OSF/1 기반 SRT 스케줄러의 성능 향상

이 승 호† · 허 신††

요 약

본 논문은 SRT 시스템에서 고정 우선순위 정책을 사용하는 쓰레드가 기존보다 더 빠른 응답시간과 반환시간을 갖도록 하기 위한 개선된 스케줄링 모델과 알고리즘을 제시하였다. 보조 실행 큐를 첨가하여 고정 우선순위 정책을 사용하는 쓰레드의 빠른 실행 및 에이징(aging)에 사용하였다. 시뮬레이션을 통하여 기존의 방식과 비교하였으며 그 결과, 새로 제안된 모델의 경우, 고정 우선순위 쓰레드의 반응시간과 반환시간이 기존보다 향상되었다.

Performance Improvement of Soft Real-Time Scheduler in OSF/1

Seung-Ho Lee† · Shin Heu††

ABSTRACT

This paper proposes an improved scheduling model and algorithm which threads in the fixed-priority policy have faster response time and turnaround time than existing ones in SRT system. Sub run queue is added into the system and is used for the quick execution and aging of threads in the fixed-priority policy. Using simulation two method are compared. The results shows that the new scheduling model gets threads in the fixed-priority policy to run faster than existing ones.

1. 서 론

실시간 시스템은 응답시간(response time)의 관점에서 볼 때, 연성 실시간(Soft Real-Time:SRT) 경성 실시간(Hard Real-Time:HRT) 시스템으로 나눌 수 있다[1]. HRT 시스템에서는 결과가 산출되어야 하는 시간이 명시되어 시스템은 이러한 데드라인을 만족시키도록 프로세스를 스케줄링 하여야 한다. SRT시스템의 경우, 프로세스의 긴급도(urgency)가 우선순위의 형태로 나타내지며, 실행준비가 된 가장 높은 우선순위의 프로세스-가장 긴급한 프로세스-를 실행시킨다. SRT 시스템에서 사용하는 우선순위에 따른 스케줄링은 빠른 응답을 얻기 위해 사용된다. 그러나 HRT 시스템에서는 시기적절한 응답시간을 갖는 것이 필요하다. 프로세스의 응답이 빠르더라도 데드라인을 만족시키지 못하면 시스템은 올바로 동작하지 못하게 된다.

SRT 시스템은 시분할 정책(time-shared policy)과 고정 우선순위 정책(fixed-priority policy)을 혼합 사용하여 프로세스를 스케줄링 한다. 시분할 정책에 의해 스케줄링 되는 프로세스의 우선순위는 프로세서 사용시간에 따라 점점 낮아진다. 반면, 조정 우선순위 정책을 사용하는 프로세스의 우선순위는 프로세스의 사용시간과 관계없이 언제나 처음에 지정된 고정값을 갖는다. 따라서 이 두 정책을 혼합하여 스케줄링하는 경우, 고정 우선순위 정책의 프로세스는 시분할 정책의 프로세스보다 더 빨리 실행된다[2]. 이러한 방

† 정 회 원:대우통신(주)
†† 정 회 원:한양대학교 전자계산학과 교수
논문접수:1996년 10월 24일, 심사완료:1996년 12월 4일

식으로 스케줄링하는 시스템으로는 REAL/IX[1], 4.3 BSD[3], OSF/1[2, 4] 등이 있다.

본 논문은 OSF/1의 스케줄링을 기존 모델로 하고, 이것의 구조와 알고리즘을 수정하여, 고정 우선순위 정책의 프로세스가 기존 모델보다 더 빠른 응답시간과 반환 시간(turnaround time)을 갖는 새로운 스케줄링 모델을 제안한다. 이 모델은 고정 우선순위 정책 프로세스의 빠른 실행으로 인한 시분할 정책 프로세스의 반환시간 증가를 억제하여 전체 효율을 떨어뜨리지 않도록 한다. 그리고 이 두 모델의 성능을 시뮬레이션을 통하여 비교한다.

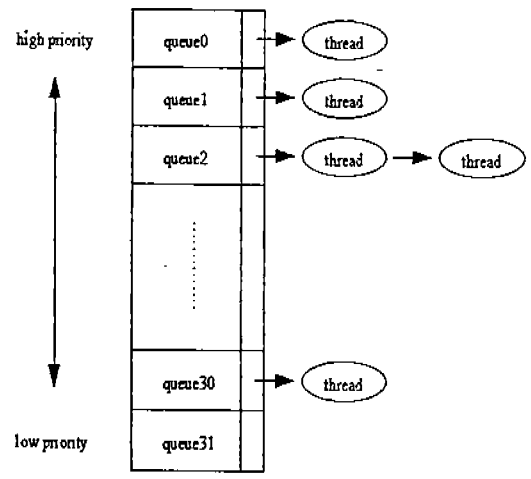
2. OSF/1의 실행 환경

UNIX 운영체제에서, 실행중인 프로그램을 프로세스(process)라 한다. 프로세스는 실행코드를 포함하는 보호된 가상주소공간과, 실행 및 관리에 필요한 여러 가지 정보를 포함하고 있다. 프로세스는 시스템 자원(resource) 소유의 단위로서, 보호된 가상주소공간을 할당받으며, 실행중 필요에 따라 I/O 채널, I/O 디바이스 및 화일 등 시스템의 다른 자원들도 배정 받는다. 또한, 프로세스는 실행 및 스케줄링의 단위로서, 실행의 경로(path)를 가지고 있으며, 실행 상태, 우선순위, 등과 같이 커널이 스케줄링하는 데 필요한 정보들을 가지고 있다[5].

OSF/1에서는 프로세스는 둘로 나누어진다. 자원 소유에 관한 것은 '태스크(task)'가, 실행 및 스케줄링에 관한 것은 '쓰레드(thread)'가 담당한다. 태스크는 시스템 자원 소유의 단위이다. 프로세스와 달리 실행이 가능하지 않으며, 그 태스크에 속한 쓰레드가 실행시 사용가능한 시스템 자원의 집합이다. 쓰레드는 태스크의 환경 하에서 수행되는 실행(execution)의 단위로서 자신이 속한 태스크에 할당된 모든 시스템 자원에 대한 억세스가 가능하다. 그리고 하나의 쓰레드는 오직 한 태스크에만 포함될 수 있다. 하나의 태스크가 여러 개의 쓰레드를 포함할 수 있는데, 이런 경우 쓰레드들은 가상주소공간을 비롯한 태스크의 자원을 공유한다[6].

멀티프로세서 시스템에서, 병렬 프로그램이 특정한 프로세서들 내에서만 실행할 수 있도록 하는 갱스케줄링 정책을 지원하기 위해, 프로세서들을 몇 개의 그룹으로 나눌 수 있다. 이러한 그룹을 프로세서 집합(processor set)이라고 하는데[6], 이는 쓰레드들과 프로세서들이 할당되어지는 단위이다. 각 프로세서는 하나의 프로세서 집합에만 소속되어야 한다. 이와 마찬가지로 모든 쓰레드는 어느 한 프로세서 집합에 배정(assign)되어야 한다. 쓰레드는, 자신이 속한 태스크의 프로세서 집합에 있는 프로세서들에서만 실행될 수 있다.

실행을 기다리는 쓰레드들을 실행큐(run queue)에서 대기한다. 실행큐는 우선순위에 따른 32개의 큐들로 구성되어 있다[그림 2-1]. 실행큐의 종류는 두 가지가 있다. 하나의 전역 실행큐(global run queue)이고 다른 하나는 지역큐(local run queue)이다. 각 프로세서 집합은 전역 실행큐를 가지고 있고, 각 프로세서는 지역 실행큐를 가지고 있다. 전역 실행큐에 있는 쓰레드들은 프로세서 집합에 속한 프로세서에서 실행된다. 특정 프로세서에서의 실행을 요구하는 쓰레드들은 그 프로세서의 지역 실행큐에서 대기한다. 프로세서가 새로운 쓰레드를 실행하고자 하는 경우, 먼저 자신의 지역 실행큐를 검사한다. 거기에 대기하고 있는 쓰레드가 있으면 그 쓰레드가 실행되고, 그렇지 않으면 전역 실행큐에서 대기하고 있는 쓰레드를 실행시킨다. 전역 실행큐도 비어 있으면, 스케줄러는 idle_thread를 실행시키는데, 이 쓰레드는 지속적으로 실행큐를 검사하여 새로 들어오는 쓰레드를 실행되게 한다.



(그림 2) 실행큐의 구조
(Fig. 2-1) Structure of the run queue

3. OSF/1의 스케줄링

OSF/1에서 지원하는 스케줄링 정책은 두 가지가 있다. 하나는 시분할 정책이고, 다른 하나는 고정 우선순위 정책이다. 이들은 프로세서 집합 단위로 적용되며, 한 집합에서 두 정책을 모두 사용할 수 있다. 한 프로세서 집합에 속한 쓰레드들은 그 프로세서 집합에 규정된 정책에 따라 스케줄링 된다. 각 쓰레드는 우선순위를 가지고 있는데, 이것은 그 쓰레드의 중요도를 나타낸다. 우선순위에는 기본 우선순위(base priority)와 스케줄러 우선순위(scheduler priority)가 있는데[7], 둘 다 0에서 31 사이의 정수 값을 갖는다. (우선순위의 정수 값이 작을수록 좋음.) 각 쓰레드의 기본 우선순위는 고정되어 있어서 실행 중에도 값이 변하지 않는다. 반면 스케줄러 우선순위는 실제로 스케줄러가 사용하는 것으로서, 일반적으로 실행 중에 값이 변화한다.

시분할 정책의 목적은 쓰레드들이 프로세서를 공평히 공유하도록 하는 것이다. 이 정책에서 쓰레드의 스케줄러 우선순위 값은 기본 우선순위 값에 최근 프로세서 사용시간 등에 따른 가변적인 값을 더한 것이다. 따라서 프로세서 사용시간이 긴 쓰레드는 시간이 지남에 따라 스케줄러 우선순위가 낮아지게 된다[7, 8]. 고정 우선순위 정책은 특정 쓰레드에 고정된 우선순위를 부여함으로써 우선적으로 처리될 수 있도록 해준다. 이 정책에서 스케줄러 우선순위는 프로세서 사용시간에 관계없이 언제나 기본 우선순위의 값을 가진다[7, 8]. 이러한 방식의 쓰레드들은, 시분할 정책의 쓰레드보다 더 높은 스케줄러 우선순위를 갖게 되므로 더 많은 실행 기회를 갖게 되고, 실행큐에서의 대기시간이 짧아진다.

우선순위를 계산하기 위해서는 쓰레드의 프로세서 사용시간과 프로세서 집합의 부하값(sched_load)을 알아야 한다. 프로세서 사용시간은 쓰레드 자료구조에 기억되어 있고, 프로세서 집합의 sched_load는 주기적으로 구해진다. sched_load는 집합 안에 있는 쓰레드의 수를 집합의 프로세서 수로 나눈 값이다. 즉 프로세서 하나가 처리해야 하는 쓰레드의 수를 나타낸다. 이 경우 sched_load의 값이 급격히 변화하는 경우가 생기게 된다. 그러면 이를 사용하는 우선순위도 변화가 클 수도 있기 때문에 이를 방지하여 원만한 변화값을 나타내게 하기 위해 계산시 특정 수를 곱해준다[7]. sched_load 값을 계산하는 방법은 다음과 같다. 우선 현재의 부하값을 나타내는 load_now를 구한다. 여기서 nthread는 집합내의 실행 중인 쓰레드 수와 대기 중인 쓰레드 수를 더한 값이고, ncpus는 집합의 프로세서의 갯수이다. load_now를 이용하여 sched load를 구한다. pset은 프로세서 집합을 나타낸다.

$$\text{load\_now} = \begin{cases} (\text{nthread} \times 27)/\text{ncpus} & (\text{nthread} > \text{ncpus}) \\ 27 & (\text{nthread} <= \text{ncpus}) \end{cases}$$

$$\text{pset의 sched\_load} = (\text{pset의 sched\_load} + \text{load\_now})/2$$

우선순위의 재계산은 쓰레드가 매 퀀텀(quantum)이 끝난 경우와 에이징(aging)시에 이루어진다. 즉, 매 퀀텀 끝날 때마다 프로세서를 사용한 쓰레드에 대한 우선순위 재계산이 이루어진다. 그리고, 시스템은 주기적으로 에이징을 실시하는데 이 때, 일정 시간동안 실행을 하지 못한 쓰레드의 우선순위가 재계산된다. 우선순위는 그 쓰레드의 스케줄링 정책이 무엇인가에 따라 계산법이 다르다. 시분할 정책의 경우, 우선 쓰레드의 프로세서 사용시간(sched_usage)을 구해야 한다. sched_usage는 최근 사용한 프로세서 시간에 sched_load를 곱한 값을 기존 sched_usage 값에 더한 값이다. 그리고 우선순위 갱신 시간이 현재와 다르면 sched_usage값을 줄여준다. sched_usage는 다음과 같이 구한다. 여기서 thrd는 쓰레드를 나타내며 delta는 쓰레드가 이번 할당 동안에 커널 모드 및 사용자 모드에서 프로세서를 사용한 시간으로 그 단위는 10^{-6} 초이다. 만일 사용시간이 0.1초이면 delta의 값은 100000이 된다. ticks는 쓰레드의 우선순위가 갱신된 최근의 시간과 현재 시간의 차이 값을 초 단위로 나타낸 값이다.

$$\text{thrd의 sched\_usage} = \text{thrd의 sched\_usage} + \text{delta} \times \text{pset의 sched\_load}$$

$$\text{thrd의 sched\_usage} = \text{thrd의 sched\_usage} \times (5/8)^{\text{ticks}}$$

시분할 정책의 경우 스케줄링 우선순위(sched_pri)는 sched_usage를 이용하여 다음과 같이 구한다. 여기서 base_pri는 쓰레드가 생성될 때 지정된 기본 우선

순위 값이다.

thrd의 sched_pri
= thrd의 base_pri +(thrd의 sched_usage/2^{25})

고정 우선순위 정책에서의 스케쥴링 우선순위는 sched_usage값과 관계없이 언제나 동일한 값을 갖는다.

thrd의 sched_pri = thrd의 base_pri

4. 제안된 스케쥴링 모델

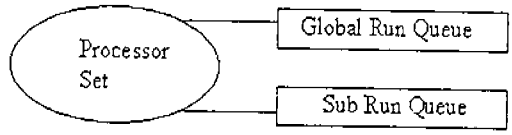
4.1 스케쥴링 모델의 개선

기존의 스케쥴링 모델은 고정 우선순위 정책 쓰레드를 중심으로 볼 때, 두가지 문제점이 발생한다. 첫째로, 우선순위가 같은 시분할 정책 쓰레드와 고정 우선순위 쓰레드는 동일한 응답시간을 갖게 된다. 이것은 처음 쓰레드가 생성되어 첫 수행을 하기 전까지는 두 정책 다 우선순위의 변화가 없기 때문이다. 따라서, 쓰레드가 실행을 완료할 때까지 실행큐에서 대기하는 시간 중에 응답시간의 비중이 클수록 고정 우선순위 정책의 효과는 떨어진다.

둘째로, 고정 우선순위 정책 쓰레드에 관한 문제점이다. 시분할 정책 쓰레드의 에이징은 우선순위 조절을 통해 이루어진다. 즉, 시스템은 일정 시간 이상 실행큐에서 대기하는 쓰레드를 찾아 우선순위를 높여주어 곧 실행될 수 있도록 한다. 이것은 시분할 정책의 경우 프로세서 사용시간에 따라 우선순위가 점점 낮아지기 때문에 가능하다. 그러나 고정 우선순위 정책의 쓰레드는 언제나 그 쓰레드가 가질 수 있는 최상의 우선순위 값-기본 우선순위-을 가지기 때문에 에이징이 불가능하다. 따라서 기존의 모델에서는 이러한 쓰레드가 일정 시간 이상 실행큐에서 대기를 하더라도 실행 기회를 현재보다 더 높여 줄 수 없다.

고정 우선순위 정책에서 발생되는 위의 두가지 문제점-응답시간과 에이징-은 구조적인 문제점으로서 기존의 모델로는 해결하기 어렵다. 기존 모델의 문제점을 해결하기 위해 본 논문에서는 프로세서 집합 구조에 보조 실행큐를 첨가한 새로운 모델을 제시한다[그림4-1]. 따라서, 프로세서 집합은 기존의 전역 실행큐와 새로운 보조 실행큐를 갖는다. 이 보조 실행큐는 전역 실행큐와 마찬가지로 32개의 큐로 구성되어 있다.

이 보조 실행큐에는 고정 우선순위 정책 쓰레드만이 들어갈 수 있다. 그리고 우선순위가 같은 경우 전역 실행큐보다 우선권을 갖는다. 즉, 스케쥴러가 실행할 쓰레드를 선택할 때, 두 실행큐에서 우선순위가 가장 높은 쓰레드가 선택하여 실행하고, 만일 각 실행큐에서 가장 높은 우선순위를 갖는 쓰레드의 우선순위가 같은 경우, 보조 실행큐의 쓰레드를 선택 실행한다.



(그림 4-1) 개선된 프로세서 집합 구조
(Fig. 4-1) Structure of the advanced processor set

4.2 개선된 스케쥴링 알고리즘

기존의 구조에 보조 실행큐가 첨가되었으므로 스케쥴링에 관련된 알고리즘은 다음과 같이 변경된다. 우선, 쓰레드가 새로 생성되었을 때, 이 쓰레드의 스케쥴링 정책에 따라 대기하는 실행큐가 달라진다. 생성된 쓰레드가 시분할 정책을 사용하는 쓰레드이면 전역 실행큐에서 대기하게 하고, 고정 우선순위 정책의 쓰레드이면 보조 실행큐에서 대기하도록 한다. 따라서 같은 우선순위인 경우 고정 우선순위 정책 쓰레드가 먼저 실행된다.

스케쥴러가 실행시킬 쓰레드이면 쓰레드를 선택할 때, 처음에는 프로세서의 지역 실행큐를 검사한다. 그곳에 대기하고 있는 쓰레드가 있으면 그것을 실행시킨다. 그러나 지역 실행큐가 비어있는 경우, 보조 실행큐와 전역 실행큐에서 대기중인 쓰레드를 선택해야 한다[그림 4-2]. 우선 두 실행큐 중에서 어느 한 실행큐가 비어 있는 경우, 비어 있지 않은 실행큐에서 우선순위가 가장 높은 쓰레드를 선택한다. 만일 두 실행큐가 모두 비어 있으면 기존과 같이 idle_thread를 실행시킨다. 그러나 두 실행큐 모두가 대기 중인 쓰레드가 존재하는 경우, 각 실행큐에서 우선순위가 가장 높은 쓰레드를 비교하여 우선순위가 더 높은-우선순위 값이 더 작은-쓰레드를 선택한다. 두 쓰레드

의 우선순위가 같으면, 보조 실행큐의 쓰레드를 선택한다.

```
if (glob_runq의 thread 수 == 0) {
        if (sub_runq의 thread 수 == 0)
            next_thread = idle_thread;
        else
            next_thread = thread in sub_runq;
        }
else {
        if (sub_runq의 thread 수 == 0)
            next_thread = thread in glob_runq;
        else {
            if (thread의 sched_pri in glob_runq
                  < thread의 sched_pri in sub_runq)
                next_thread = thread in glob_runq;
            else
                next_thread = thread in sub_runq;
             }
        }
```

(그림 4-2) 실행된 쓰레드의 선택 알고리즘
(Fig. 4-2) Selection algorithm for running threads

쓰레드의 우선순위 계산 방법은 이전과 동일하다. 그러나 우선순위 계산 후 실행큐에 삽입될 때에는 어느 실행큐에서 대기해야 할 것인지를 결정해야 한다[그림 4-3]. 시분할 정책 쓰레드는 이전과 같이 전역 실행큐에 삽입된다. 그러나 고정 우선순위 정책 쓰레드는 프로세서 사용시간(sched_usage)에 따라 대기할 실행큐가 달라진다. 프로세서 사용시간이 기준 값(limit)보다 작은 경우 쓰레드는 보조 실행큐에 삽입되고, 그렇지 않으면 전역 실행큐에서 대기하게 한다. 여기서 기준값은 적절히 조절된 수치이어야 하는데, 이 값이 너무 크면 보조 실행큐의 크기가 늘어나서 효과가 감소되고, 반대로 너무 작으면 보조 실행큐로의 이동이 어려워지므로 고정 우선순위 정책 쓰레드의 응답시간은 빨리 지나 그 후의 대기시간이 길어지게 된다.

보조 실행큐의 첨가로 인한 효과는 크게 두가지로

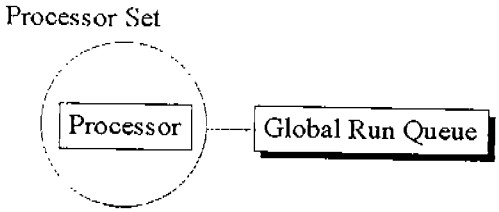
```
if (thread의 policy == TimeShare)
        file this thread into glob_runq;
else {
        if (thread의 sched_usage < limit)
            file this thread into sub_runq;
        else
            file this thread into glob_runq;
        }
```

(그림 4-3) 대기할 실행큐의 선택
(Fig. 4-3) Selection of the waiting run queue

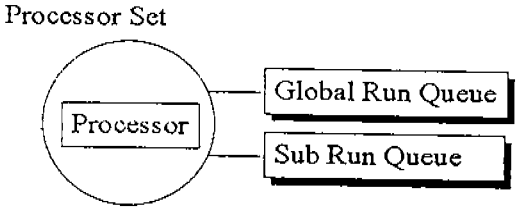
나눌 수 있다. 첫째는 고정 우선순위 정책의 쓰레드가 기존보다 더 빠른 응답시간을 갖게 되는 것이다. 프로세서에 있는 지역 실행큐의 경우를 제외하고, 시분할 정책의 쓰레드가 생성되면 전역 실행큐에서 대기하고, 고정 우선순위 정책의 쓰레드가 생성되면 보조 실행큐에서 대기한다. 같은 우선순위에서는 보조 실행큐가 우선권을 가지므로 새로 생성된 고정 우선순위 정책 쓰레드는 기존의 모델보다 빠른 응답을 할 수 있다. 이러한 우선권 뿐만 아니라, 일반적으로 보조 실행큐의 쓰레드 수가 전역 실행큐보다 작으므로 이로 인해 더욱 빠른 응답이 가능해 진다. 둘째로, 고정 우선순위 정책 쓰레드의 대기시간을 줄일 수 있다. 우선순위는 퀀텀 하나를 사용한 후와, 주기적인 에이징 시에 재계산된다. 고정 우선순위 정책 쓰레드의 우선순위를 재계산할 때, 프로세서 사용시간이 정해진 한계보다 적은 경우, 이 쓰레드를 보조 실행큐로 넣기 때문에, 고정 우선순위 정책 쓰레드의 에이징이 가능해지고, 실행큐에서 대기하는 시간을 줄일 수 있다.

5. 성능 평가 및 비교

시뮬레이션을 위한 기존 OSF/1 스케줄링 모델은 하나의 프로세서로 이루어진 프로세서 집합과 전역 실행큐로 구성된다[그림 5-1]. 본 논문에서 제안한 스케줄링 모델 역시 하나의 프로세서로 이루어진 프로세서 집합으로 이루어 졌으며 전역 실행큐와 보조 실행큐로 구성된다[그림 5-2].



(그림 5-1) OSF/1 스케줄링 모델
(Fig. 5-1) OSF/1 scheduling model



(그림 5-2) 제안된 스케줄링 모델
(Fig. 5-2) Proposed scheduling model

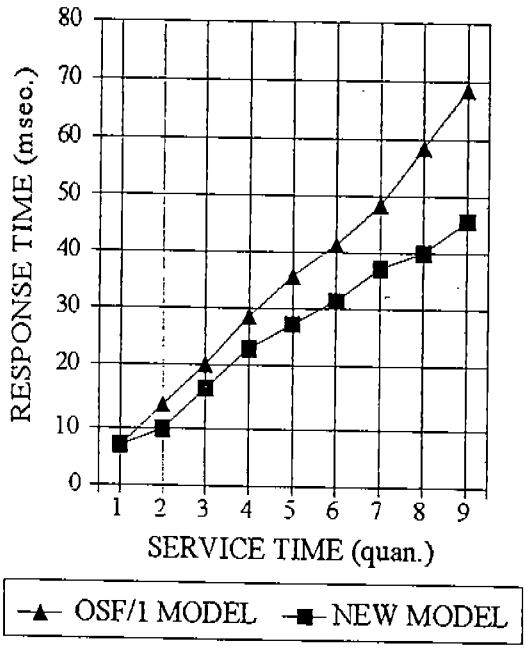
시뮬레이션의 목적이 두 모델의 비교에 있으므로, 스케줄링 시 발생하는 문맥 교환의 시간은 고려하지 않으며, 스케줄링을 수행하는 코드의 실행 시간 역시 무시한다. 그리고 지역 실행큐를 고려하지 않고 모델에서 제외시켰다. 쓰레드의 생성시간은 주어진 값을 평균으로 하는 지수분포를 사용하였고, 쓰레드의 실행시간 역시 주어진 값을 평균으로 하는 지수분포를 사용하였다. 고정 우선순위 쓰레드와 시분할 쓰레드의 비율은 난수 발생 함수를 이용하였다.

OSF/1의 우선순위는 0에서 31까지의 32단계로 나누어져 있다. 여기서 0부터 15까지는 커널 쓰레드에 의해 사용되고, 나머지 16에서 31까지는 사용자 쓰레드에 의해 사용된다. 본 논문에서는 커널 쓰레드의 경우 고려하지 않고 사용자에 의해 생성된 쓰레드의 경우만을 시뮬레이션 한다. 따라서 생성된 쓰레드의 우선순위는 사용자 우선순위 중 제일 상위의 값인 16으로 고정된다. 그리고 새로운 모델에서 우선순위 계산 후 대기할 실행큐를 결정하는 기준값(limit)은 3으로 하였다.

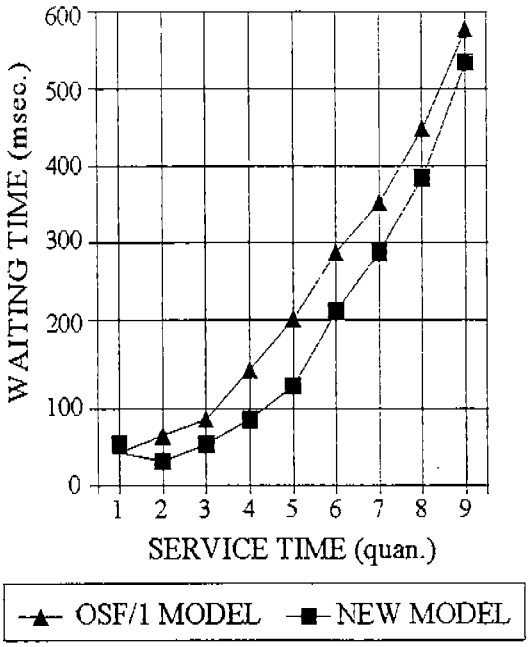
프로세서 집합의 부하값은 1초마다 재계산되고, 에이징은 2초마다 수행된다. 퀀텀의 길이는 100 msec.으로 하였고, 실행큐가 비어 있는 경우, 1msec.의 시간 경과시마다 실행큐를 검사하도록 하였다. 실행시간의 변화에 따른 시뮬레이션의 경우, 고정 우선순위 쓰레드의 비율을 20%로, 평균 생성시간을 1000 msec.로 하고 평균 실행시간을 100 msec.에서 900 msec.까지 변화시켜 그 값을 비교하였다. 두 정책 쓰레드의 생성 비율 변화에 따른 시뮬레이션의 경우, 평균 실행시간을 900 msec로, 평균 생성시간을 1000 msec.로 하고 고정 우선순위 쓰레드의 비율은 10%에서 90%까지 변화시켜 그 결과를 비교하였다. 시뮬레이션은 시뮬레이션 전용 언어인 SIMSCRIPTII.5를 사용하였다.

5.1 실행 시간의 변화에 따른 성능평가 및 비교

고정 우선순위 쓰레드의 경우, 제안된 모델이 기존 모델에 비해 응답시간이 줄어들었음을 알 수 있다[그림 5-3]. 이는 제안된 모델이 고정 우선순위 쓰레드의 빠른 응답을 위한 보조 실행큐를 가지고 있기 때문으로 생각할 수 있다. 실행시간이 커질수록 전역 실행큐의 길이가 커지므로, 보조 실행큐를 사용하는 경우의 응답시간은 더욱 줄어든다. 대기 시간에 있어서도 새로운 모델의 경우가 더 작은 것을 볼 수 있다[그림

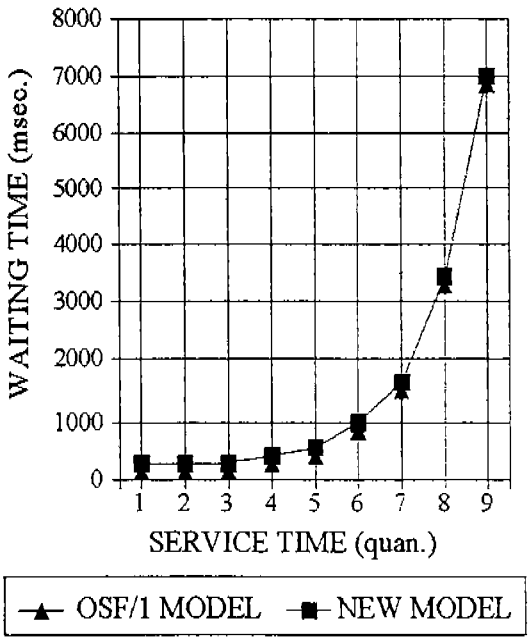


(그림 5-3) 실행시간 변화에 따른 고정 우선 순위 쓰레드의 반응시간
(Fig. 5-3) Graph of execution time vs. response time of fixed-prioroty thread



(그림 5-4) 실행시간 변화에 따른 고정 우선 순위 쓰레드의 대기시간

(Fig. 5-4) Graph of execution time vs. Waiting time of fixed-priority threads



(그림 5-5) 실행시간 변화에 따른 시분할 쓰레드의 대기시간

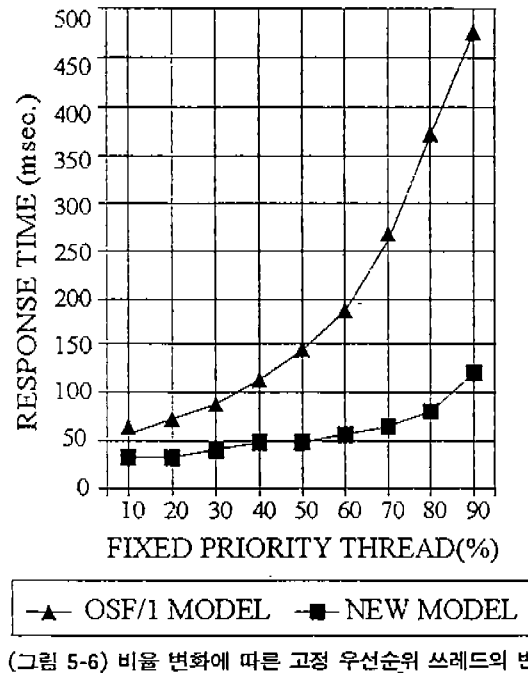
(Fig. 5-5) Graph of running time vs. waiting time of time-sharing thread

5-4]. 보조 실행큐가 고정 우선순위 쓰레드의 에이징에 사용되어 실행큐에서의 대기시간이 줄어들었음을 알 수 있다.

시분할 쓰레드의 경우, 두 방식의 대기시간(waiting time) 차이는 거의 나지 않는 다[그림 5-5]. 이것은 시분할 쓰레드의 프로세서 사용시간에 따른 우선순위 변화의 차이라고 볼 수 있다. 즉, 새로운 모델 상의 시분할 쓰레드가 기존의 것에 비해 프로세서 사용시간에 따른 우선순위 하강 폭이 더 적기 때문이다. 따라서 반환 시간의 차이가 근소하기 때문에 새로운 알고리즘이 시분할 쓰레드의 효율을 떨어뜨리지 않는다고 볼 수 있다.

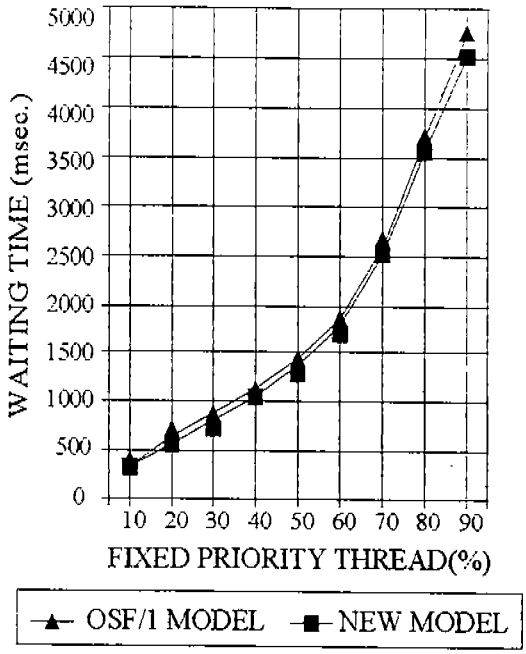
5.2 쓰레드 비율의 변화에 따른

고정 우선순위 쓰레드의 경우, 제안된 모델이 더 빠른 응답 시간을 보여주고 있다(그림 5-6). 기존 모델의 경우 비율이 커짐에 따라 그 시간이 꾸준히 증가하는 반면, 제안된 모델의 경우 그 증가세가 약함을 알 수 있다. 대기 시간도 그 비율에 관계없이 제안

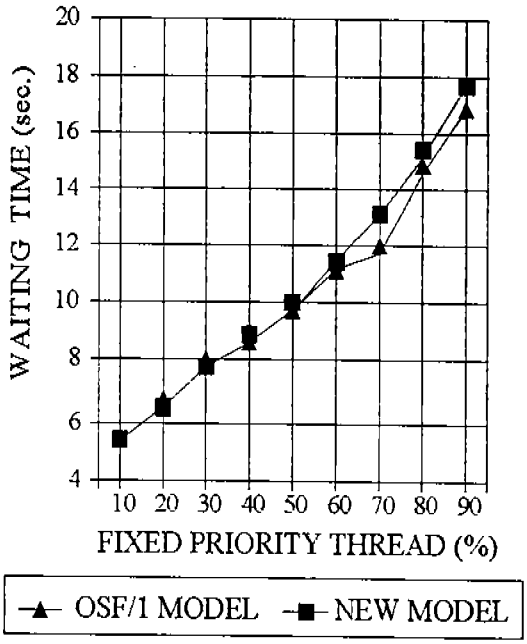


(그림 5-6) 비율 변화에 따른 고정 우선순위 쓰레드의 반응시간

(Fig. 5-6) Graph of ratio vs. response time of fixed-priority thread



(그림 5-7) 실행시간 변화에 따른 시분할 쓰레드의 대기시간
(Fig. 5-7) Graph of execution time vs. waiting time of time-sharing thread



(그림 5-8) 비율 변화에 따른 시분할 쓰레드의 대기시간
(Fig. 5-8) Graph of ratio vs. waiting time of time-sharing thread

된 모델이 일정시간 만큼의 빠른 대기시간을 보여준다(그림5-7). 즉 제안된 모델의 경우, 고정 우선순위 쓰레드의 비율에 관계없이 좋은 효율을 낸다. 시분할 쓰레드의 경우, 대기시간을 비교하여 보면 50%까지는 거의 차이가 없으나 이후부터 새로운 모델의 경우가 대기시간이 더 큰 폭으로 커졌음을 알 수 있다(그림 5-8). 그러나 고정 우선순위 쓰레드의 비율이 더 높은 경우는 두 정책을 혼용하는 효과가 적기 때문에 실제로 이런 상황이 발생할 확률은 거의 없다고 할 수 있다.

6. 결 론

본 논문에서는 SRT시스템에서 고정 우선순위 정책을 사용하는 쓰레드가 기존보다 더 빠른 응답시간 및 반환 시간을 갖도록 하는 새로운 스케줄링 모델을 제시하였다. 그리고 시뮬레이션을 통하여 반응시간과 대기시간을 측정하여 그 성능을 평가하였다. 시스템의 부하량과 고정 우선순위 쓰레드의 비율을 각각 변화시키면서 그 결과를 측정하였다.

기존의 구조에 보조 실행큐를 첨가하여 새로 생성된 고정 우선순위 쓰레드가 대기하도록 하고, 쓰레드의 우선순위가 같은 경우, 보조 실행큐에 우선권을 주었다. 그리고 한번 실행된 쓰레드는 전역 실행큐에서 대기하도록 하였다. 또한 고정 우선순위 쓰레드의 에이징에도 보조 실행큐를 사용하였다. 그 결과, 시스템의 부하량과 고정 우선순위 쓰레드의 비율을 각각 변화시킨 경우 제안된 모델의 고정 우선순위 정책 쓰레드가 더 빨리 실행되었다. 즉, 고정 우선순위 정책 쓰레드의 반응시간과 반환시간은 줄어들었고, 이에 따른 시분할 쓰레드의 반환시간의 증가도 미미하였다. 앞으로 새로운 모델을 멀티프로세서 시스템에 적용하는 연구가 필요하다고 본다.

참 고 문 헌

[1] Broko Furth, Dan Grostick, David Gluch, Guy Rabbit, John Parker and Meg McRoberts, Real-Ti me UNIX Systems, Kluwer Academic Publishers, 1991.

[2] Tomas W. Doeppner, "OSF/1 Internals", Tutorial

presented at USENIX 1992 Winter Technical Conference, San Francisco, 20-24 Jan., 1992.

[3] S. J. Leffler, M. K. McKusick, M. J. Karels and J. S. Quartermann, The Design and Implementation of the 4.3BSD UNIX Operating System, Addison Wesley, 1989.

[4] Linda Mui and Steve Talbott, Guide to OSF/1: A Technical Synopsis, O'Reilly & Associates, Inc., 1991.

[5] M. J. Bach, The Design of the UNIX Operating System, Prentice-Hall, 1986.

[6] Open Software Foundation, Design of the OSF/1 Operating System, OSF/1 Documentation, Nov. 1990.

[7] David L. Black, "Scheduling Support for Concurrency and Parallelism in the Mach Operating System", IEEE Computer, May 1990.

[8] David L. Black, "Processors, Priority, and Policy :Mach Scheduling for New Environments", Proceeding of USENIX 1991 Winter Conference, Jan. 1991.

[9] Avadis Tevanian, Jr and Richard F. Rashid, David B. Golub, David L. Blacck, Eric Cooper and Michael W. Yung, "Mach Thread and Unix kernel:The Battle for Control", Computer Science Department, Carnegie Mellon University, Technical Report No. CMU-CS-87-149, Aug. 1987.

[10] Open Software Foundation, OSF/1 Technical Seminar, Internal document in OSF, 1990.

[11] Avadis Tevanian, "MACH Tutorial", Tutorial presented at USENIX 1990 Winter Technical Conference, Washington D. C., 22-26 Jan., 1990.

[12] Dror G. Feitelson and Larry Rudolph, "Distributed Hierarchical Control for Parallel Processing", IEEE Computer, May 1990.

[13] Prabhat K. Andleigh, UNIX System Architecture, Prentice-Hll, 1990.

[14] William Stallings, Operating Systems, Macmillan Publishing Company, 1992.

[15] Harvey M. Deitel, Operating Systems, Addison-Wesley, 1990.

이 승 호

1991년 한양대학교 전자계산학과 졸업(학사)

1993년 한양대학교 전자계산학과 석사학위 취득

1993년~현재 대우통신 소프트웨어 사업부 연구원

관심분야:분산처리 시스템, 실시간 운영체제임

허 신

1973년 서울대학교 전기공학과 졸업

1979년 Univ. of Southern California 전자계산학 석사학위 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 운영체제임