] S-C Cheng, et al., "Scheduling Algorithms for Hard Real-Time Systems-A Brief Survey," Tutorial : Hard Real-Time Systems, IEEE Computer Society Press, 1988.

[6] John Lehoczky, Lui Sha and Ye Ding, "The Rate Monotonic Scheduling Algorithm : Exact Characterization And Average Case Behavior," Proc. of IEEE Real-Time Systems Symposium, pp.166-171, Dec. 1989.

[7] N. Audsley et al., "Hard Real-Time Scheduling : The Deadline Monotonic Approach," Proc. of IEEE Workshop on Real-Time Operating Systems, 1992.

[8] H. Chetto and M, Chetto, "Some Results of the Earliest Deadline Scheduling Algorithm," IEEE Trans. on Software Eng., Vol.15, No.10, pp.1261-1269, Oct. 1989.

[9] Bach, 'Design of the UNIX Operating Systems,' Prentice-Hall, 1986.

[10] M. J. Accetta, W. Baron, R.V. Bolosky, D.B. Golub, R.F. Rashid, A. Tevanian, and M.W. Young, "Mach : A new kernel foundation for unix development," Proc. of the Summer Usenix Conference, Jul. 8 1986.

[11] J. Boykin, D. Kirschen, A. Langerman, S. LoVerso, 'Programming under Mach,' Addison-Wesley Publishing Company, 1993.

[12] David L. Black, "Scheduling support for concurrency and parallelism in the Mach operating system," IEEE Computer, Vol.23, No.5, 1990.

[13] C. Warren, "Rate Monotonic Scheduling," IEEE Micro, pp.34-38, Jun. 1991.

[14] C. Liu and J. Layland, "Scheduling algorithms for multi-programming in hard real-time environment," Journal of ACM, pp.46-61, Jan. 1973.

[15] K.W. Tindell, A. Burns and A.J. Wellings, "An extendable approach for analyzing fixed priority hard real-time tasks," Journal of Real-Time Systems, Vol.6, No.2, pp.133-152, Mar. 1994.

류 진 열

e-mail : mrlew@lgis.lg.co.kr

1994년 한양대학교 전자계산학과 졸업(학사)

1996년 한양대학교 대학원 전자계산학과 졸업(석사)

현재 LG정보통신 정보시스템연구소 미디어 S/W실 근무

관심분야 : 실시간 시스템, 멀티미디어 통신 등

김 광

e-mail : kkim@mail.hanyang.ac.kr

1994년 한양대학교 전자계산학과 졸업(학사)

1996년 한양대학교 대학원 전자계산학과 졸업(석사)

1999년 한양대학교 대학원 전자계산학과 박사과정 수료

현재 삼일데이타시스템(주) 부설 기술연구소 근무

관심분야 : 실시간 시스템, 지리정보 시스템, 멀티미디어 정보보호 등

허 신

e-mail : shinheu@cse.hanyang.ac.kr

1973년 서울대학교 전기공학과 졸업

1979년 Univ. of Southern California 전자계산학 석사 학위 취득

1986년 Univ. of South Florida 전자계산학 박사학위 취득

1986년~1988년 The Catholic University of America 조교수

1988년~현재 한양대학교 전자계산학과 부교수

관심분야 : 분산처리 시스템, 결함허용 컴퓨터 시스템, 실시간 운영체제