# 가상 메모리 시스템의 일시적인 과부하 완화 기법

고 영 웅[†] · 아 재 용[†] · 홍 철 호[†] · 유 혁[††]


## 요 약

가상 메모리 시스템에서 프로세스가 현재 물리 메모리에 없는 페이지에 접근하게 되면 페이지 폴트가 발생하고, 이를 처리하기 위하여 예측할 수 없는 지연을 초래하는 페이지 폴트 핸들러가 동작하게 된다. 따라서 가상 메모리 시스템은 실시간 태스크의 제한 시간 실패율을 높여주는 요인이 될 수 있으므로 실시간 시스템에서 사용하기 어렵다. 뿐만 아니라, 새로운 태스크가 동적으로 유입되는 범용 운영체제 환경에서 가상 메모리 시스템은 태스크의 집중적인 페이지 폴트로 인하여 일시적인 과부하 상태에 빠질 수 있다. 본 논문에서는 집중적으로 발생하는 페이지 폴트에 의한 일시적인 과부하를 효율적으로 처리할 수 있는 RBPFH(Rate-Based Page Fault Handling) 알고리즘을 제시하고 있다 알고리즘의 특징은 현재 시스템의 자원을 모니터링하면서 페이지 폴트가 제한된 범위를 초과하지 않도록 분산시키는 방법을 사용하고 있으며, 페이지 폴트 처리 비율을 조정해줌으로써 동적으로 시스템의 부하가 변하는 상황을 고려하고 있다. RBPFH 알고리즘은 리눅스 운영체제에서 구현하였으며, 실험을 통하여 멀티미디어와 같은 연성 실시간 태스크를 지원하는데 있어서 유용하게 사용될 수 있음을 확인하였다. 실험 결과 RBPFH는 태스크의 제한 시간 실패율을 10%~20% 감소시키고, 평균 지연 시간을 50%~60% 감소시켜주었다.


# Transient Overloads Control Mechanism for Virtual Memory System

Young-Woong Ko[†] · Jae-Yong Ah[†] · Cheol-Ho Hong[†] · Hyuck Yoo[††]


## ABSTRACT

In virtual memory system, when a process attempts to access a page that is not resident in memory, the system generates and handles a page fault that causes unpredictable delay. So virtual memory system is not appropriate for the real-time system, because it can increase the deadline miss ratio of real-time task. In multimedia system, virtual memory system may degrade the QoS(quality of service) of multimedia application. Furthermore, in general-purpose operating system, whenever a new task is dynamically loaded, virtual memory system suffers from extensive page fault that cause transient overloading state. In this paper, we present efficient overloading control mechanism called RBPFH (Rate-Based Page Fault Handling). A significant feature of the RBPFH algorithm is page fault dispersion that keeps page fault ratio from exceeding available bound by monitoring current system resources. Furthermore, whenever the amount of available system resource is changed, the RBPFH algorithm dynamically adjusts the page fault handling rate. The RBPFH algorithm is implemented in the Linux operating system and its performance measured. The results demonstrate RBPFH's superior performance in supporting multimedia applications. Experiment result shows that RBPFH could achieve 10%~20% reduction in deadline miss ratio and 50%~60% reduction in average delay.


**키워드 : 가상 메모리(virtual memory), 멀티미디어(multimedia), 실시간(real-time), 페이지 폴트(page fault)**

## 1. 서 론

다중 프로그램을 지원하는 대부분의 운영체제는 효율적으로 주기억 장치를 관리하기 위한 방법으로 가상 메모리(virtual memory) 개념을 사용하고 있다. 가상 메모리 기법은 사용자 프로그램을 분할하여 필요한 블록들만 비연속적으로 주기억장치에 적재시키는 기법이며, 이 개념을 사용할 경우 아무리 용량이 큰 프로그램도 중첩구조(overlay) 등의 특별한 기법 없이 주기억 장치에 적재되어 실행될 수 있다[6]. 이러한 기법이 가능한 이유는 프로그래머에 의해서 작성되는 프로그램이 메모리에 적재되어 수행이 될 때 지역성(locality)이라는 특성을 가지기 때문이다. 지역성이란 사용자 프로그램이 실행될 때 프로그램 내의 모든 명령들이 균일하게 사용되지 않고, 일부 명령어들이 집중적으로 실행되는 현상을 말하며, 시간 지역성(temporal locality)과 공간 지역성(spatial locality)으로 구분된다. 대부분의 프로그램들이 지역성을 가지고 있으므로 지역성을 유지시켜줄 수 있는 메모리 공간만 충분하면, 여러 개의 프로그램이 동시에 수행될 수 있는 것이다.

가상 메모리 장치를 사용하는 시스템에서 프로그램을 수

※ 본 연구는 한국과학재단의 특정기초 연구과제 연구비 지원(과제번호 97-01-00-09-01-3) 및 한국 전자 통신 연구원의 연구비 지원(과제번호 0701-2001-0036)에 의한 결과임.
† 준 회 원 : 고려대학교 대학원 컴퓨터학과
†† 정 회 원 : 고려대학교 이과대학 컴퓨터학과 교수
논문접수 : 2001년 10월 4일, 심사완료 : 2001년 12월 4일

행할 때, 프로그램 로더(loader)는 디스크에 저장되어 있는 프로그램 이미지 전체를 물리 메모리에 모두 적재하지 않고, 일부 부분만 메모리에 적재하며, 프로그램이 수행되면서 필요할 때마다 새로운 물리 메모리를 할당하는 방법을 사용한다. 이와 같은 방법을 요구 페이징(demanding paging) 기법[6]이라고 하며, 요구 페이징은 프로그램이 수행하면서 참조하는 주소 영역이 물리 메모리에 존재하지 않을 때, 페이지 폴트 핸들러(page fault handler)가 동작하여 필요한 물리 메모리를 할당하고, 필요한 경우 디스크에 있는 프로그램 코드 또는 데이터를 메모리로 가져온 후에 프로그램을 수행하게 된다. 따라서 가상 메모리 시스템에서 메모리 접근 명령을 수행하는 코드는 물리 메모리를 할당받고, 디스크 입출력을 처리하는 작업을 병행할 수 있으므로 예측할 수 없는 지연 시간이 추가될 수 있는 것이다. 이러한 문제점으로 인하여, 가상 메모리 시스템은 엄격한 시간 제약을 요구하는 경성 실시간 시스템에서는 사용할 수 없으며, 시간 제약이 완화된 연성 실시간 시스템에 있어서도 서비스 품질(Quality of Service)에 영향을 주는 요인으로 작용하게 된다.

최근에 컴퓨터 고성능화 및 인터넷의 확대에 따라서 멀티미디어 응용에 대한 수요가 급증하고 있으며, 동영상 플레이어 또는 MP3 플레이어와 같이 멀티미디어 응용 프로그램을 사용하는 것은 일반적인 추세가 되었다. 멀티미디어 응용이 수행되는 일반적인 범용 운영체제 환경은 제한된 집합의 태스크가 수행되는 실시간 시스템과는 다르며, 사용자의 선택에 따라서 새롭게 프로그램이 수행되거나 현재 수행되는 작업이 종료될 수 있는 동적인 시스템이다. 따라서 가상 메모리 시스템을 사용하는 범용 운영체제 환경에서 해결해야 하는 또 다른 주요한 문제점 중의 하나는 새로운 태스크가 동적으로 유입될 때 발생하는 시스템의 일시적인 과부하(transient overload)이다. 가상 메모리를 사용하는 범용 운영체제 환경에서 새로운 태스크가 유입될 때, 프로그램 수행 초기에 프로그램 코드 및 데이터가 집중적으로 적재되는 현상이 발생하며, 짧은 시간 동안 과도하게 페이지 폴트 핸들러가 수행되면서 과부하 상태에 빠지게 된다. 이와 같이 일시적인 과부하가 발생되는 원인은 페이지 폴트를 처리하는 모듈이 가장 우선 순위가 높기 때문이며, 페이지 폴트 핸들러가 수행 되는 동안에는 실시간 특성을 요구하는 태스크들이 제한된 시간 안에 작업을 완료하지 못하는 현상이 발생할 수 있다. 따라서 가상 메모리 시스템을 사용하는 운영체제 있어서, 현재 수행 중에 있는 멀티미디어 응용과 같은 태스크가 제한된 시간 내에 작업이 수행될 수 있도록 하기 위해서는 일시적인 과부하를 처리해주는 메커니즘이 반드시 필요하다.

본 논문에서는 동적으로 태스크가 유입되는 범용 시스템 환경에서 일시적으로 발생하는 페이지 폴트 과부하를 분산시킬 수 있는 비율 기반 페이지 폴트 처리(Rate-Based Page Fault Handling) 메커니즘을 제시하고 있으며, 이를 통하여 멀티미디어 응용이 제한된 시간을 지키면서 수행될 수 있음을 보이고 있다. 본 논문의 구성은 다음과 같다. 2장에서는 관련 연구를 설명하며, 3장에서는 동적 태스크 유입이 가상 메모리 시스템에 미치는 영향을 분석한다. 4장에서는 동적으로 태스크가 유입되면서 발생하는 일시적인 과부하를 분산시키는 비율 기반 페이지 폴트 처리 알고리즘을 설명하며, 5장에서는 본 논문에서 제안하는 실험 결과에 대해서 기술하고, 마지막으로 6장에서 결론을 맺는다.

## 2. 관련 연구

동적인 태스크 유입에 의해서 집중적으로 발생되는 페이지 폴트는 일시적으로 프로세서(processor) 및 메모리(memory) 자원에 대한 과도한 요구를 초래하게 된다. 따라서 멀티미디어 응용과 같은 시간 제약적인 태스크를 지원하기 위해서 프로세서 및 메모리에 대한 효율적인 자원 관리가 필수적이다. 기존의 연구는 프로세스 스케줄링[2-4, 8-12] 및 자원 관리[1]에 대한 연구를 통하여 많은 성과가 있었으며, 대표적인 연구는 다음과 같다.

Mercer[1]의 자원 예약(resource reservation)은 사용자가 각 태스크의 프로세서 용량을 할당할 수 있게 하며, 승인 제어를 통해서 시스템이 허용 가능한 한계를 넘는 태스크의 수행을 제한하고 있다. 따라서 각 태스크는 지정된 프로세서 용량을 보장받고 수행이 되며, 남아있는 프로세서 용량은 일반 태스크가 수행되도록 함으로써 범용 시스템에서 수행되기에 적합한 메커니즘이다. 예약 기법은 특히 과부하 상태에서 유용하며, 태스크가 수행 요청을 할 때 시스템의 허용 가능한 프로세서 용량을 초과하는 경우 승인 제어는 재협상을 통해서 태스크를 수행시키게 된다. 하지만 선입선출(first come first service)에 기반한 승인 제어 메커니즘은 기존에 수행되는 태스크의 프로세서 용량을 보장하는데 우선 순위를 가지게 되므로 새로 유입된 태스크는 중요도와 상관없이 제한된 범위 내에서 프로세서 용량을 할당받는 문제를 가지게 된다. 또한 수행될 태스크가 어느 정도의 프로세서 용량을 요구하는지에 대한 측정 방법에 있어서 어려움을 가지고 있다.

Yavatkar[2]는 멀티미디어 태스크의 수행시간을 예측하기 어렵다는 전제 하에 비율에 기반한 적응적 우선 순위 스케줄링(Rate-based Adjustable Priority Scheduling)을 제안하였다. Mpeg[7]과 같은 스트림(stream)의 경우 각 프레임의 수행 시간이 매우 불규칙하므로 자원 예약에서 프로세서 용량을 측정하는데 어려움이 있으며, 이를 해결하기 위해서 비율에 기반한 적응적 우선 순위 스케줄링 방식은 태스크 시작 시에 주어진 프로세서 용량에 대해서 계속적인 모니터링을 한 후에 응용 프로그램 수준의 QoS(Quality of Service) 관리자가 각 태스크의 프로세서 용량을 줄이거나 늘리는 방식

으로 수행된다.

Goyal[3]이 제시한 계층적 프로세서 스케줄러(Hierarchical CPU Scheduler) 방식은 우선 순위(priority)와 비례 할당(proportional sharing)을 기본 메커니즘으로 사용하며, 각 스케줄링 정책은 비례 할당에 따라서 일정량의 프로세서 용량을 할당 받고 모든 태스크는 우선 순위에 따라서 수행되는 메커니즘이다 이 방식은 태스크마다 일정량의 자원을 예약하지 않고 태스크가 속해있는 스케줄링 정책별로 프로세서 용량을 할당해주고 있다.

Nieh[4]에 의해서 제안된 SMART(Scheduler for Multimedia And Real-Time application)는 Solaris UNIX에서 구현되었으며, 시간 제약적인 응용 프로그램을 지원하며, 시스템의 현재 부하량을 살펴보고 동적인 조정을 통해서 리소스를 할당한다. 스케줄링을 결정하는 요소를 중요도(importance)와 긴급도(urgency)로 분류하고, 중요도를 제공하기 위해서 우선 순위와 WFQ(weighted fair queuing)를 사용하며, 긴급도를 위해서 마감 시간 우선 스케줄링(Earliest Deadline First)을 사용한다. SMART는 실시간 태스크와 일반적인 태스크가 동시에 수행될 수 있도록 잘 고려되어 있으며, 이것은 태스크의 각 특성을 분류하고, 중요도와 긴급도에 따라서 스케줄링을 해주기 때문이다.

언급한, 자원 예약 및 스케줄링 연구는 멀티미디어 응용이 일정한 프로세서 용량을 할당받고 안정적으로 수행될 수 있는 기법들을 보여주고 있다. 하지만 동적으로 태스크가 유입될 때 발생되는 일시적인 과부하를 해결하는 부분에 있어서는 적절한 해답을 제공해주지 못하고 있다. 기존의 연구 내용은 지속적으로 과부하가 발생했을 때 재협상을 통해서 멀티미디어 응용 프로그램의 QoS(Quality of Service)를 낮추거나, 또는 일반 태스크가 사용중인 프로세서 용량을 일시적으로 멀티미디어 응용 프로그램이 사용함으로 과부하를 해결하는 방법[1]을 거론하고 있다. 그러나 범용 시스템에서 태스크가 동적으로 유입되면서 발생되는 과부하는 우선 순위가 높은 커널 수준의 페이지 폴트 처리 작업 및 디스크 입출력 작업에 의해서 발생되는 일시적인 현상이므로 지속적인 과부하 처리 기법은 오히려 과도한 오버헤드를 초래하게 된다.

지금까지 언급한 자원 예약 및 스케줄링 연구외에도, 페이지 폴트 처리는 결과적으로 물리 메모리를 필요로 하며, 이러한 요구를 처리하기 위해서 실시간적인 메모리 관리에 대한 연구[13-15]가 진행되었다. 실시간 메모리 관리에 대한 연구는 메모리 할당이 실시간적으로 이루어질 수 있도록 제한된 자원을 관리하는 기법이다. 경성 실시간 시스템에 있어서는 동적인 메모리 관리를 배제하고 있으며, 연성 실시간 시스템에서는 동적 메모리 할당에 관한 연구[13-15]를 통하여 메모리 사용 효율을 높이고 예측 가능한 메모리 할당이 이루어질 수 있도록 하고 있다.

## 3. 가상 메모리 시스템에서 동적인 태스크 유입의 특성 분석

이번 장에서는 태스크가 동적으로 유입되는 단계에서 발생되는 일시적인 과부하 현상을 살펴보기 위해서 범용 운영체제 상에서 사용되는 주요한 응용 프로그램들을 수행시키면서 가상 메모리 시스템의 특성을 파악하였다. 또한 멀티미디어 응용 프로그램이 수행되는 상황에서 동적으로 태스크를 유입시킬 때 발생되는 문제점을 고찰하였다.

가상 메모리 시스템의 특성을 살피기 위해서 사용한 시스템은 Linux[5] 커널 버전 2.4, Intel Pentium 500Mhz, 물리 메모리 64Mbyte이며, 응용 프로그램은 크게 두 가지로 나누어 사용하였다. 첫 번째는 프로그램의 크기가 크고, 메모리 자원 소비가 많은 GUI(Graphic User Interface) 관련 프로그램이며, 두 번째는 프로그램의 크기가 작고 일반적으로 콘솔(console) 환경에서 사용되는 프로그램으로 분류하여 실험을 하였다. <표 1>은 실험에 사용된 응용 프로그램을 보여주고 있으며 프로그램의 코드 및 데이터의 크기를 바이트(byte) 단위로 표시하고 있다. 여기서 text, data, bss 및 total은 유닉스 명령어 file을 수행시켜서 얻은 결과이며 프로그램의 크기를 나타내고 있다. 하지만, <표 1>이 보여주는 정보는 디스크 상에 저장된 이미지의 크기이며, 런타임(run time)에 어느 정도의 메모리를 차지하게 되는지에 대한 정확한 예측은 할 수 없다.

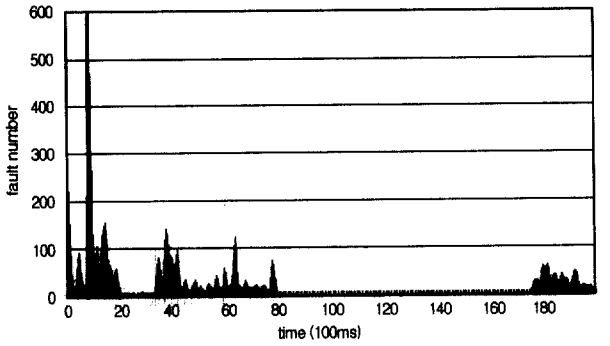
〈표 1〉 실험에 사용된 프로그램 특성

| | gimp | netscape | gzip | find | mpeg_play |
|---|---|---|---|---|---|
| text | 1683884 | 11616067 | 46829 | 72148 | 138363 |
| data | 138100 | 2013332 | 3072 | 592 | 8120 |
| bss | 347044 | 317412 | 329264 | 604 | 941636 |
| total | 2169028 | 13946811 | 379165 | 73344 | 1088119 |

### 3.1 동적 태스크 유입에 따른 페이지 폴트 및 메모리 사용량

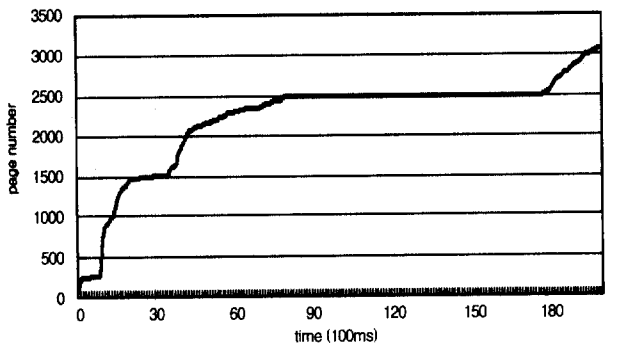
일반적으로 프로그램의 크기가 큰 응용 프로그램이 동적으로 유입될 때 가상 메모리 시스템은 프로그램의 코드 및 데이터를 메모리로 적재하기 위해서 지속적으로 페이지 폴트를 발생시킨다. 이 때 프로그램의 크기가 큰 대부분의 GUI 프로그램은 X 윈도우 시스템을 이용하고 있고, X 윈도우 시스템에서 필요로 하는 자원을 로딩하기 위한 프로그램 모듈들이 연속적으로 수행되면서 프로그램 적재 초반에 집중적으로 페이지 폴트가 발생된다. X 윈도우 시스템에서 사용하는 자원은 창(window), 폰트(font), 색상(color), 그래픽 컨텍스트(Graphic Context) 등이 있다. (그림 1)은 범용 운영체제에서 많이 사용되는 Netscape 웹브라우저의 시간에 따른 페이지 폴트 횟수를 보이고 있다. (그림 1)에서 알 수 있듯이 프로그램 적재가 시작된 이후 8초 이내에 프로그램 수행에

필요한 대부분의 프로그램 코드와 데이터에 대한 페이지 폴트가 집중적으로 발생했음을 알 수 있다.



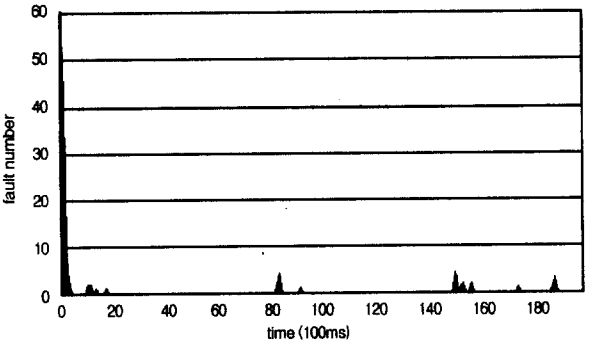

(그림 1) 시간에 따른 Netscape의 페이지 폴트 발생 횟수

(그림 2)는 Netscape 프로그램에서 페이지 폴트가 발생하면서 실제 물리 메모리를 얼마나 할당받았는지 그래프로 보여주고 있다. 그림에서 알 수 있듯이 프로그램이 수행된 초반 8초 이전까지 대략적으로 2500개의 물리 프레임이 집중적으로 요구되었음을 알 수 있으며, 8초 이후로는 새롭게 발생되는 페이지 폴트가 거의 없으므로 메모리 사용량에 변화가 없음을 알 수 있다.

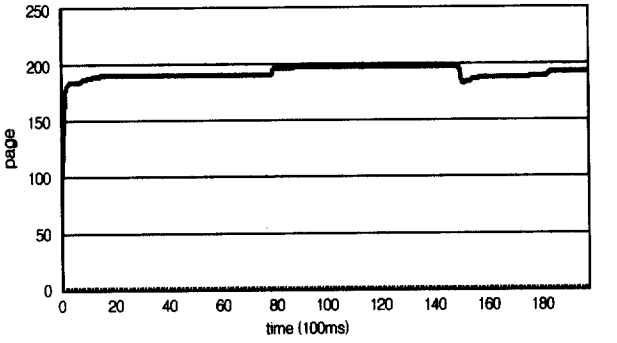



(그림 2) 시간에 따른 Netscape의 메모리 사용량

다음은 메모리 자원에 대한 요구가 작은 응용 프로그램들이 동적으로 유입될 때 가상 메모리 시스템에 어떠한 형태로 부하를 발생시키는지 살펴보겠다. 여기에 속하는 응용 프로그램은 대부분 GUI를 사용하지 않는 프로그램들과 GUI 프로그램 중에서도 프로그램의 크기가 작으며, X 윈도우 시스템의 자원을 거의 사용하지 않는 프로그램들이 포함된다. 대부분의 응용 프로그램들이 비슷한 결과를 보여주었으며, 이 중에서 (그림 3)은 find 프로그램의 수행 결과를 보이고 있다. find도 Netscape와 마찬가지로 프로그램이 적재되자마자 대부분의 코드와 데이터에 대해서 페이지 폴트가 발생되었으며 이후에는 거의 페이지 폴트가 발생되지 않는 특성을 보이고 있다. (그림 4)는 find 프로그램이 시간에 따라서 물리 메모리를 점유하고 있는 상황을 보이고 있으며, 프로그램이 적재되는 초기에 급격히 메모리를 요구하고 있음을 알 수 있다.

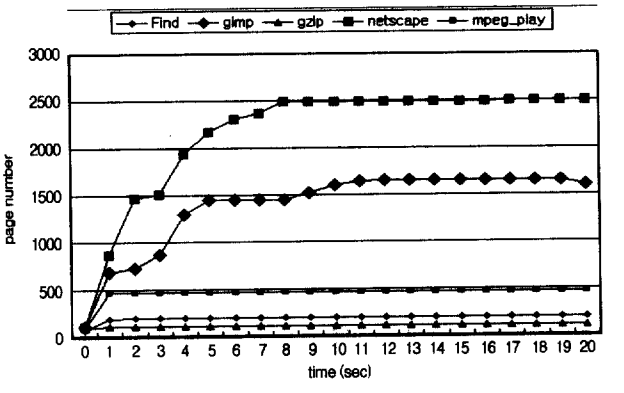



(그림 3) 시간에 따른 find의 페이지 폴트 횟수




(그림 4) find의 시간에 따른 메모리 사용량

앞에서 살펴본 Netscape와 find의 예에서 알 수 있는 것처럼 대부분의 응용 프로그램은 수행되자마자 짧은 시간 동안 수많은 페이지 폴트가 발생되는 것을 확인할 수 있다. 아래 (그림 5)는 Linux에서 수행시킨 여러 가지 응용 프로그램들이 수행 시간에 따라서 물리 메모리를 어느 정도 사용하는지 보여주고 있다. 프로그램의 크기가 작은 find, gzip 그리고 mpeg_play(Berkeley MPEG-1) 등은 수백 밀리 세컨드 내에 프로그램이 필요로 하는 대부분의 메모리 요구를 완료하고 있고, 상대적으로 프로그램의 크기가 큰 Netscape와 gimp는 수초 정도의 시간이 소요되는 것을 알 수 있다. 즉 대부분의



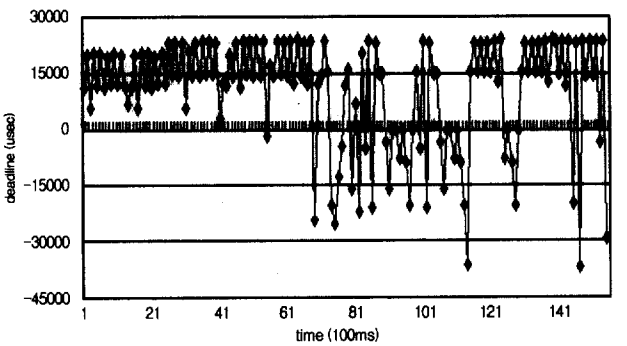

(그림 5) 범용 태스크의 메모리 사용량

응용 프로그램은 프로그램의 시작 초기에 필요한 메모리를 대부분 할당받고 이후에는 물리 메모리에 대한 요구가 거의 없음을 보여주고 있다.

### 3.2 동적 태스크 유입에 의한 멀티미디어 응용의 제한 시간 실패 현상

멀티미디어 응용은 엄격한 제한시간을 요구하지 않으며, 일부 수행이 지연되더라도 시스템에 치명적인 오류를 발생시키지 않는 특성을 가지고 있다. 따라서 사용자가 허용할 수 있는 기대치만큼의 통계적인 수행이 보장될 수 있어야 하며, 제한된 시간을 만족하면서 작업이 수행될 수 있도록 운영체제 수준에서 지원을 해주어야 한다. 하지만, 앞에서 살펴본 예와 같이 동적으로 태스크가 유입되면서 집중적으로 발생되는 페이지 폴트는 멀티미디어 응용이 제한된 시간 내에 수행되지 못하는 원인을 제공하여 일시적으로 서비스 품질을 약화시키는 결과를 초래한다.

다음은 언급한 문제점을 살펴보기 위해서, 멀티미디어 응용이 수행되는 중간에 태스크가 동적으로 유입되었을 때의 결과를 보이고 있다. 실험으로 사용된 멀티미디어 응용 프로그램은 Berkeley Mpeg-1 동영상 플레이어이며, 디코딩(decoding)에 사용된 동영상 데이터는 가로 304 픽셀, 높이 228 픽셀이며, 초당 30 프레임(frame)에 화면에 출력되는 뮤직비디오를 사용하였다. (그림 6)은 동영상 플레이어가 수행되는 중간에 Netscape를 수행시킨 결과를 보이고 있으며, 태스크가 유입된 6초 근방 이후부터 제한시간을 놓치는 횟수가 갑자기 증가하고 있음을 알 수 있다. 따라서 X 윈도우 시스템 상에서 수행되는 동영상 플레이어는 6초 근방에서 갑자기 느려지거나 일시적으로 정지해 있는 것으로 보이게 된다.




(그림 6) 동적 태스크 유입에 따른 동영상 플레이어의 제한 시간 실패 현상

## 4. 비율 기반 페이지 폴트 처리 알고리즘(Rate-Based Page Fault Handling Algorithm)

앞에서 언급한 바와 같이, 가상 메모리 시스템의 문제점은 태스크가 유입될 때 발생되는 페이지 폴트 작업이 시스템 자원을 급격히 소모함으로 시스템에 일시적인 과부하를 초래하는 것이다. 이와 같이 페이지 폴트에 의한 가상 메모리 시스템의 부하가 일시적으로 높아졌을 때, 실시간 태스크 및 멀티미디어 응용의 수행에 영향을 주지 않도록 부하를 분산시키는 방법이 제공되어야 한다. 본 논문에서는 이러한 문제점을 효율적으로 해결할 수 있는 방법으로 비율 기반 페이지 폴트 처리(Rate-Based Page Fault Handling) 알고리즘을 제시하고 있다.

비율 기반 페이지 폴트 처리 알고리즘은 페이지 폴트 처리에 의한 일시적인 과부하를 해결하기 위해서 단위 시간당 발생되는 페이지 폴트 발생 빈도를 일정한 비율로 유지시키는 기법이며, 페이지 폴트 자원 할당 단계와 페이지 폴트 처리 단계로 나뉘어진다. 페이지 폴트 처리를 위한 자원 할당 방법은 프로세스가 적재되는 과정에서 페이지 폴트 처리 주기(fault_period) 및 한계 값(fault_limit)을 할당하는 것이다. 여기서 페이지 폴트 처리 주기는 페이지 폴트가 주어진 한계 값 내에서 수행되고 있는지 점검하는 시간 구간이며, 한계 값은 페이지 폴트 처리 주기에 프로세스가 사용할 수 있는 페이지 폴트 처리 상한 값을 의미한다. 페이지 폴트 처리 단계는 허가된 자원의 범위 내에서 페이지 폴트가 처리되고 있지 않을 때, 태스크의 우선 순위를 조절하는 방법을 사용하여, 일시적인 과부하를 제어한다.

비율 기반 페이지 폴트 처리 알고리즘은 페이지 폴트 처리를 위한 자원 할당을 조절하는 알고리즘이며, 실시간적으로 메모리 교체를 하는 알고리즘이 아니다. 따라서 본 알고리즘은 물리 메모리가 충분한 상황에서 태스크의 동적인 유입에 의한 과부하 제어만을 가정하고 있다.

### 4.1 페이지 폴트 처리를 위한 자원 할당 알고리즘

본 알고리즘은 실시간 태스크와 비실시간 태스크가 존재하는 범용 시스템을 대상으로 하고 있으며, 실시간 태스크는 Liu&Layland의 주기적 태스크 모델(periodic task model) [16]로 가정하고 있다. 여기서 주기적 태스크 모델은 태스크의 계산 및 데이터 전송이 일정한 주기(period)로 일정한 수행 시간(execution time)을 가지고 반복되는 것을 의미한다.

Liu&Layland의 주기적 태스크 모델에서는 태스크 집합이 스케줄링 가능한지를 판단하기 위해서 프로세서 이용률(processor utilization)을 이용한다. 태스크의 주기를 $T_i$, 예상 수행 시간을 $e_i$,라 할 때 프로세서 이용률 $U_i$는 다음과 같은 식을 따른다.

$$U_i = \frac{e_i}{T_i} \quad (1)$$

n개의 태스크를 가진 집합 $S$가 있을 때, 태스크 집합 $S$의 프로세서 이용률은 각 태스크들의 프로세서 이용률의 합이 되며, 이를 총 프로세서 이용률(total processor utilization)

$P(S)$라 한다.

$$P(S) = \sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{e_i}{T_i} \quad (2)$$

여기서 $P(S)$의 의미는 프로세서를 얼마나 이용하고 있는지를 나타내주는 수치가 되며, 가용한 프로세서 이용률을 초과해서는 안된다. 예를 들어 $P(S)$의 값이 0.5인 경우, 프로세서가 태스크의 집합 $S$의 수행으로 인하여, 50 % 정도 busy함을 나타낸다. $P(S)$의 값은 스케줄링 가능성(schedulibility)을 점검하는 한계값으로 사용되며, 프로세스 스케줄링의 정책에 따라서 한계값이 가변적이다. 잘 알려진 동적 우선 순위 스케줄러인 EDF(Earlist Deadline First)[16] 스케줄링 정책의 경우 한계값이 1이며, 따라서, EDF 스케줄링을 하는 경우 $P(S)$의 값은 1을 초과할 수 없다. 본 논문에서는 동적인 우선 순위 스케줄링을 하는 시스템으로 가정하고 있으며, 따라서 $P(S)$의 한계값은 1로 정한다.

실시간 태스크와 비실시간 태스크가 혼합되 범용 시스템 환경에서, 실시간 태스크의 집합 R과 비실시간 태스크의 집합 N이 있을 때, 실시간 태스크가 제한된 시간내에 수행되기 위해서는 프로세서 이용률 $P(R)$이 한계값 1을 초과해서는 안된다. 이 때, 비실시간 태스크들은 실시간 태스크가 수행하고 남아있는 여유 프로세서 이용률을 이용해서 수행한다. 식 (3)은 비실시간 태스크의 집합이 사용할 수 있는 프로세서 이용률의 범위를 보이고 있다.

$$0 \le P(N) \le 1 - P(R) \quad (3)$$

새로운 태스크 $\tau$가 동적으로 유입되는 단계에서 발생되는 집중적인 페이지 폴트를 처리하기 위해서 가용한 프로세서 이용률이 필요하며, 페이지 폴트 처리를 위한 작업이 실시간 태스크의 작업을 방해하지 말아야 하므로, 비실시간 프로세서 이용률에 포함되어야 한다.

단위 페이지 폴트가 소비하는 프로세서 자원을 $\theta(1)$이라 하고, $\theta(1)$은 하나의 페이지 폴트를 처리하는데 걸리는 시간으로 정의한다. 태스크 $\tau$에 의해서 발생된 총 페이지 폴트의 횟수를 $n$이라 하면, 페이지 폴트 처리를 위한 필요한 프로세서 자원은 $\theta(1)*n$이 된다. 이 때, $\theta(1)*n$는 $n$개의 페이지 폴트를 처리하는데 걸리는 예상 수행 시간이므로, 프로세서 자원을 페이지 폴트를 처리한 시간으로 나누게 되면, 페이지 폴트 처리를 하는데 소요되는 프로세서 이용률 $U_\tau$를 구할 수 있다. 따라서 페이지 폴트 처리 시간을 $I$라고 하면, 다음과 같은 식을 구할 수 있다.

$$U_\tau = \frac{\theta(1)*n}{I} \quad (4)$$

실시간 태스크가 수행하고 남아있는 프로세서 용량은 비실시간 태스크가 사용한다는 정책에 의해서, 새로운 비실시간 태스크 $\tau$가 유입되는 경우, 식 (3)은 다음과 같이 바뀌게 된다.

$$0 \le P(N) + U_\tau \le 1 - P(R) \text{ 또는}$$

$$0 \le P(N) + \frac{\theta(1)*n}{I} \le 1 - P(R) \quad (5)$$

식 (5)에서 알 수 있듯이, I의 값이 작아지면 페이지 폴트 처리를 빠른 시간 내에 할 수 있지만, 결과적으로 비실시간 태스크가 사용할 수 있는 프로세서 이용률을 초과하게 되어 실시간 태스크가 제한된 시간 내에 수행되는 것을 방해하게 된다. 반대로 I의 값이 커지면 비실시간 태스크가 사용할 수 있는 프로세서 이용률 범위 내에서 페이지 폴트를 처리할 수 있게 되며, 실시간 태스크들이 제한된 시간 내에 수행되는 것을 보장할 수 있게 된다. 따라서 페이지 폴트를 처리하는 시간을 비실시간 태스크가 수행할 수 있는 범위 내로 유지하는 것이 실시간 태스크의 제한 시간을 놓치지 않게 하는 것임을 알 수 있다. 이를 위해서, 주어진 비실시간 태스크의 프로세스 이용률의 범위 내에서 어느 정도의 페이지 폴트를 처리할 것인지를 결정하는 것이 필요하다.

비실시간 태스크에게 할당된 프로세스 이용률의 한계값을 $\alpha$라고 할 때, $\alpha$의 범위 내에서, 페이지 폴트를 처리 작업이 수행될 수 있도록, 페이지 폴트 처리 주기와 해당 주기에 몇 개까지의 페이지 폴트를 처리할 수 있는지의 관계로 나타낼 수 있다. $\theta(1)$은 앞에서 언급했듯이, 단위 페이지 폴트를 처리하는데 소요되는 프로세서 시간을 의미하며, 프로세서 이용률이 1인 경우에 처리되는 페이지 폴트의 개수는 $\frac{1}{\theta(1)}$이 된다. 따라서 $\alpha$ 만큼의 프로세서 이용률이 주어진 경우에 처리할 수 있는 페이지 폴트의 개수는 $\frac{\alpha}{\theta(1)}$이 된다. 이 때, 페이지 폴트를 처리할 주기를 ***fault_period***, 해당 주기에 페이지 폴트를 처리할 개수를 ***fault_limit***라고 할 때, 매 주기 ***fault_period***에 처리할 수 있는 페이지 폴트의 개수 ***fault_limit***는 다음과 같다.

$$fault\_limit = \frac{\alpha}{\theta(1)} * fault\_period \quad (6)$$

즉, 태스크에게 페이지 폴트 처리를 위해서 주어진 프로세서 이용률 $\alpha$가 0.2이고, 단위 페이지 폴트 $\theta(1)$가 소비하는 자원이 0.0001이라고 하였을 때, 페이지 폴트 처리 주기(fault_period) 100ms마다 처리할 수 있는 페이지 폴트 처리 개수(fault_limit)는 $fault_limit = \frac{0.2}{0.0001} * 0.1$에 의해서 200개가 된다. 따라서 비실시간 태스크에게 주어진 프로세서 이용률 $\alpha$가 0.2인 경우, 100ms 주기에 발생할 수 있는 페이지 폴트 처리는 200개를 넘지 않아야 하며, 결과적으로 집중적으로 발생되는 페이지 폴트는 실시간 태스크 및 멀티미디어 태스

크의 수행에 지장을 주지 않게 된다.

### 4.2 페이지 폴트 처리 단계

비율에 기반한 페이지 폴트 처리 단계에서는 각 태스크의 페이지 폴트 처리 주기 동안에 fault_limit를 초과하는 요청이 들어올 경우 더 이상의 페이지 폴트가 처리되는 것을 제한하고 있다. 따라서 페이지 폴트 처리 주기 내에서 페이지 폴트가 주어진 한계 이상으로 빈번하게 발생될수록 태스크의 우선 순위를 저하시키거나 또는 수행을 정지시키는 방법을 사용한다. 페이지 폴트 처리 단계의 알고리즘은 (그림 7)과 같다.

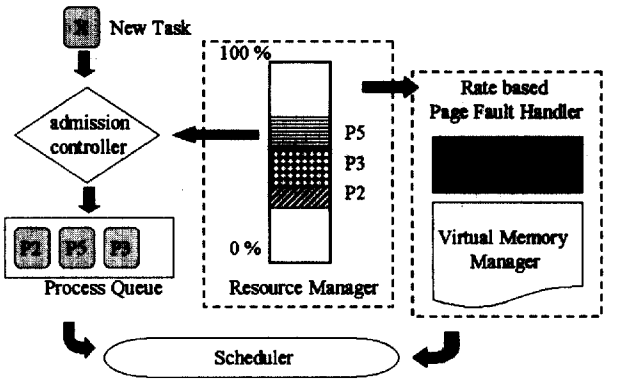
페이지 폴트를 처리하는 가상 메모리 시스템 부분에서 페이지 폴트 제어 모듈을 수행시킨다. 제어 모듈은 페이지 폴트 처리 주기가 만료되었는지 확인하며, 페이지 폴트 처리 주기가 만료된 경우 새로운 페이지 폴트 처리 주기를 세팅하고 이전 주기 동안 페이지 폴트가 발생된 횟수를 담고 있는 fault_count의 값을 0으로 초기화 해준다. 만약 페이지 폴트 처리 주기가 만료되지 않은 상태에서 페이지 폴트 처리 한계값을 초과하는 페이지 폴트가 요청되는 경우 현재 수행중인 태스크의 우선 순위를 하락시키는 방법이나 현재 수행을 정지시키는 방법을 통해서 제한된 범위 이상의 페이지 폴트 처리를 제제하게 된다.

```
if (fault_period is expired) {
  fault_period = NEW_FAULT_PERIOD ;
  fault_count = 0 ;
} else {
  fault_count++ ;
  If(fault_count > fault_limit) {
    suspend task or adjust priority ;
  }
}
```

(그림 7) 페이지 폴트 처리 알고리즘

### 4.3 알고리즘 구현

본 논문에서 제안한 알고리즘을 적용하기 위하여, 시스템 자원에 대한 모니터링 및 자원 예약이 가능하도록 Linux 커널을 수정하였으며, 수정된 Linux 커널을 LMX(Linux Multimedia Extension)라 부른다. LMX는 태스크를 실시간 태스크(real-time task)와 범용 태스크(general task)로 분류하며, 커널은 실시간 태스크가 사용하는 시스템 자원을 계속적으로 모니터링하며, 실시간 태스크가 사용하지 않는 자원은 범용 태스크가 사용할 수 있도록 관리한다. 다음 (그림 8)은 LMX의 구성 요소와 동작 과정을 보여주고 있다. LMX는 승인제어부(addmision controller)와 자원 관리부(resource manager), 비율 기반 페이지 폴트 핸들러로 구성되어 있으며, LMX의 자원 관리 모듈에서 유지하는 값은 각 실시간 태스크의 자원 사용량, 전체 범용 태스크의 자원 사용량, 유휴 자원량 등이다.




(그림 8) LMX 모듈 구성도

새로운 실시간 태스크가 유입될 때, LMX의 승인 제어부(admission control module)는 각 실시간 태스크의 자원 사용량과 유입된 태스크의 자원 사용량의 합계가 지정된 시스템 자원 한계를 초과하지 않는 경우 실시간 태스크를 수행시키게 된다. 이때 시스템 자원 전체를 실시간 태스크에게 할당하지 않고, 일부는 범용 태스크가 수행 될 수 있도록 예약하는 방식을 사용하고 있으며, 범용 태스크는 유휴 자원량이 남아 있는 경우에 수행이 허가되며, 남아 있는 유휴 자원량의 범위 내에서 페이지 폴트 작업이 수행될 수 있도록 한다. 자원 예약 기법을 활용하기 위해서 현재 시스템의 자원 사용 현황에 대해서 실시간 통계를 가지고 있어야 하며, 페이지 폴트가 소모하는 시스템 자원에 대해서도 미리 측정된 결과를 가지고 있어야 한다. 즉 페이지 폴트에 따른 프로세서 자원 소모율 정보를 시스템에서 유지하고 있어야 하며, 이러한 정보를 이용해서 현재 가용한 시스템 자원의 범위 내에서 새롭게 유입된 태스크에 대해서 페이지 폴트를 허가하게 되는 것이다.

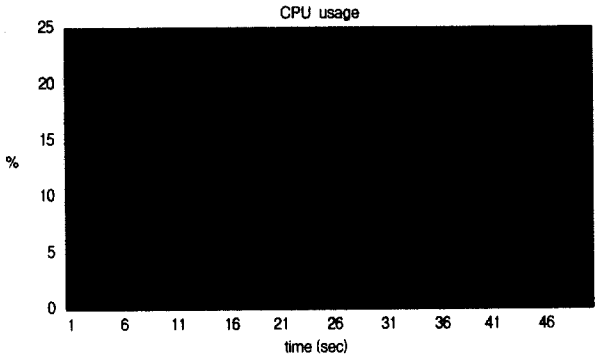
## 5. 실험 결과

이번 장에서는 본 논문에서 제시하는 비율 기반 페이지 폴트 처리 방식과 기존의 Linux 시스템을 상호 비교한다. 실험의 목적은 페이지 폴트가 집중적으로 발생되는 시점에서 본 논문이 제시하는 알고리즘을 적용시켰을 때 페이지 폴트에 의한 과부하가 효율적으로 분산되는지 확인하는 것이며, 이때 멀티미디어 응용 프로그램이 제한시간을 잘 지키면서 수행되는지 살펴본다. 실제 범용 운영체제 환경과 유사하게 하기 위해서 연성 실시간 태스크인 멀티미디어 응용 프로그램과, 배치 작업(batch task)에 속하는 수식 계산 프로그램 등을 수행시킨 후 응용 프로그램을 동적으로 유입시키면서 실험을 하였다.

### 5.1 실험 환경

실험에 사용된 Linux 시스템과 LMX는 Pentium 500MHz, 128MByte RAM을 장착하였으며, Linux kernel 2.4를 사용하였다. 그리고 실험에 사용되는 단위 페이지 폴트 처리에 소요되는 $\theta(1)$은 페이지 폴트를 계속해서 발생시키는 프로그램을 반복 수행시켜서 얻은 평균값인 0.0004를 사용하였다. 또한 본 실험에서는 LMX의 RBPFH 기법과 기존의 Linux 시스템과의 공정한 비교 및 실시간 스케줄링 효과를 제거하기 위해서 프로세스 스케줄링 정책을 Linux의 기본 스케줄러인 SCHED_OTHER[5]를 사용하였다. 따라서 멀티미디어 태스크와 일반 수식 계산 및 배치 프로그램이 수행될 경우 동일한 스케줄링 정책에 따라서 스케줄링되며, 프로세서를 많이 사용하는 태스크의 우선 순위가 저하되는 특성을 가지게된다.

실험에 사용된 mpeg_player는 Berkeley에서 개발한 MPEG-1 동영상 플레이어이며, 디코딩에 사용된 비디오 데이터는 총 8874개의 프레임으로 구성되었다. 비디오 데이터의 전체 수행 시간은 295.80초, 프레임 넓이는 304 픽셀(pixel), 프레임의 높이는 228 픽셀(pixel), 평균 프레임의 크기는 3721byte 그리고 평균 전송률은 893096bit/sec의 특성을 갖는다. (그림 9)는 동영상 플레이어가 단독으로 수행되는 상황에서 소모하는 프로세서 자원을 보여주고 있으며, 평균적으로 25% 근방의 프로세서 자원을 소모하고 있음을 알 수 있다.

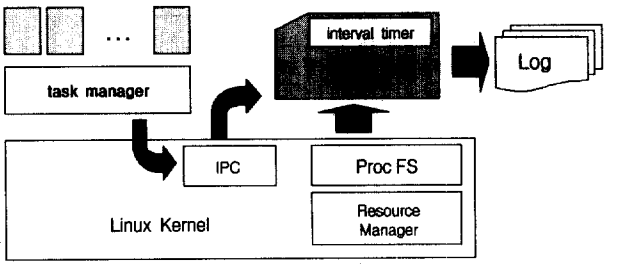



(그림 9) 실험에 사용된 동영상 플레이어의 프로세서 이용률

실험에서 배치 작업의 특성을 가지고 수행된 수식 계산 프로그램은 CPU 중심의 작업이며 프로그램의 크기는 13Kbyte이므로 물리 메모리를 차지하는 비율이 무시할만한 수준이다. 그리고 수식 계산 프로그램은 동영상 플레이어와 동시에 수행이 되면서 동영상 플레이어가 사용하지 않고 있는 프로세서 이용률 중에서 X 윈도우 시스템이 사용하고 있는 프로세서 이용률 5%를 제외한 나머지 70%에 해당되는 프로세서 자원을 이용하여 수행하게 된다. 따라서 새로운 태스크가 유입이 되는 경우, 현재 시스템에서 수행중인 동영상 플레이어, X 윈도우 시스템, 수식 계산 프로그램이 사용중인 프로세서 자원을 분할해서 사용하게 된다.

### 5.2 실험 결과 측정 방법

본 논문에서 태스크의 페이지 폴트 처리 횟수, 메모리 사용량, 프로세스 사용량을 구하기 위하여 proc 파일 시스템[5]의 정보를 이용하고 있다. Proc 파일 시스템은 리눅스의 가상 파일 시스템으로, 프로세스 관련 정보, 메모리 정보, 디바이스 정보 및 기타 커널의 수행 상태에 대한 정보를 제공하고 있다. 본 논문에서는 비율 기반 페이지 폴트 처리 알고리즘을 구현하면서 추가적으로 생성된 정보인 프로세스 사용량, 페이지 폴트 처리 비율 등을 읽기 위해서 새로운 시스템 호출을 추가하지 않고, proc 파일 시스템에 정보를 추가하는 방법을 사용하였다. (그림 10)에서 볼 수 있듯이, 실험 태스크를 관리하고 필요한 부하(load)를 생성하기 위해서 task manager가 존재하며, task manager는 멀티미디어 태스크 및 범용 태스크를 수행시키고 관리하는 역할을 한다. resource monitor는 주기적인 타이머에 의해서 동작하며, 100ms의 간격으로 실험 태스크의 페이지 폴트 처리 횟수 및 메모리 사용량에 대한 정보를 수집하는 역할을 하며, task manager와 resource monitor는 IPC(message queue)를 통해서 정보를 주고 받는다.




(그림 10) 실험 결과 측정 환경

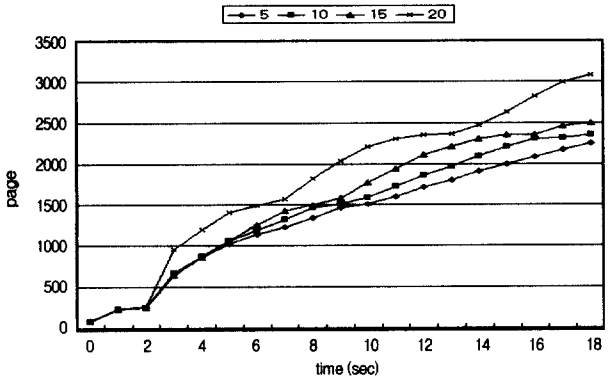
### 5.3 비율의 변화에 따른 실험 결과

본 실험은 페이지 폴트 처리를 위해서 할당된 프로세서 이용률과 과부하 처리에 대한 상관 관계를 보이고 있다. 다음 (그림 11)은 동영상 플레이어 및 수식 계산 프로그램, X 윈도우 시스템이 수행중인 환경에서 Netscape 프로그램을 동적으로 유입시키면서 Netscape가 차지하는 메모리 사용량을 보이고 있다. 이 때, fault_period와 fault_limit의 비율을 변화시키기 위해서 fault_period를 50ms로 고정시키고 fault_limit의 값은 5에서 20까지 조절하면서 실험을 하였다. 앞에서 언급했듯이 단위 페이지 폴트 처리를 위해서 소모되는 $\theta(1)$이 0.0004이므로, 페이지 폴트 처리에 할당된 비실시간 태스크의 프로세서 이용률 $\alpha$는 다음과 같다.

- fault_period : 50ms, fault_limit : 5 → $\alpha$ : 4%

- fault_period : 50ms, fault_limit : 10 → $\alpha$ : 8%
- fault_period : 50ms, fault_limit : 15 → $\alpha$ : 12%
- fault_period : 50ms, fault_limit : 20 → $\alpha$ : 16%

실험 결과, Netscape의 페이지 폴트 처리를 위해 할당된 비실시간 태스크의 프로세서 이용률 $\alpha$가 높아질수록 메모리의 사용량이 급격히 높아지는 것을 볼 수 있으며, 반대로 폴트 처리를 위한 $\alpha$의 값이 낮아질수록 메모리의 사용량이 완만히 높아지는 것을 확인할 수 있다. 즉 동적으로 유입된 태스크를 위한 페이지 폴트 처리 비율을 낮출수록 페이지 폴트를 처리하는 비율이 줄어들면서 전체적으로 메모리 사용량이 완만하게 증가하는 것을 알 수 있다.



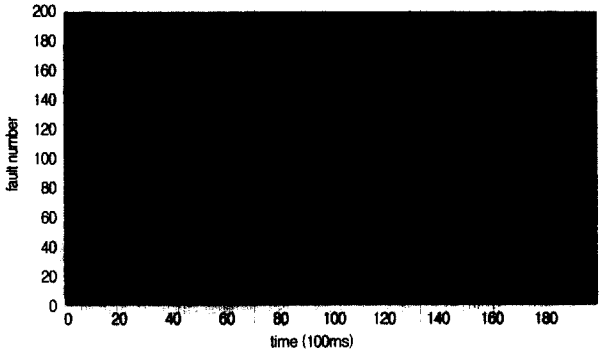

(그림 11) 폴트 처리 비율에 따른 메모리 사용량 변화

### 5.4 Netscape와 gimp의 동적 유입 결과

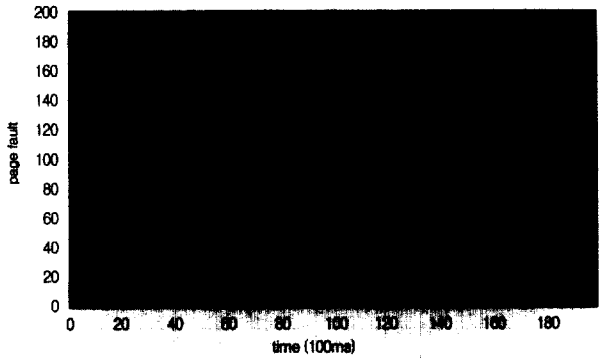
(그림 12)는 기존의 Linux 상에서 Netscape를 동적으로 유입시켰을 때의 결과를 보여주고 있으며, (그림 13)은 LMX에서 Netscape를 동적으로 유입시켰을 때의 결과이다. 이 때, $\alpha$는 8 %로 고정시켜서 실험하였다. 실험 결과를 살펴보면, Linux에서 프로그램 수행 초기에 과도하게 발생하는 페이지 폴트 비율이 LMX에서는 완만한 수준으로 유지되고 있음을 알 수 있다.

비율 기반 페이지 폴트 처리 알고리즘을 사용함에도 불구하고 (그림 13)과 같이 4초 근방에서 급격히 페이지 폴트가 발생하는 이유는 Linux와 같은 비실시간 커널이 가지고 있는 근본적인 문제점에서 기인한다. 왜냐하면, 실험에서 Linux의 기본 스케줄러인 SCHED_OTHER를 이용하여 프로세스 스케줄링을 하고 있으므로 실시간 특성이 보장되지 않으며, 각 프로세스는 주어진 퀀텀을 모두 소모하거나 프로세서(processor)를 많이 사용해서 우선 순위가 떨어지기 전까지 계속적으로 작업이 이루어지기 때문이다. 따라서 이러한 문제를 해결하기 위해서는 궁극적으로 실시간 스케줄러를 도입하여야 한다.

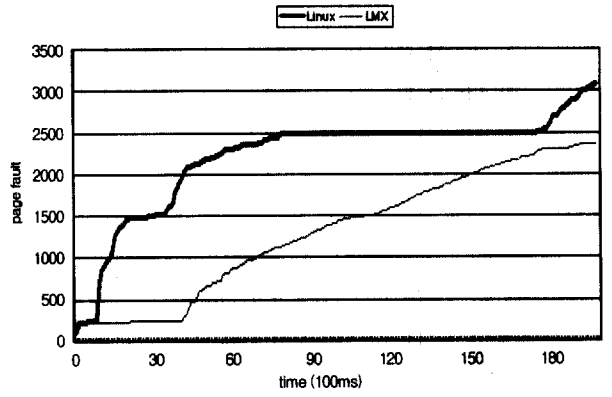
(그림 14)는 Netscape가 시간에 따라서 사용하고 있는 물리 메모리 양의 변화를 보여주고 있다. LMX는 점진적으로 메모리 양을 늘려주고 있는데 비해서 Linux는 프로그램 수행 초반에 과도하게 물리 메모리를 할당 해주고 있음을 알 수 있다. 이러한 이유는 앞에서도 언급했듯이, 페이지 폴트 발생 빈도와 밀접한 연관 관계가 있기 때문이다.




(그림 12) Linux에서 Netscape의 페이지 폴트 발생 횟수




(그림 13) LMX에서 Netscape의 페이지 폴트 발생 횟수




(그림 14) Linux와 LMX에서 Netscape의 메모리 사용량

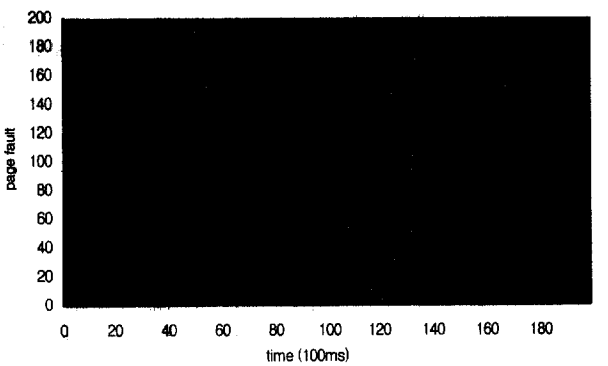
다음은 동적으로 태스크가 유입되는 환경에서 멀티미디어 응용 프로그램이 원활하게 수행되는지 확인하기 위해서, 동영상 플레이어의 수행 결과를 <표 2>에서 보였다. <표 2>에서 동영상 플레이어의 제한 시간 실패율이 10 % 정도 줄

었음을 관찰할 수 있으며, 제한 시간 실패 프레임이 Linux에서는 평균적으로 15ms 지연되었지만, LMX에서는 7ms로 지연시간이 줄어들었음을 알 수 있다. 멀티미디어 응용의 품질을 평가함에 있어서 제한 시간 실패 횟수도 중요하지만, 제한 시간이 실패했을 때의 평균 지연 시간이 어떤값을 갖는지도 중요하다. 왜냐하면, 멀티미디어 응용이 제한 시간 내에 수행되지 못하였어도, 제한 시간의 근방에서 수행이 완료되면 멀티미디어 응용의 품질에 큰 손상이 없기 때문이다. 실험 결과에서 LMX는 Linux에 비해서 제한 시간의 실패 횟수를 크게 줄였으며, 평균 지연 시간도 두 배 이상 줄였음을 보였다.

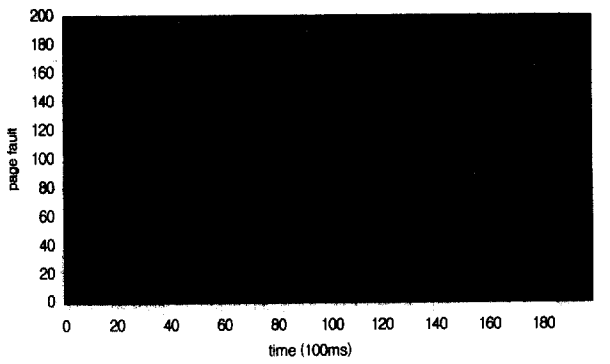
〈표 2〉 Mpeg-1 플레이어의 제한시간 실패율 및 평균 지연 시간

| | frame number | deadline miss | miss rate(%) | average delay |
|---|---|---|---|---|
| LINUX | 500 | 161 | 32.2 | 15054 |
| LMX | 500 | 110 | 22 | 6954 |

다음은 그래픽 이미지 툴인 gimp를 동적으로 유입시켰을 때 실험결과를 보이고 있다. 실험 결과에서 Netscape와 비슷한 경향을 보이고 있으며, 기존의 Linux에서는 (그림 15)와 같이 프로그램 실행 초반부에 집중적으로 나타나는 페이지 폴트가 LMX에서는 (그림 16)에서 보듯이 일정한 기간 동안 과부하가 분산되는 것을 확인할 수 있다.



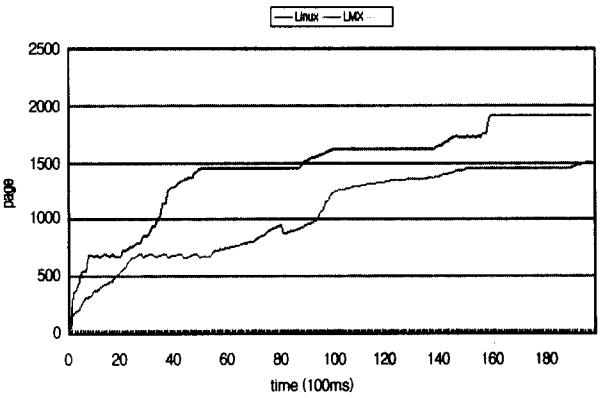

(그림 15) Linux에서 gimp의 페이지 폴트 발생 횟수




(그림 16) LMX에서 gimp의 페이지 폴트 발생 횟수

(그림 17)에서는 gimp 프로그램의 시간에 따른 메모리 사용량을 보이고 있다. 프로그램 시작 초반부에 집중적으로 메모리를 요구하는 Linux에 비해서 LMX는 완만한 경사를 보이고 있음을 알 수 있다.

gimp 프로그램이 동적으로 유입되는 환경에서 동영상 플레이어를 수행시켰을 때, <표 3>에서 보듯이 제한 시간 실패율은 20% 향상되었으며, 평균 지연 시간도 7ms 낮출 수 있었다. 따라서 netscape의 실험 결과와 비슷한 경향을 보이고 있음을 알 수 있다.




(그림 17) Linux와 LMX에서 gimp의 메모리 사용량

〈표 3〉 gimp의 페이지 폴트 발생 횟수 및 평균 지연 시간

| | frame number | deadline miss | miss rate(%) | average delay |
|---|---|---|---|---|
| LINUX | 500 | 234 | 46.8 | 14194 |
| LMX | 500 | 131 | 26.2 | 7742 |

## 6. 결론 및 향후 계획

범용 시스템 환경에서 사용자는 임의의 시점에서 새로운 태스크를 유입시키거나, 종료시키는 작업을 반복하게 된다. 이러한 동적인 환경에서 멀티미디어 응용을 수행하고자 할 때 태스크의 유입에 의한 일시적인 과부하는 멀티미디어 서비스 품질을 악화시키는 요인으로 작용하게 된다. 본 논문에서는 동적으로 유입되는 태스크가 시스템에 일시적인 과부하를 초래하게 되고, 따라서 일정한 제한시간을 가지고 수행되어야 하는 태스크들에게 심각한 영향을 초래한다는 문제점에 대해서 살펴보았다. 또한 동적으로 태스크가 유입 될 때, 대부분의 과부하가 페이지 폴트를 처리하는 부분에서 발생되고 있으며, 페이지 폴트 모듈을 포함하는 가상 메모리 시스템에 있어서 과부하를 조절하는 기능이 필요한 이유에 대해서 언급하고 효율적으로 과부하를 조절할 수 있는 기법을 제시하였다. 본 논문에서 제시한 기법은 범용 운영체제 환경에서 멀티미디어 응용과 같은 연성 실시간 태스크를 일시적인 과부

하로부터 보호하기 위한 방법으로 가상 메모리 시스템의 자원 사용율에 제한을 가하는 비율 기반의 페이지 폴트 처리 방식이다.

본 논문의 주요한 기여도는 동적인 태스크 유입으로 발생되는 일시적인 과부하를 해결하고 있다는 것이며, 기존의 연구에서는 이러한 문제점을 해결하기 위한 연구가 부족하였다. 기존의 연구에서, 일시적인 과부하를 해결하기 위해서 멀티미디어 태스크의 품질을 조절하는 방법이 거론되었으나, 이는 지속적인 과부하의 처리에 유용한 방법이며, 일시적인 과부하 처리를 위해서는 과도한 오버헤드를 주게 된다. 따라서 본 논문에서 제시하는 기법은 새로운 태스크가 동적으로 유입되는 환경에서 기존의 멀티미디어 태스크의 서비스 품질을 계속적으로 유지할 수 있도록 하는 효율적인 방법이며, 구현 및 수행 중에 발생되는 오버헤드를 최소화시킨 방법이다.

본 논문에서 제시한 기법은 프로그램 시작 시에 집중적으로 발생하는 페이지 폴트를 분산시키는 특성을 보이고 있으며, 이러한 방법으로 멀티미디어 응용이 제한된 시간내에 수행될 수 있는 기반을 제공하고 있다. 향후 물리 메모리에 적재된 페이지가 메모리 부족에 의해서 페이지 교체되거나 스와핑(swapping)됨으로 멀티미디어 응용이 제한 시간을 놓치게 되는 문제점을 연구할 계획이다.

## 참 고 문 헌


[1] C. W. Mercer, S. Savage, H. Tokuda, "Processor Capacity reserves : Operating System Support for Multimedia Applications," Proceedings of the IEEE international Conference on Multimedia Computing and Systems, Boston, MA, pp.90-99, May, 1994.

[2] Raj Yavatkar, K. Lakshman, "A CPU Scheduling Algorithm for Continuous Media Applications," In 6th International NOSSDAV Workshop.

[3] P. Goyal, X. Guo, H. M. Vin, "A Hierarchical CPU Scheduler for Multimedia Operating Systems," Proceedings of the Second Symposium on Operating Systems Design and Implementation, Seattle, WA, pp.107-122, Oct. 1996.

[4] J. Nieh and Monica S. Lam, "The Design, implementation and Evaluation of SMART : A Scheduler for Multimedia Applications," Proceedings of 16th ACM Symposium on Operating Systems Principles, St Malo, France, October, 1997.

[5] Michael Beck, et al. "Linux Kernel Internals second Edition," Addison-Wesley, 1998.

[6] Silberschatz Galvin. "Operating System Concepts," Addison-Wesley, 1994.

[7] Ralf Steinmetz and Klara Nahrstedt, Multimedia. Computing "Communications and Applications," Prentice-Hall, Englewood Cliffs, NJ, 1995.

[8] I. Stoica, H. Abdel-Wahab, K. Jeffey, S. Baruah, J. Gehrke, and G. Plaxton, "A Proportional Share Resource Allocation Algorithm for Real-Time, Time-shared Systems," Proc. of Real-Time Systems Symposium, pp.288-299, Dec. 1996.

[9] M. B. Jones, D. Rosu, and M-C. Rosu, "CPU Reservation and Time Constraints : Efficient, Predictable Scheduling of Independent Activities," Proc. of the 16th ACM Symposium on Operating System Principles, pp.198-211, Oct. 1997.

[10] K. Jeffay, F. D. Smith, A. Moorthy, and J. Anderson, "Proportional Share Scheduling of Operating System Services for Real-Time Application," Proc. of Real-Time Systems Symposium, pp.480-491, Dec. 1998.

[11] Michael Barabanov, "A Linux-based Real-Time Operating System," Master thesis, New Mexico Institute of Mining and Technology, June, 1997.

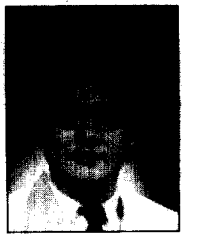
[12] J. Stankovic and K. Ramamritham, "The SPRING Kernel : A New Paradigm for Real-Time Systems," IEEE Software, Vol.8, No.3, pp.62-72, May, 1991.

[13] Ray Ford, "Concurrent Algorithms for Real-Time Memory Management," IEEE Software, Vol.5, Issue 5, pp.10-23, 1988.

[14] Kelvin D. Nilsen and Hong Gao, "The Real-Time Behavior of Dynamic Memory Management in C++," Proc. of Real-Time Technology and Application Symposium, pp. 142-153, 1995.

[15] Charles B. Weinstock, "Dynamic Storage Allocation Techniques," Ph.D. thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1976.

[16] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," Journal of the ACM, Vol.20, No.1, pp.40-61, 1973.


**고 영 웅**

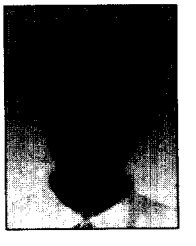
e-mail : yuko@os.korea.ac.kr

1997년 고려대학교 컴퓨터학과 졸업(학사)

1999년 고려대학교 대학원 컴퓨터학과 (이학석사)

1999년~현재 고려대학교 대학원 컴퓨터학과 박사과정

관심분야 : 멀티미디어 운영체제, 실시간 운영체제

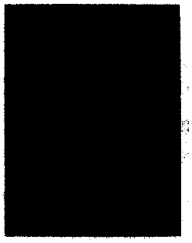

**아 재 용**

e-mail : jyah@os.korea.ac.kr

2000년 고려대학교 컴퓨터학과 졸업(학사)

2000년~현재 고려대학교 대학원 컴퓨터학과 석사과정

관심분야 : 운영체제, 보안 운영체제, 내장형 시스템



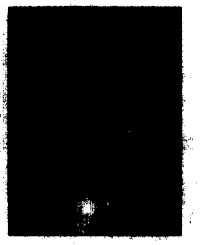
**홍 철 호**

e-mail : chhong@os.korea.ac.kr

2001년 고려대학교 컴퓨터학과 졸업(학사)

2001년~현재 고려대학교 대학원 컴퓨터학과 석사과정

관심분야 : 운영체제, 보안 운영체제, 보안 평가 도구



**유 혁**

e-mail : hxy@os.korea.ac.kr

1982년 서울대학교 전자공학과 학사

1984년 서울대학교 전자공학과 석사

1986년 University of Michigan 전산학 석사

1990년 University of Michigan 전산학 박사

1990년~1995년 Sun Microsystems Lab. 연구원 & Sun Microsystems Lab. 초빙 연구원

1995년~현재 고려대학교 이과대학 컴퓨터학과 교수

관심분야 : 운영체제, 네트워크, 멀티미디어, thin client