실시간 시스템에서 공유자원의 효율적 사용을 위한 혼합형 우선순위 작업자 모델

박 홍 진† · 천 경 아† · 김 창 민††

요 약

최근 들어 많이 사용되어지는 원격 전자회의 시스템이나 멀티미디어 브로드캐스팅과 같은 분산 멀티미디어 어플리케이션을 지원하기 위해서는 시스템이 어플리케이션의 시간제약성을 만족시켜주어야 한다. 따라서, 이와 같은 실시간 시스템에서는 시스템의 행위를 예측하고 분석하기 어렵게 하는 우선순위 반전 문제를 해결하여야 하며, 시스템의 오버헤드를 최소화하면서 공유자원에 대한 선점가능성을 높일 수 있는 실시간 서버모델을 사용할 필요가 있다. 현재 동기화에서 주로 사용되는 실시간 서버 모델에는 단일 스래드 서버모델, 작업자 모델 그리고 동적 서버 모델이 있으나 공유자원을 관리하기 위한 효율적인 구조를 제시하고 있지는 못하다. 본 논문에서는 우선순위 반전문제를 해결하기 위하여 우선순위 계승 프로토콜을 이용하고 있으며, 시스템의 오버헤드에 영향을 최소화하면서 서버에 대한 보다 나은 선점가능성을 제공할 수 있고 좀더 빠른 응답시간을 갖는 실시간 서버 모델로서 혼합형 우선순위 작업자 모델을 제안한다. 혼합형 우선순위 작업자 모델은 정적 우선순위 작업자 모델과 동적 우선순위 작업자 모델을 혼합한 형태로서 성능평가 결과 혼합형 우선순위 작업자 모델이 기존의 다른 모델들 보다 좀 더 나은 성능을 보이고 있음을 알 수 있다.

A hybrid prioritized worker model for efficiency of shared resources in the real-time system

Hong-Jin Park† · Kyung-Ah Chun† · Chang-Min Kim††

ABSTRACT

To support multimedia applications such as a multimedia communication systems and multimedia broadcasting, an operating system need to predict their timing-constraints. So, In this real-time systems, we must solve the priority inversion problem that may make the behavior of unpredictable systems and need a real-time server model that provides a better preemptability and minimizes a system overhead. In current real-time systems, the single thread server model, the worker model and the dynamic server model are being used for synchronization but they cannot propose an effective structure for managing shared resources. In this paper, the priority inheritance protocol is used to solve the priority inversion problem and the hybrid prioritized worker model is proposed, which can provide a more effective structure and a faster response time minimizing a system overhead. The hybrid prioritized worker model is to combine the static and the dynamic prioritized worker model, and have a better performance than other models in response time which is an important factor in a real-time system.

† 정 회 원 : 중앙대학교 대학원 컴퓨터공학과
†† 정 회 원 : 성결대학교 컴퓨터공학부
논문접수 : 1999년 6월 1일, 심사완료 : 1999년 11월 9일

1. 서 론

최근 들어 인기를 끌고 있는 원격 전자회의 시스템이나 멀티미디어 브로드캐스팅 시스템과 같은 분산된 멀티미디어 어플리케이션들을 지원하기 위해서 운영체제는 어플리케이션의 시간 제약성(time constraints)과 실시간적 행위(real-time behavior)를 분석하고 예측하여 실시간 어플리케이션들의 요구사항을 만족시키기 위한 스케줄링과 자원관리 정책을 제공할 수 있어야 한다. 특히, 실시간 어플리케이션을 지원하기 위해서는 서버 작업과 같은 공유 자원에 대해 높은 우선순위를 갖는 활동의 최악블로킹 시간(worst case blocking time)을 미리 예측하여 계산하여야 하는데, 이 활동이 계속해서 낮은 우선순위의 활동에 의해 선점당한다면 블록되는 한계를 계산할 수가 없다. 이와 같이 현상을 우선순위 반전문제(priority inversion problem)라고 하며, Ada 랑데뷰(rendezvous)나 세마포 또는 모니터(monitor) 등과 같은 공유자원들을 둘 이상의 작업들이 접근하려 할 경우 우선순위 반전 현상은 피할 수 없는 문제가 된다[1].

이와 같이 실시간 시스템이 예측가능성(predictability)과 스케줄가능성(schedulability)을 제공하는데 있어서 심각한 문제를 유발하는 우선순위 반전 문제를 해결하기 위해서는 우선순위 계승 프로토콜(priority inheritance protocol)을 이용해야 하는데[2], 이러한 해결방법이 요청들에게 보다 나은 서버 작업에 대한 선점가능성(preemptability)을 제공하여 최적의 응답시간(response time)을 제공할 수 있는 실질적인 구조를 제시하고 있는 것은 아니다. 따라서, 서버 작업이 요청 처리를 위해 몇 개의 활동 작업을 갖도록 함으로써 서버에 대한 선점가능성을 향상시킬 수 있는 실시간 서버 모델이 필요하다.

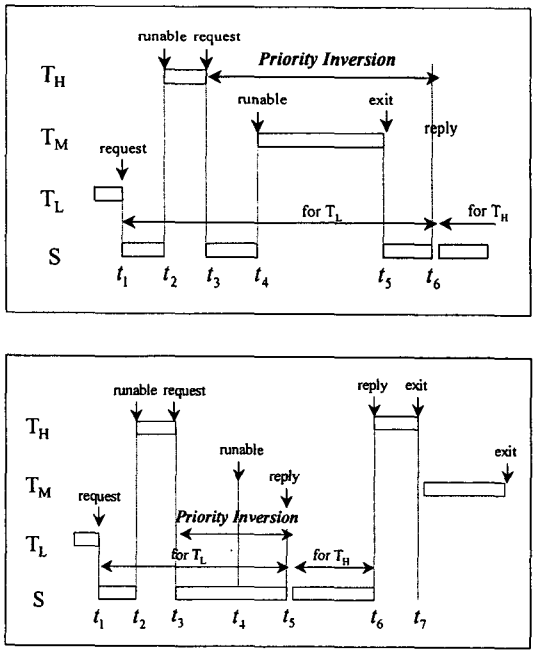
따라서, 본 논문에서는 기존 시스템에서 사용되고 있는 실시간 서버 모델들을 살펴보고, 그에 따른 문제점을 개선한 혼합형 우선순위 작업자 모델(hybrid prioritized worker model)을 제안하여 보다 효과적인 실시간 지원 성능을 보장하기 위한 방안에 초점을 맞추고 있다. 또한, 본 논문의 목적은 실시간 시스템에서 공유자원에 대한 높은 선점가능성을 제공하고, 우선순위 반전 시간을 감소시켜 요청에 대한 응답시간을 최소화시킬 수 있는 실시간 서버를 설계하는데 있다.

본 논문의 구성은 다음과 같다. 2장에서는 기반연구로서 우선순위 계승 프로토콜과 기존에 사용되어지던 실시간 서버 모델들을 살펴보고, 3장에서는 본 논문이 제안한 모델인 혼합형 우선순위 작업자 모델에 대해서 설명한다. 4장에서 제안된 모델에 대한 성능을 평가하고 5장에서는 본 논문의 결론을 맺는다.

2. 기반 연구

2.1 우선순위 계승 프로토콜

여러 작업이 상호작용을 함으로써 하나의 처리를 수행하는 실시간 시스템에서는 자원의 공유를 위해 서로 동기화하는 과정에서 우선순위 반전 문제가 발생될 수 있다[2]. 이러한 우선순위 반전 문제는 높은 우선순위 작업이 낮은 우선순위작업으로 인해 블록 되어 있는 그 한계를 예측해 낼 수 없으므로 실시간 시스템이 예측가능성과 스케줄가능성을 제공하는데 있어서 심각한 문제를 유발한다.



(그림 1) 우선순위 반전 문제와 우선순위 계승 프로토콜

예를 들어, (그림 1)의 위 그림에서 클라이언트 작업으로 T_L, T_M, T_H가 있으며, 우선순위는 낮은 순서라고 하자. 클라이언트 작업으로부터 요청을 받아 서비스를

해주는 서버 작업은 S라고 하고, 가장 낮은 우선순위 작업인 T_L이 서버 작업 S에게 t_1시간에 요청을 보냈다고 가정하자. 그리고 t_2에 가장 높은 우선순위의 T_H가 실행상태가 되어 t_3에 서버 작업 S에게 요청을 보내면, T_L의 요청이 현재 처리 중이므로는 T_L의 요청 처리를 큐에서 기다려야 한다. 이렇게 T_H가 T_L의 요청처리를 완료할 때까지 기다리는 동안 중간 우선순위를 갖는 T_M이 서버 작업 S에 요청을 하지 않고 독자적으로 실행이 되면, 서버작업 S의 우선순위가 가장 낮은 관계로 T_M의 수행이 완전히 끝나는 t_5후에나 서버 S는 T_L의 요청처리를 완료하여 t_6에 응답할 수 있다. T_H에 대한 요청처리는 이와같은 모든 작업들이 끝난 후에나 처리되게 된다. 여기에서 우선순위 반전이 t_3에서 t_6사이에서 발생되었는데, T_M과 같은 작업의 개수가 많아질 경우 그 개수와 실행 시간을 모두 예측하고 파악하기 힘들기 때문에 T_H의 최악블로킹 시간의 한계를 측정하기는 어렵다.

이와같은 우선순위 반전 문제를 해결하기 위한 방안으로 우선순위 계승 프로토콜이 있다[2, 3, 4]. 우선순위 계승 프로토콜은 한 작업으로부터 요청이 들어와 이전 작업의 요청처리로 인해 큐에 대기될 때, 서버 작업의 우선순위가 요청의 우선순위보다 낮을 경우 서버의 우선순위를 요청의 우선순위로 높여 주는 것이다. 이렇게 함으로써 요청의 우선순위보다 낮은 우선순위가 서버 작업을 선점 하는 경우를 방지할 수 있다. (그림 1)의 아래 그림에서와 같이 T_H가 서버 작업 S에게 요청을 보낸 후 T_L의 요청처리로 인해 큐에 대기될 때 서버 작업 S는 T_H의 우선순위를 계승받게 된다. 이와같은 상황에서는 서버 작업의 우선순위가 T_M의 우선순위보다 높으므로 T_M은 서버 작업의 활동을 선점 할 수 없게 된다.

2.2 실시간 서버 모델

우선순위 반전 문제를 해결하기 위한 방법으로서 우선순위 계승 프로토콜을 살펴보았다. 그러나, 우선순위 계승 프로토콜이 클라이언트 작업들의 요청에 대해서 보다 나은 서버 작업에 대한 선점가능성을 제공하여 최적의 응답시간을 제공할 수 있는 실질적인 구조를 제시하고 있는 것은 아니다. 즉, CPU의 효율성을 높이고, 요청에게 높은 선점가능성을 제공할 수 있는 실시간 서버의 구조가 필요하다.

실시간 서버 모델은 서버 작업이 요청을 처리하기 위해 몇 개의 활동 작업을 갖도록 함으로써 서버 작업에 대한 선점가능성을 향상시키는데 있다. 실시간 서버 모델의 초점은 서버 작업의 구조와 우선순위 관리에 있으며, 이미 다른 요청을 처리 중인 서버 작업은 공유 자원을 다루게 되므로 아무리 높은 우선순위 작업의 요청이 들어온다 할지라도 현재 처리중인 작업의 요청 처리가 완료될 때까지는 대기 큐에서 대기해야 한다. 이와같은 요청을 보다 효과적으로 처리할 수 있는 구조를 제공하는 실시간 서버 모델에는 요청을 처리하기 위한 스래드의 개수와 우선순위의 관리 방법에 따라 다음과 같이 분류된다[3].

첫 번째 서버 모델은 단일 스래드 서버 모델(single thread server model)로서 가장 간단한 구조이고 서버 작업이 하나의 스래드를 갖으며, 한 스래드가 모든 요청을 처리한다. 클라이언트 작업으로부터 들어온 요청은 서버 작업에서 이미 처리되고 있는 다른 작업의 요청 처리를 선점 할 수 없으므로 요청 처리가 완료 될 때까지 블록 되어 대기 큐에 대기해야 한다. 이 때 해당 작업 요청의 우선순위가 서버 작업의 우선순위보다 높을 경우 우선순위 계승 프로토콜에 따라 해당 요청은 자신의 우선순위를 서버 작업의 우선순위에 계승한 후 블록된다. 따라서 우선순위가 높은 작업의 요청은 큐에서 대기 중 다른 낮은 우선순위 작업의 실행으로 인해 서버 작업을 선점당하는 경우를 배제시킬 수 있다. 즉, 낮은 우선순위 작업들의 예측하지 못하는 선점이 없으므로 해당 작업의 최악블로킹 시간을 계산할 수 있는 것이다.

두 번째 서버 모델은 작업자 모델(worker model)로서 단일 스래드 서버 모델과 달리 클라이언트 작업의 요청에 대해서 서버 작업이 이를 처리하기 위해 다중 스래드 구조를 갖는 모델이다. 작업자 모델에는 두 가지 종류가 있는데 우선순위 관리 방법에 따라 정적 우선순위 작업자 모델(static prioritized worker model)과 동적 우선순위 작업자 모델(dynamic prioritized worker model)로 나눌 수 있다.

정적 우선순위 작업자 모델에서는 서버 작업에 있는 각 작업자 스래드가 자신이 처리해야 할 우선순위를 미리 정하여 갖고 있다. 따라서 이 모델에서 필요한 작업자 스래드는 시스템에서 제공하는 우선순위 범위만큼 존재해야한다. 만약 해당 작업자 스래드가 다른 클라이언트 작업의 요청을 처리중이라면, 새로운 요청은 이전 요청의 처리가 끝날 때까지 해당 작업자 스래

드의 대기 큐에서 대기해야 한다. 또한, 서버는 현재 자신의 작업자 스래드들이 처리하고 있는 요청들의 우선순위 중 가장 높은 우선순위를 갖게 된다. 이 모델은 단일 스래드 서버 모델보다 높은 선점성을 제공하고 있다.

동적 우선순위 작업자 모델에서는 작업자 스래드의 개수가 우선순위 단계와 같을 필요가 없이 시스템의 성능을 저하시키지 않을 정도의 개수로 존재할 뿐 아니라 작업자 스래드들 모두 우선순위에 관계없이 클라이언트 작업의 요청을 처리한다. 만약 모든 작업자 스래드들이 클라이언트 작업의 요청을 처리중이라면, 새로운 클라이언트 작업 요청은 작업자 스래드들 중 어느 것 하나가 작업을 마칠 때까지 블록 된 채 대기 큐에서 기다려야 한다. 이 때 새로운 요청은 서버 작업의 우선순위와 자신의 우선순위를 비교하여 서버 작업의 우선순위가 자신의 것보다 낮을 경우 대기 큐에 들어가기 전에 서버 작업에 자신의 우선순위를 계승하게 된다. 이렇게 함으로써 서버 작업은 자신이 처리하고 있는 모든 클라이언트 작업의 요청들 중 가장 높은 우선순위를 갖는 요청의 우선순위를 계승받아 처리 작업을 계속할 수 있다.

세 번째 서버 모델은 동적 서버 모델(dynamic server model)은 클라이언트 작업으로부터 새로운 요청이 들어왔을 때, 해당 요청을 처리할 스래드가 없을 경우 요청 처리를 위해 스래드 관리자를 통해 새로운 스래드를 생성시킨다. 스래드 관리자는 처리할 요청이 들어왔을 때 요청을 처리할 스래드가 있을 경우에는 해당 스래드를 할당하고, 그렇지 않을 경우에는 새로운 스래드를 생성하는 역할을 담당한다. 여기서 새로운 클라이언트 작업 요청의 우선순위가 서버 작업의 우선순위보다 높을 경우 서버 작업의 우선순위는 요청의 우선순위를 계승받게 된다. 이 모델은 다른 모델에서처럼 클라이언트 작업으로부터의 요청이 이전 클라이언트 작업 요청 처리의 완료를 기다릴 필요가 없으므로 서버 작업에 대한 가장 높은 선점가능성을 갖는다.

앞에서 살펴 본 실시간 서버 모델에서 단일 스래드 서버 모델은 스래드의 개수가 하나밖에 없는 관계로 서버 작업에 대한 클라이언트 요청의 개수가 많을 경우 우선순위 반전 문제가 발생되지 않음에도 불구하고 요청에 대한 응답시간이 길어지는 문제가 발생한다. 또한, 서버 작업이 짧은 서비스만을 제공할 경우에는 적합하지만 그렇지 못한 경우에는 서버 작업에 대한 낮은 선점가능성과 우선순위 반전 시간이 길어지게 된다.

정적 우선순위 작업자 모델은 높은 우선순위의 요청이 낮은 우선순위 처리를 위해 대기되는 문제는 발생하지 않는다. 그러나 우선순위 단계가 많아질 경우 작업자 스래드의 개수가 함께 많아짐으로 인해 문맥교환(context switching)이 빈번하게 이루어 질 수 있으며, 이는 CPU 효율성(utilization)의 심각한 저하를 초래할 수 있다. 또한, 작업자 스래드의 개수가 많음에도 불구하고 같은 우선순위의 클라이언트 작업들이 계속해서 요청을 할 때 다른 작업자 스래드들은 요청을 처리하고 있지 않더라도 일정 작업자 스래드의 요청 처리 완료를 모두 기다려야 하는 문제가 발생한다.

반면에 동적 우선순위 작업자 모델은 시스템 성능을 저하시키지 않는 범위에서 스래드를 최적의 개수로 정할 수 있지만 모든 스래드들이 낮은 우선순위의 요청을 처리하고 있을 경우, 높은 우선순위의 요청이 낮은 우선순위의 요청처리를 기다릴 수밖에 없는 문제가 발생한다.

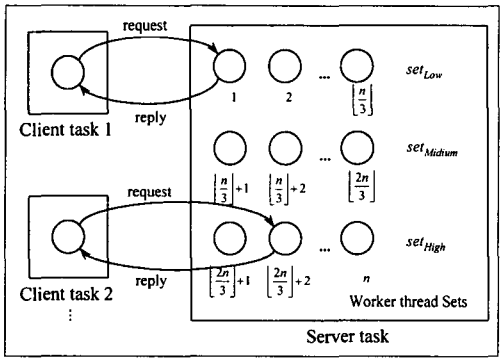
마자막으로 동적 서버 모델의 경우는 가장 높은 선점가능성을 가지고 있으며 우선순위 반전 문제가 일어나지 않지만 요청에 대한 새로운 스래드의 생성으로 인한 비용을 무시할 수는 없다. 또한 정적 우선순위 작업자 모델과 마찬가지로 계속적으로 스래드의 개수가 늘어나게 되면 빈번하게 일어나는 문맥교환의 문제점을 갖고 있을 뿐 아니라, 스래드가 동적으로 생성되어지기 때문에 서버 작업상에서 자원 활용율을 최대한으로 높이기란 상당히 어렵다.

위의 모델들은 우선순위 반전 시간을 줄이고 서버에 대한 선점가능성을 제공하기 위한 구조를 제공하나, 선점가능성와 오버헤드간의 트레이드 오프(trade-off)를 해결하지 못하였다. 따라서, 우선순위 반전 문제를 해결하여 높은 선점가능성을 갖고 동시에 전체 시스템 성능에 영향을 최대한 줄일 수 있는 새로운 모델이 필요하다.

3. 하이브리드 우선순위 작업자 모델

본 논문에서는 선점가능성과 오버헤드간의 트레이드 오프를 최적화 한 모델로서, 정적 우선순위 작업자 모델과 동적 우선순위 작업자 모델의 특성을 결합한 혼합형 우선순위 작업자 모델을 제안한다(그림 2). 혼합형 우선순위 작업자 모델에서는 작업자 스래드 n개를

작업자 스래드 집합으로 나눈다. 작업자 스래드 집합은 높은 우선순위의 클라이언트 작업 요청을 처리하기 위한 작업자 스래드 집합과 중간 우선순위와 낮은 우선순위의 클라이언트 작업 요청을 처리하기 위한 각각의 작업자 스래드 집합으로 구성된다. 각 작업자 스래드 집합은 해당 우선순위 범위에 속하는 클라이언트 작업의 요청만을 처리하게 된다. 만약 클라이언트 작업으로부터 새로운 요청이 들어오면, 해당 요청의 우선순위가 3개의 범위 중 어느 범위에 속하는 지에 따라서 해당 범위의 요청을 처리하는 작업자 스래드 집합으로부터 서비스를 받게 된다. 작업자 스래드 집합 내에서의 요청처리는 동적 우선순위 작업자 모델과 같은 방법으로 클라이언트 작업의 요청을 처리할 유효한 작업자 스래드가 요청을 처리하게 된다. 이 때 작업자 스래드 집합 내에 모든 작업자 스래드들이 다른 클라이언트 작업의 요청을 처리중이라면, 새로운 요청은 서버 작업과 자신의 우선순위를 비교하여 서버 작업의 우선순위가 자신의 것보다 낮을 경우 자신의 우선순위를 서버 작업에게 계승시킨 후에 해당 작업자 스래드 집합의 대기 큐에 대기하여 블록된다.



(그림 2) 혼합형 우선순위 작업자 모델

(그림 2)에서 클라이언트 작업의 요청이 set_{High}의 작업자 스래드 집합이 처리하는 우선순위를 가질 경우 set_{High}에서 다른 클라이언트 작업의 요청을 처리하고 있지 않은 스래드가 요청을 처리하게 된다. 그러나 해당 작업자 스래드 집합내의 스래드들이 모두 이전 클라이언트 작업의 요청을 처리 중이고, 서버 작업의 우선순위가 요청의 우선순위보다 낮을 경우, 해당 요청은 자신의 우선순위를 서버 작업에 계승시킨 후 큐에 대기된다. 여기서 전체 스래드 개수 n은 CPU 효율성을 저하시키지 않는 범위에서 클라이언트 작업 요청에 대한 응답 시간을 최대한 줄일 수 있는 개수로 결정된다.

혼합형 우선순위 작업자 모델은 낮은 우선순위의 클라이언트 작업 요청들이 모든 스래드를 선점 하는 경우를 방지할 수 있으며, 제일 높은 우선순위의 클라이언트 작업 요청이 제일 낮은 우선순위의 클라이언트 작업 요청 처리를 기다리게 되는 경우도 배제할 수 있다. 따라서 새로운 클라이언트 작업 요청에게 보다 높은 선점가능성을 제공하고 우선순위 반전 시간을 감소하여 요청에 대한 응답 시간을 줄일 수 있다. 또한 정적 우선순위 작업자 모델에서 우선순위 단계 수만큼의 작업자 스래드를 두어 빈번한 문맥교환이 일어나는 것과 다른 작업의 요청을 처리하고 있지 않음에도 불구하고 다른 우선순위의 요청을 처리 할 수 없는 것과는 달리 스래드 집합이 자신이 처리할 클라이언트 작업의 우선순위 범위를 갖고 작업자 스래드의 개수를 시스템의 성능을 저하시키지 않는 범위에서 정하므로 특정 범위의 우선순위를 갖는 클라이언트들이 요청을 해 올 때 해당 작업자 스래드에만 부하가 높고 다른 작업자 스래드들은 요청을 처리하지 않는 상태가 줄어들며, 문맥교환으로 인한 시스템의 오버헤드를 감소할 수 있다.

또한, 이 모델에서는 높은 우선순위의 클라이언트 작업 요청이 낮은 우선순위의 클라이언트 작업 요청에 의해서 블록 되는 경우가 현저히 불어들게 되므로, 우선순위 계승 프로토콜로도 완전히 해결하지 못한 우선순위 반전 현상을 거의 없앨 수가 있다.

4. 성능 평가

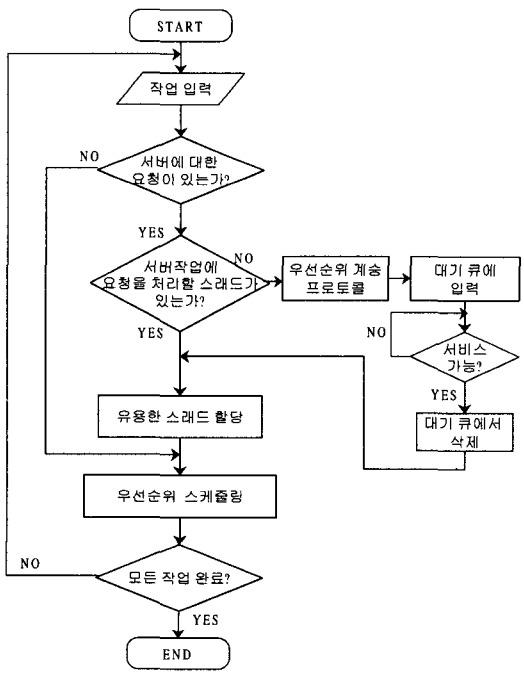
4.1 시뮬레이션

본 논문에서는 혼합형 우선순위 작업자 모델의 성능을 평가하기 위해 기존 실시간 서버 모델인 단일 스래드 서버 모델, 정적 우선순위 작업자 모델, 정적 우선순위 작업자 모델을 비교 대상으로 하였다. 단일 스래드 서버 모델의 경우 가장 기본적인 형태를 갖는 모델로서 서버 작업이 하나의 스래드만으로 클라이언트 작업의 요청의 처리하는 경우와 스래드를 여러 개 두는 멀티스래드 구조를 가지고 요청을 처리하는 경우를 비교하기 위하여 비교평가 모델로 채택하였으며, 정적 우선순위 작업자 모델과 동적 우순 순위 작업자 모델은 혼합형 우선순위 작업자 모델에 직접적인 영향을 끼친 모델로서 작업자 스래드가 처리할 클라이언트 작

업 요청의 우선순위가 정적으로 정해졌을 경우와 그렇지 않을 경우, 그리고 스래드의 개수를 어떻게 정할지에 대한 비교 대상이 된다.

본 논문에서 혼합형 우선순위 작업자 모델과 기존이 실시간 서버 모델들과의 성능을 평가하는데 있어서 요소로서 채택한 것은 특정 클라이언트 작업이 서버 작업에게 요청을 보내어 서버 작업으로부터 응답을 받을 때까지의 응답시간을 측정하였으며, 성능을 측정하기 위해 기반이 되는 환경은 대표적인 OS인 RT-Mach의 특징을 바탕으로 하였다. 응답시간을 측정하는데 있어서는 해당 클라이언트 작업 외에 주변에서 실행되는 작업들 즉, 같은 서버 작업에 요청을 해오는 클라이언트 작업들이나 서버 작업과는 무관하게 실행되는 비서버 작업들의 갯수를 조정하면서 평가해 보았다.

시뮬레이션 모델에서는 새로운 작업들이 들어오면 우선 해당 작업이 서버 작업에 대해 요청 처리를 요구하는 클라이언트 작업인지 아니면 서버 작업과는 상관없이 실행되는 비서버 작업인지를 파악한다. 만약 비서버 작업일 경우에는 곧바로 CPU의 프로세서 스케쥴링 큐에 들어가 스케줄 된다. 그러나 클라이언트 작업일 경우에는 서버 작업내의 스래드가 다른 작업의 요청 처리를 하고 있는 지를 검사하여 만약 곧바로 요청 처리를 받을 수 있다면 서버로부터 요청을 처리 받고, 그렇지 않을 경우에는 서버의 대기 큐에서 서버로부터 요청을 받을 때까지 대기하게 된다. 이때, 클라이언트 작업의 요청은 자신의 우선순위와 서버 작업의 우선순위를 비교하여 자신의 우선순위가 서버 작업의 우선순위보다 높으면 자신의 우선순위를 서버 작업에게 계승시킨 후에 블록 된다. 일단 서버로부터 모든 요청을 다 처리 받은 클라이언트 작업은 다시 프로세서를 할당받기 위하여 큐에서 스케줄 된다(그림 3).



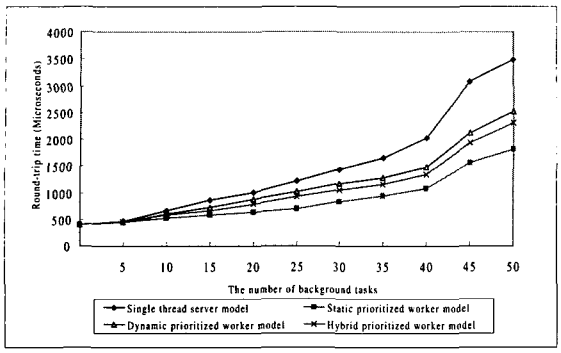
(그림 3) 시뮬레이션 흐름도

각 모델의 구조를 시뮬레이션 하여 비교 평가하기 위해서 SIMAN이라는 시뮬레이션 도구를 사용하였으며, 전체 시스템의 우선순위 단계(level)는 RT-Mach의 우선순위 단계인 1부터 32까지의 수치로 잡았으며, 높은 수치가 높은 우선순위를 갖는다[6]. 시스템에서 서버 작업에 요청을 하거나 서버 작업과 상관없이 실행되는 모든 작업들(background tasks)의 개수는 5개에서부터 50개까지 5개씩 변경하면서 평가하였다.

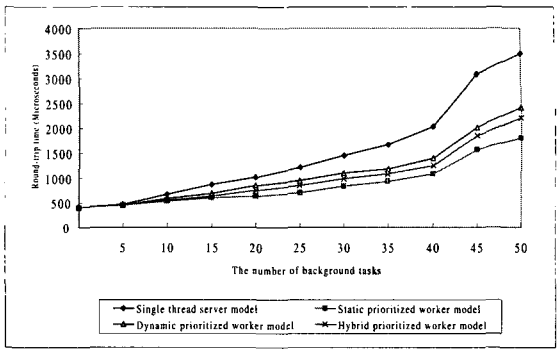
4.2 파라미터 설정

혼합형 우선순위 작업자 모델의 성능을 평가하기에 앞서 우선 정해져야 할 것은 전체 작업자 스래드의 개수이다. 이 개수는 동적 우선순위 작업자 모델에도 같게 적용된다. 작업자 스래드의 개수는 너무 적을 경우 클라이언트 작업의 요청을 효율적으로 처리할 수가 없어 단일 스래드 서버 모델과 별다른 차이점을 갖기 힘들다. 그러나 작업자 스래드의 개수가 너무 많을 경우에는 정적 우선순위 작업자 모델에서 문제점으로 지적되었던 것처럼 많은 문맥교환으로 인한 시스템 전체 성능의 저하를 가져 올 수 있다. 따라서 가장 효율적인 작업자 스래드의 개수를 찾는 것이 중요하다고 하겠다.

본 논문에서는 작업자 스래드의 개념을 도입한 여러 시스템을 살펴본 결과 전체 시스템의 성능에 영향을 주지 않는 작업자 스래드의 개수를 8에서 12개 정도로 잡고 있으며, 보통의 시스템들도 20개 이하의 스래드를 이용하고 있음을 알 수 있었다[7, 8, 9]. 따라서 본 논문에서는 동적 우선순위 작업자 모델의 작업자 스래드의 개수를 9개에서 12개 사이로 변경시키면서 측정하였으며, 혼합형 우선순위 작업자 모델의 작업자 스래드의 전체 개수 또한 9개에서 12개로 변화를 주어 평가해 보았다. 이와 같은 설정을 가지고 여러 가지 성능을 비교 평가하였다. 또한, 각 모델의 성능을 비교



(a) 스래드 개수가 9개일 경우



(b) 스래드 개수가 12개일 경우

(그림 4) 백그라운드 작업들의 우선순위가 균일하게 들어올 경우

하기 위해서는 다음과 같은 식을 이용하였다.

$$\frac{(t_x - t_{hybrid})}{t_x} \times 100(\%)$$

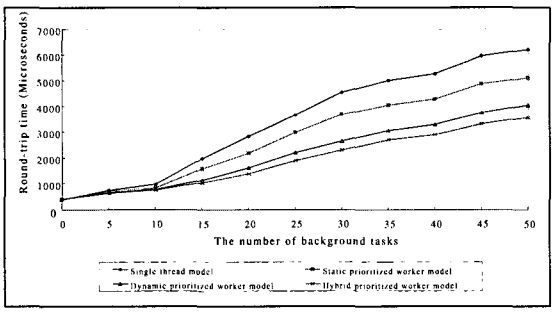
t_x는 혼합형 우선순위 작업자 모델과 비교되는 모델에서의 응답시간이며, t_{hybrid}는 혼합형 우선순위 작업자 모델을 이용할 경우의 응답시간이 된다. 본 논문에서는 위의 수식을 백그라운드 작업이 변화함에 따라 달라지는 응답시간 각각에 대하여 적용하여 그 평균값으로 성능을 비교하였다.

4.3 혼합형 우선순위 작업자 모델의 성능 평가

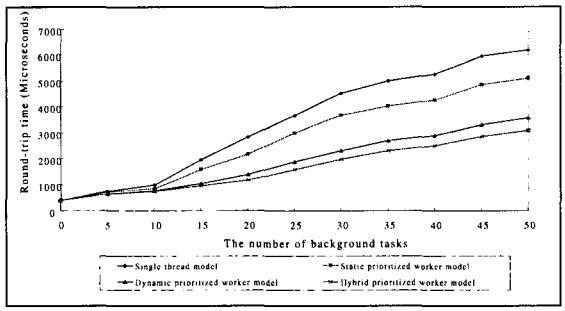
단일 스래드 서버 모델의 경우 서버 작업에서 클라이언트 작업의 요청을 처리하기 위한 스래드는 한 개이다. 정적 우선순위 작업자 모델은 시스템의 우선순위 단계가 1에서부터 32까지이므로 총 32개의 작업자 스래드를 두어 각각이 자신의 순서에 따라 해당 우선순위의 클라이언트 작업 요청을 처리하도록 하였으며, 각각의 작업자 스래드가 자신의 요청 처리를 위해 클라이언트 작업 요청을 대기시킬 대기 큐를 갖고 있다. 동적 우선순위 작업자 모델은 많은 문맥 교환으로 시스템의 성능을 저하시키지 않는 효율적인 스래드 개수로서 9개에서 12개의 작업자 스래드를 갖도록 하여 측정하였다. 작업자 스래드로부터 요청을 바로 받지 못하는 클라이언트 작업의 요청이 대거하기 위한 대기 큐는 하나이다. 마지막으로 혼합형 우선순위 작업자 모델은 동적 우선순위 작업자 모델과 같은 개수의 작업자 스래드를 갖고 있으며, 작업자 스래드 집합으로는 set_{Low}, set_{Midium}, set_{High}로 나누어 각각이 작업자 스래드를 균일하게 포함한다. 즉, set_{Low}, set_{Midium}, set_{High} 각각이 1부터 10, 11부터 21 그리고 22부터 32까지의 우선순위를 갖는 클라이언트 작업의 요청을 처리하며, 이들 각각의 집합은 다른 클라이언트 작업들의 요청 처리로 인해 클라이언트 작업이 블록 되기 위한 대기 큐를 하나 갖는다.

4가지 모델의 성능을 비교하기 위해서 서버 작업으로부터 요청을 처리 받아 실행되는 클라이언트 작업들과 서버 작업과 관계없이 실행되는 비서버 작업들을 포함하는 백그라운드 작업(background tasks)들의 개수를 5개씩 늘려가면서 이에 따라 임의의 우선순위를 갖는 특정 클라이언트 작업의 요청에 대한 응답시간의 변화를 측정하였다.

(그림 4)에서는 백그라운드 작업들의 우선순위를 1부터 32까지의 범위를 갖는 균일 분포 상태에서 실험을 하였다. 먼저 (a)는 동적 우선순위 작업자 모델과 혼합형 우선순위 작업자 모델의 작업자 스래드 개수를 9개로 하였으며, 서버 작업으로 요청을 해오는 클라이언트 작업의 우선순위가 균일하게 들어올 경우에는 정적 우선순위 작업자 모델이 가장 좋은 응답시간을 갖음을 알 수 있다. 이는 정적 우선순위 작업자 모델이 가장 많은 작업자 스래드를 가지고 있으며, 서버 작업에 요청을 해 오는 클라이언트 작업들의 우선순위가 균일하게 들어오기 때문에 특정 작업자 스래드의 대기 큐에 많은 클라이언트 작업요청이쌓이게 되는 경우가 없기 때문이다. 따라서, 정적 우선순위 작업자 모델이 작업자 스래드를 32개를 갖음으로 인해 발생하는 많은 문맥교환으로 시스템의 성능이 저하되는 것을 고려하여야 한다.



(a) 스래드 개수가 9개일 경우



(b) 스래드 개수가 12개일 경우

(그림 5) 백그라운드 작업들의 우선순위가 지수분포를 기반으로 할 경우

작업자 스래드를 9개 갖는 혼합형 우선순위 작업자 모델은 단일 스래드 서버 모델보다 응답 시간이 평균 22.5% 향상되었으며, 동적 우선순위 작업자 모델보다는 평균 7.2% 향상되었다. 정적 우선순위 작업자 모델에 비해서는 평균 18.63% 성능의 차이를 보였으며, 작업자 스래드를 12개 갖는 경우에는(b) 단일 스래드 서버 모델보다 응답 시간이 평균 25.8% 향상되었으며, 동적 우선순위 작업자 모델보다는 평균 7.23% 향상되었다.

(그림 5)는 클라이언트 작업들의 우선순위를 평균 12인 지수분포를 역으로 한[1] 분포도를 근거로 하여 실험한 결과 그래프이다. (그림 4)에서 클라이언트 작업의 우선순위가 균일하게 들어온 것과 달리 우선순위의 분포가 20에서 32사이에 집중되므로 임의의 우선순위를 갖는 클라이언트 작업 요청에 대한 응답시간이 훨씬 길어지고 있다. 또한, 정적 우선순위의 작업자 모델의 경우 클라이언트 작업 요청의 우선순위가 균일하게 들어오지 않고, 특정 범위의 우선순위를 갖는 요청들이 들어오게 되므로 동적 우선순위 작업자 모델보다 긴 응답시간을 갖는다. 이는 특정 범위의 클라이언트 작업의 요청이 계속적으로 들어옴에 따라 이에 해당되는 작업자 스래드의 대기 큐에 많은 요청들이 대기되기 때문이다.

(a)의 경우 정적 우선순위 작업자 모델이 백그라운드 작업들의 우선순위 특성에 따라 클라이언트 작업 요청 처리 시간이 크게 변화되는 것과는 달리 혼합형 우선순위 작업자 모델은 작업자 스래드의 개수가 9개일 경우 단일 스래드 서버 모델보다는 평균 36.98%, 정적 우선순위 작업자 모델보다는 평균 26.3%의 성능 향상을 보이고 있다. 또한, 동적 우선순위 작업자 모델과는 평균 9.2%의 성능 차이를 보인다(그림 5). 따라서, 백그라운드 작업들의 우선순위가 균일하지 않고, 일정 범위의 우선순위를 갖는 백그라운드 작업들의 개수가 늘어남에 따라 혼합형 우선순위 작업자 모델은 더욱 높은 성능 향상을 가져오고 있음을 알 수 있다.

(b)의 경우 동적 우선순위 작업자 모델과 혼합형 우선순위 작업자 모델의 작업자 스래드의 개수를 12개로 하여 성능을 측정하였을 경우에는 혼합형 작업자 모델이 단일 스래드 서버 모델보다는 평균 42.45%, 정적 우선순위 작업자 모델보다는 평균 33.07%, 그리고 동적 우선순위 작업자 모델보다는 평균 10.18%의 성능 향상을 보인다.

5. 결 론

실시간 시스템은 어플리케이션의 실시간적 행위를 분석하고 예측할 수 있으며, 우선순위를 고려한 스케줄링과 우선순위 제어와 같은 실질적인 자원 관리를 위한 기법도 제공할 수 있어야 한다. 또한, 각 작업들의 동기화로 인한 블록킹에서 심각한 시간적 문제를 발생시킬 수 있는 우선순위 반전 문제를 해결하기 위해서는 우선순위 계승 프로토콜을 이용해야만 한다.

본 논문은 우선순위 계승 프로토콜을 이용하여 서버 작업에 대한 보다 나은 선점가능성과 빠른 응답시간을 제공할 수 있는 실질적인 구조를 제안하는 혼합형 우선순위 작업자 모델을 제안하였다. 이 모델은 정적 우선순위 작업자 모델과 동적 우선순위 작업자 모델의

1) 지수분포는 평균보다 낮은 값의 수치가 많은 형태의 그래프를 띄게 되므로, 높은 우선순위를 갖는 백그라운드 작업들의 개수가 더욱 많게 하기 위해서는 지수 분포를 이용하여 나온 데이터 값을 32에서 빼주어 역 그래프를 만들어야 한다.

각각의 장점을 살린 모델로서 오버헤드를 줄이고 높은 선점가능성을 갖기 위해 설계된 것이다.

본 논문에서 제안한 혼합형 우선순위 작업자 모델을 단일 스래드 서버 모델, 정적 우선순위 작업자 모델, 동적 우선순위 작업자 모델과 비교하여 실험한 결과 백그라운드 작업의 우선순위 분포가 균일할 경우 임의의 우선순위를 갖는 클라이언트 작업에 대한 응답시간이 단일 스래드 서버 모델보다는 평균 25.8%, 동적 우선순위 작업자 모델보다는 평균 7.23%, 정적 우선순위 작업자 모델보다는 -12.97%의 차이를 보임을 알 수 있었고, 백그라운드 작업의 우선순위 분포를 지수 분포를 기반으로 하였을 때에는 단일 스래드 서버 모델, 정적 그리고 동적 우선순위 작업자 모델 각각에 대해서 평균 42.45%, 33.07%, 10.18%의 성능향상을 가져왔다.

이와 같이 본 논문에서는 우선순위 계승 프로토콜과 서버 작업에 대한 선점가능성을 높일 수 있는 혼합형 우선순위 작업자 모델을 이용하여 우선순위 반전 문제를 해결하고 클라이언트 작업 요청에 대한 응답시간을 줄일 수 있다.

참 고 문 헌

[1] T. Nakjima, T. Kitayama, H. Tokuda, "Experiments with Real-Time Servers in Real-Time Mach," In Proceedings of 3rd USENIX Mach Symposium, Apr. 1993.

[2] L. Sha, R. Rajkumar and J. P. Lehoczky, "Priority Inhertiance Protocols -An Approach to Real-Time Synchronization," Technical Report CMU-CS-87-181, Carnegie-Mellon Univ., 1987.

[3] C. W. Mercer and H. Tokuda, "An Evaluation of Priority Consistency in Protocol Architecture," In Proceedings of the IEEE 16th Conference on Local Computer Networks, 1991.

[4] H. Tokuda, C. W. Mercer, Y. Ishikawa, T. E. Marchok, "Priority inversion in real-time communication," In Proceedings of 10th IEEE Real-Time Systems Symposium, Dec. 1989.

[5] H. Tokuda, C. W. Mercer, Y. Ishikawa, T. E. Marchok, "Priority inversion in real-time communication," In Proceedings of 10th IEEE Real-Time Systems Symposium, Dec. 1989.

[6] K. Loepere(ed), "Mach 3 Server Writer's Guide," Open Software Foundation and Carnegie Mellon Univ., 1992.

[7] H. Dai and B. Paulsen, "Multithreading VHDL Simulation," the VIUF Fall 1994 Conference, 1994.

[8] D. M. Tullsen, S. J. Eggers, J. S. Emer and H. M. Levy, "Exploiting Choice : Introduction Fetch and Issue on an Implementable Simultaneous Multithreading Processor," Proceedings of the 23rd Annual International Symposium on Computer Architecture, Philadephia, May 1996.

[9] C. Chen, M. K. Johnson, K. Moore and D. Treadwell, "December 1994/Feature : The Threading models," http://www.byte.com/art/9412/sec9/art3.htm.

[10] S. Mitsuhias, "Performance Evaluation and Analysis of Multithreading," http://www.etl.go.jp/etl/compare/EM-X/emx-multithread.htm.

[11] T. Kitayama, T. Nakjima, H. Tokuda, "RT-IPC: An IPC Extension for Real-Time Mach," In Proceedings of the Winter 1992 USENIX Conference, pp.91-104, Sep. 1993.

[12] S.Sommer, "Removing Priority Inversion from an Operation System," In proceedings of the 19th Australasian Computer Science Conference, Melbourne, Australia, Jan.31-Feb.2 1996.

[13] M. I. Chen and K. J. Lin, "Dynamic Priority Ceilings : A concurrency control protocol for real-time systems," Real Time Systems, Vol.2, pp. 325-246, 1990.

[14] M. I. Chen and K. J. Lin, "A Priority Ceiling Protocol for multiple-instance resources," In Real Time Systems Symposium, pp.141-148, 1991.

[15] L. Sha, R. Rajkumar, S. H. Son and C. H. Chang, "A Real-Time Locking Protocol," In Proceedings of IEEE Real-Time Systems, 1991.

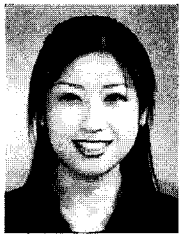
[16] O. Babaoglu, K. Marzullo and F. B. Schneider, "A Formalization of Priority Inversion," Technical Report UBLCS-93-4 of Laboratory for Computer Science, Bologna University, Mar. 1993.

[17] L. C. Shu, M. Young and R. Rjkumar, "An Abort Ceiling Protocol for Controlling Priority Inversion," In Proceeding of 1st Interntional Workshop on Real-Time Computing Systems &

Application, Seoul Korea, pp.48-52, 1994.
[18] K. W. Lam, S. H. Son and S. L. Hung, "A Priority Ceiling Protocol with Dynamic Adjustment of Seralization Order," Dep. of Computer Science City University of Hong Kong, 1997.
[19] K. W. Lam, Victor C. S. Lee, S. L. Hung and K. Y. Lam, "An Augmented Priority Ceiling Protocol for Hard Real-Time Database Systems," In Journal of Computing and Information, Vol.2, No.1, pp.849-866, 1996.
[20] T. P. Baker, "Stack-Based Scheduling of Real-time Processes," In Journal of Real-Time Systems, Vol.3, No.1, pp.67-99, 1991.

박 홍 진

e-mail : hjpark@sslab.cse.cau.ac.kr
1993년 원광대학교 컴퓨터공학과 졸업(공학사)
1995년 중앙대학교 대학원 컴퓨터공학과(공학석사)
1997년~현재 중앙대학교 대학원 컴퓨터공학과 박사과정
관심분야 : 동기화, 실시간 운영체제, 분산 시스템 등



천 경 아

e-mail : amie@sslab.cse.cau.ac.kr
1996년 중앙대학교 컴퓨터공학과 졸업(학사)
1998년 중앙대학교 대학원 컴퓨터공학과(공학학사)
1999년~현재 중앙대학교 대학원 컴퓨터공학과 박사과정
관심분야 : 동기화, 실시간 운영체제, 분산시스템 등

김 창 민

e-mail : kimcm@hana.sungkyul.ac.kr
1977년 서강대학교 수학과 졸업(학사)
1985년 중앙대학교 대학원 전산과(공학석사)
1991년 중앙대학교 대학원 전산과(공학박사)
1989년~1994년 관동대학교 전산과 조교수
1994년~현재 성결대학교 컴퓨터학부 조교수
관심분야 : 분산 시스템, 동기화, 운영체제 등