2002년 한국정보처리학회 춘계학술발표논문집 제9권 제1호

연성 실시간 태스크를 위한 메모리 관리 기법

고영웅*, 홍철호, 유혁
고려대학교 컴퓨터학과
e-mail : {yuko, chhong, hxy} @os.korea.ac.kr

Memory Management for Soft Real-time Task

Young-Woong Ko*, Cheol-Ho Hong, Chuck Yoo
Dept. of Computer Science, Korea University

요 약

실시간 태스크는 제한된 시간 내에 작업이 수행되어야 하는 특성을 가지고 있으며, 이러한 특성을 만족시켜주기 위해서 필요한 자원이 확보되어 있어야 한다. 본 논문에서는 제한 시간이 완화된 연성 실시간 태스크를 수행시킴에 있어서 기존에 제시된 메모리 관리 기법의 문제점을 분석하고, 연성 실시간 태스크가 원활히 수행될 수 있도록 하는 방법을 제시하고 있다. 본 논문에서 제시하는 방법은 프로세스 스케줄러와 메모리 관리 스케줄러가 상호 협조적으로 동작하는 것에 초점을 맞추고 있다. 즉, 프로세스 스케줄링 정보와 물리 메모리 사용량을 기초로 연성 실시간 태스크가 필요로 하는 메모리 자원이 page-out 되거나 swap-out 되는 비율을 조정하고 있다. 제시된 방법을 실험을 통하여 검증을 하였으며, 실험 결과에서는 물리 메모리의 부족으로 인한 과부하 상황에서 연성 실시간 태스크의 제한 시간 실패율을 감소되고 전체적인 성능이 향상됨을 보이고 있다.

1. 서론

최근 멀티미디어 응용과 같은 연성 실시간 태스크를 수행시키기 위한 연구가 진행되고 있으며, 특히 프로세스 스케줄링 분야에서 SMART, Raialto, Proportional share Scheduling, Feedback based scheduling 등의 연구 결과들이 제시되었다. 그러나 연성 실시간 태스크가 제한시간을 지키면서 수행되기 위해서는 프로세스 스케줄링만으로는 한계가 있다. 왜냐하면 프로세스가 제대로 수행되기 위해서는 프로세서(processor) 이외에 물리 메모리(physical memory) 및 디스크 대역폭(disk bandwidth)과 같은 자원이 충분히 확보되어 있어야 하기 때문이다. 따라서 연성 실시간 태스크가 이상적으로 수행되기 위해서는 운영체제는 태스크가 필요로 하는 모든 자원들에 대한 관리를 효율적으로 제공해야 하는 필요성을 가진다.

특히 물리 메모리를 관리하는 부분에 있어서, 현재의 범용 운영체제 환경은 여러 가지 문제점을 가지고 있다. 첫번째로 언급할 수 있는 것은 응용 프로그램이 비대해지면서 점점 더 많은 물리 메모리를 요구하고 있다는 것이다. 메모리의 가격이 하락하면서 범용 시스템에는 수십 메가 바이트 이상의 물리 메모리를 장착하고 있지만, 멀티미디어 응용 프로그램 및 그래픽 사용자 인터페이스를 가지는 응용 프로그램이 널리 보편화되면서 물리 메모리의 부족현상을 초래하고 있다. 두 번째로 대부분의 범용 운영체제는 다양한 응용 프로그램의 특성을 수용하는 메모리 관리 기법이 제공되고 있지 않다. 널리 사용되는 대부분의 운영체제에 있어서 메모리가 부족한 경우, LRU(Least Recently Used) 방식에 기초한 페이지 교체 정책을 사용하고 있으며, 이것은 시스템에서 수행중인 모든 태스크에 대해서 일률적으로 적용되는 방식이다. 따라서 태스크의 중요도나 연성 실시간 특성의 반영되지 않으며, 시스템 전역적인 메모리 관리 기능에 초점이 맞추어져 있다. 세 번째로 프로세서는 특정 태스크에게 할당되었을 때, 타임 퀀텀(time quantum)이 만료되거나 우선순위가 높은 태스크에게 선점되는 경우를 제외하고는 다른 태스크의 방해 없이 독점적으로 수행될 수 있는 자원인 반면에 특정 태스크에게 할당된 물리 메모리는 공간적(spatial), 시간적(temporal)으로 여러 태스크의 영향을 받아서 점유량이 변할 수 있는 공유 자원의 특성을 가진다. 따라서 어떤 태스크가 프로세서를 할당 받아서 수행하고자 할 때, 수행 되는 일정 시간에 필요한 코드, 데이터, 힙(heap) 그리고 스택(stack) 등이 물리 메모리 상에 존재해야 원활한 수행이 보장된다. 그렇지 않는 경우 페이지 폴트 처리 루틴이 수행되어

서 수행 가능한 조건이 만족될 때까지 지연이 발생한다.

따라서 앞에서 언급한 범용 운영체제의 메모리 관리 문제점을 해결하기 위한 방법으로 본 논문에서는 프로세스 스케줄러와 메모리 스케줄러의 상호 연동을 통하여 연성 실시간 태스크의 특성이 메모리 스케줄러에 반영될 수 있는 자원 스케줄러간의 협조 모델을 제안하였다. 2 장에서는 실시간 메모리 관리에 대한 관련 연구를 기술하고, 3 장에서는 본 논문에서 제안하는 자원 스케줄러간의 협조 모델에 대해서 언급하며, 4 장에서는 실제 리눅스 운영체제 상에 구현한 자원 스케줄러간의 협조 모델의 세부 사항을 기술한다. 5 장에서는 실험을 통한 본 논문의 유용성을 보이고 있으며, 6 장에서 결론을 내린다.

2. 관련 연구

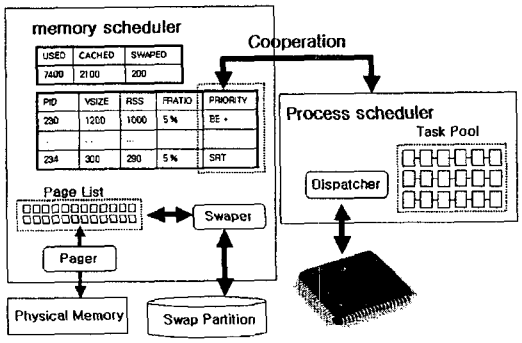
Nemesis 운영체제에 구현된 self-paging[1] 기법은 커널이 관리하는 페이징 작업을 사용자 영역에 존재하는 응용 프로그램에서 수행하게 하였으며, 커널은 페이지 폴트가 발생되었음을 알리는 기능을 담당하도록 하였다. Self-paging 기법은 사용자 영역에서 페이저(pager)가 수행된다는 측면에서 mach, Spring 및 L4 와 같은 마이크로 커널의 external pager 와 접근 방법이 유사하나, 페이저가 독립 서버로 존재하지 않고, 각 응용 프로그램이 개별적으로 페이저를 가지고 있다는 점에서 차이가 난다. Harty[2]의 V++ external page-cache 관리 기법은 각 응용 프로그램이 자신에게 할당된 물리 페이지를 관리한다는 측면에서 self-paging 과 유사하며, external pager 가 응용 프로그램의 주소 공간에 포함되거나 또는 독립적인 모듈로 존재할 수 있는 방안을 제시하였다.

Exokernel[3]은 하드웨어를 멀티플렉싱 시켜주는 기능을 담당하는 마이크로 커널 위에서 응용 프로그램과 응용 프로그램에 특화된 라이브러리 운영체제가 결합되어 동작되고 있다. 따라서 각 응용 프로그램이 필요로 하는 메모리 관리 정책을 라이브러리 운영체제에 적용시키는 방법의 메모리 관리 기법을 제공해 줌으로써 특정 응용 프로그램의 성능을 최대화 시켜 줄 수 있는 장점이 있다. 하지만 Exokernel 은 다양한 응용 프로그램이 동작하는 멀티 태스킹 환경에 적합하지 않은 한계를 지니고 있다.

대표적인 범용 운영체제인 솔라리스(Solaris) 운영체제[4]에서는 메모리 스케줄러가 존재하며, 부팅시에 동작이 시작된 후, 자유 메모리의 용량이 일정한 수준 이하(desfree)로 떨어지게 되면, swap out 을 시킬 프로세스를 선택한다. 이때, 메모리 스케줄러는 메모리 부족 정도에 따라서 연성 스왑 아웃(soft swap out) 방식과 경성 스왑 아웃(hard swap out) 방식으로 동작하게 된다. 연성 스왑 아웃은 메모리가 일정 용량이하로 떨어졌을 때, 일정 시간 이상(20 초가 디폴트 값)으로 수면 상태에 있는 프로세스를 선택하여 그 프로세스가 개별적으로 소유하고 있는 모든 페이지를 스왑 아웃 시킨다. 경성 스왑 아웃은 페이지 아웃과 페이지 인 되는 값의 합이 정해진 최대 페이지 입출력(maxpgio)의 값보다 크게 되면 발생하며, 이때는 커널 모듈과 프로세스 메모리가 제거된다. 그러나 솔라리스 메모리 스케줄러는 범용 태스크들이 동시에 수행되는 멀티 프로세스 환경에서 적합한 방법으로 사용 가능하지만, 연성 실시간 태스크가 메모리 사용량을 보장 받고 수행되는데 있어서 한계를 가지고 있다.

3. 자원 스케줄러 협조 모델

본 논문에서 제시하는 자원 스케줄러간의 협조 모델은 연성 실시간 태스크와 범용 태스크가 공존하는 환경에서 연성 실시간 태스크의 제한 시간을 만족시키고 범용 태스크가 메모리 부족으로 인한 기아 현상이 발생되는 것을 방지하는 것을 목적으로 하고 있다. 이러한 목적을 달성하기 위하여 자원 할당 스케줄러는 크게 프로세스 스케줄러와 메모리 스케줄러로 분할하고 스케줄링 정보를 교환하는 구조로 설계되었다. 다음 그림 1 은 자원 스케줄러간의 협조 모델을 도식하고 있다.



[그림 1] 자원 스케줄러간의 협조 모델

메모리 스케줄러는 시스템 전체적인 메모리 자원 상황에 대한 정보와 각 프로세스의 메모리 사용 정보를 가지고 있다. 대표적인 정보는 프로세스가 사용중인 물리 페이지(RSS), 일정한 주기 동안에 발생된 페이지 아웃(page out) 및 페이지 인(page in)의 비율(FRATIO) , FRATIO 의 방향(DFLAG) 등이다. 다음은 메모리 관리 정책을 연성 실시간 태스크에 대한 것과 범용 태스크에 대한 것으로 나누어 설명한다.

3.1 연성 실시간 태스크 메모리 관리

연성 실시간 메모리 스케줄링 알고리즘은 연성 실시간 태스크의 수행을 보장하기 위해서 메모리 과부하 발생시에 연성 실시간 태스크가 사용하는 RSS 가 태스크가 수행되는데 큰 영향을 주지 않는 범위 내에서 페이지 아웃 될 수 있도록 보장해야 한다. 본 논문에서는 SRT 클래스에 속하는 프로세스는 항상 일정한 수준 이상의 FRATIO 를 유지시키는 방법을 사용하여 페이지 아웃 되거나 페이지 인 되는 비율의 한계를 정한다. N(page in)과 N(page out)을 각각 페이지 인 개수와 페이지 아웃 개수라 하고, Period 를 프로세스가

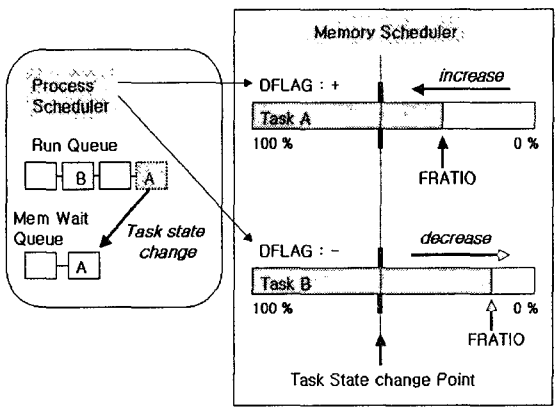
사용자 영역(utime)과 커널 영역에서 수행된 시간(stime)이라고 할 때, FRATIO 는 다음과 같은 식을 따른다.

$$FRATIO = \frac{N(page\ in) + N(page\ out)}{Period}$$

FRATIO 의 값이 커지게 되면, 태스크는 실제 작업을 수행하는 시간보다 새로운 물리 페이지를 할당 받고 해제하는 작업 시간이 더 커지게 된다. 따라서 연성 실시간 태스크의 FRATIO 는 태스크의 제한 시간을 놓치는 횟수가 허용 가능한 범위 내로 한계 값이 정해져야 한다. FRATIO 의 한계 값은 프로세스 스케줄러가 설정하며, 태스크의 제한시간 미스가 발생하는 경우에 한계 값을 낮추어 물리 메모리 할당량을 동적으로 조정한다.

3.2 범용 태스크 메모리 관리

범용 태스크의 메모리 관리는 연성 메모리 과부하 상태와 경성 메모리 과부하 상태로 나누어서 관리를 한다. 연성 메모리 과부하 상태는 현재 수행중인 모든 범용 태스크가 모두 수행 가능한 상태이며, 경성 과부하 상태는 일부 태스크를 희생시켜서 나머지 태스크를 수행시켜야 하는 상황이다. 연성 메모리 과부하 상태에서 메모리 관리 방법은 다음 그림 2 와 같다.

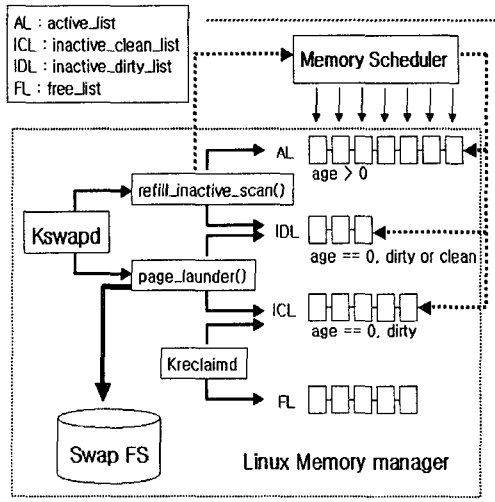


[그림 2] 범용 태스크 메모리 관리

즉 각 태스크는 DFLAG 를 유지하고 있으며, 프로세스 스케줄러가 우선 순위가 낮은 태스크를 희생 태스크로 간주하고 DFLAG 값을 증가 방향으로 설정을 해주고 우선 순위가 높은 태스크의 DFLAG 값을 감소 방향으로 설정을 한다. 따라서 DFLAG 의 방향에 따라서 FRATIO 값이 증가하거나 감소하게 된다. 메모리 스케줄러는 페이지 교체를 해야 하는 시점에서 LRU 정책에 의해서 희생 페이지를 선택할 때, DFLAG 의 값을 참조한다. 만일 선택된 페이지가 DFLAG 값이 증가 방향으로 되어 있는 경우 다른 희생 페이지를 찾고, DFLAG 값이 감소 방향으로 되어 있는 경우는 페이지를 교체한다. 따라서 프로세스의 스케줄링 정책이 메모리 스케줄링 정책에 반영되어 동작 가능하게 되는 것이다. DFLAG 가 필요한 이유는 희생 태스크가 빈번하게 바뀌게 되는 경우에는 범용 태스크들이 실제적인 작업처리를 하는 것보다는 페이징에 소모하는 시간이 더 커지게 되므로 이를 효율적으로 제어할 수 있는 메커니즘이다. 따라서 프로세스 스케줄러가 희생 태스크를 선택하면, 메모리 스케줄러는 DFLAG 를 증가 방향으로 설정하여 희생 태스크의 FRATIO 를 높여주는 상호 협조적인 정책을 사용하고 있다. 경성 메모리 과부하 상태에서는, 희생(victim) 태스크를 선택한 후에 그 태스크가 사용하는 물리 메모리의 일부를 다른 태스크들이 사용하게 허용하는 방법을 사용한다. 즉 일정 기간 동안 희생 프로세스의 메모리를 나머지 BE 클래스 태스크가 이용함으로 전체적인 수행률(throughput)을 향상시키게 하는 것이다.

4. 자원 스케줄러 협조 시스템 구현

프로세스 스케줄러와 메모리 스케줄러가 상호 협조하는 모델을 구현하기 위하여, 본 논문에서는 리눅스 운영체제를 수정하였다. 다음 그림 3 은 리눅스 커널의 Page cache 내부의 개념도를 보이고 있으며, 본 논문에서 제시하는 메모리 스케줄러의 연관 관계를 도식화하고 있다.



[그림 3] 메모리 스케줄러 연관도

리눅스는 메모리 관리를 위해서 크게 4 가지의 리스트를 유지하고 있으며, 각각은 active list, inactive clean list, inactive dirty list 그리고 free list 이다. 리눅스 커널은 일정한 주기마다 커널 쓰레드로 동작하는 kswapd 데몬이 메모리의 사용여부를 감시하고 있으며, 사용 가능한 물리 메모리가 일정 수준 이하로 떨어지는 경우에 여러 가지 정책(page out, swap out 등)을 사용하여 사용 가능한 페이지를 일정한 수준으로 유지시키고 있다. 또한 refill_inactive_scan()을 통해서 리스

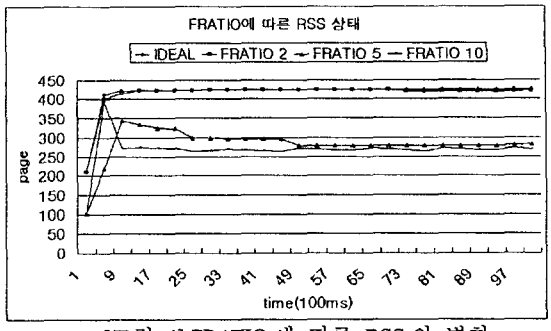
트에 연결된 페이지들의 사용 여부를 주기적으로 조사한다. 만일 페이지에 대한 접근이 없었다면, 페이지마다 가지고 있는 age 변수를 감소시키고, age 변수가 0 가 되는 경우에는 active list 에 있는 페이지를 inactive list 로 이동시킨다. 페이지에 대한 접근이 있는 경우는 age 변수를 초기화 시킨 후에 active list 에 연결시킨다. 따라서 자주 사용되는 페이지는 교체되지 않고 active list 에서 찾을 수 있다. Inactive dirty list 에 있는 페이지들은 page_launder()에 의해서 스왑 영역으로 방출된 후 inactive clean list 로 이동된다. 메모리 스케줄러는 Kswapd 를 수정하여 동작시키고 있으며, active list 에서 페이지 교체가 이루어질 때, 기존의 리눅스에서 제공하고 있는 LRU 페이지 교체 기법을 수정하였다. LRU 페이지 교체 기법은 페이지의 age 값이 0 가 되는 경우에 inactive list 를 거쳐서 디스크에 스왑되거나 또는 inactive clean list 로 보내어지고 마지막으로 free list 로 이동된다. 본 논문에서 제시하는 메모리 스케줄러는 페이지 교체를 하는 부분에서 FRATIO 의 값과 비교하여 페이지 교체를 결정하는 역할을 담당한다.

5. 실험 결과

본 논문의 유용성을 검증하기 위하여 범용 시스템 환경에서 연성 실시간 태스크의 수행이 원활히 진행되는지를 실험하였다. 실험 환경은 커널 2.4 버전이 수행되는 리눅스 운영체제 상에서 다양한 범용 태스크와 멀티미디어 태스크를 혼합하여 멀티미디어 태스크가 제한된 시간에 수행이 완료되는지를 확인하였다.

실험에 사용된 태스크는 mpeg_play, cscope 그리고 calculator 이다. mpeg_play 는 버클리 대학의 멀티미디어 플레이어이며, cscope 는 소스 분석을 위한 툴로써, 소스 데이터베이스를 구성하는 단계에서 메모리의 소모량의 큰 프로그램이다. Calculator 는 수식 계산을 수행하는 Best-effort 태스크로 CPU 집약적인 특성을 가진다.

실험에서 FRATIO 의 값이 연성 실시간 태스크(mpeg_play)의 메모리 사용량에 미치는 영향을 분석하였다. 이를 위하여 mpeg_play, cscope, 그리고 calculator 가 동시에 수행되는 환경에서 mpeg_play 의 RSS 변화 상황을 조사하였다. 그림 4 는 이상적인 환경에서 mpeg_play 가 사용하는 RSS 의 용량을 기준으로 FRATIO 의 값이 각각 2%, 5% 그리고 10 % 일 때 RSS 의 용량이 변화하는 것을 보이고 있다. 실험 결과에서 알 수 있듯이 FRATIO 의 값이 2 퍼센트 정도의 한계값을 유지할 때는 이상적인 상황의 mpeg_play 의 수행 결과와 큰 차이가 나지 않는다. 하지만 페이지 아웃과 페이지 인 작업이 5 퍼센트가 되었을 때는 메모리 상에 존재하는 mpeg_play 물리 페이지는 급격히 저하되고 있음을 알 수 있다. 이때 FRATIO 에 따른 mpeg_play 의 제한 시간 실패 비율은 표 1 에서 볼 수 있듯이 급격한 차이를 보이고 있다. 따라서 FRATIO 의 값이 증가되어 일정한 물리 메모리를 확보하지 못하는 mpeg_play 는 연성 실시간 특성은 보장되지 않고 있음을 알 수 있다.



[그림 4] FRATIO 에 따른 RSS 의 변화

	IDEAL	FRATIO 2	FRATIO 5	FRATIO 10
Deadline miss	0	2	39	43
Average miss time	0	4.3	23.3	24.5

[표 1] FRATIO 에 따른 mpeg_play 의 제한시간 실패 비율

6. 결론

본 논문에서는 멀티미디어 응용과 같은 연성 실시간 태스크가 원활히 수행될 수 있도록 연성 실시간 메모리 관리 기법의 설계 및 구현에 대해서 기술하였다. 연성 실시간 특성을 제공하기 위해서, 태스크의 특성을 메모리에 반영하는 방법을 사용하였으며, 이를 위하여 프로세스 스케줄러와 메모리 스케줄러의 상호 협조 방법이 본 논문의 핵심이 된다. 실험결과에서 알 수 있듯이 각각의 자원에 대해서 개별적으로 동작하는 스케줄링 방법보다는 프로세서와 물리 메모리와 같은 자원 사용에 관한 정보 교환은 연성 실시간 태스크가 제한 시간을 놓치지 않고 수행될 수 있도록 하고 있다.

참고문헌

[1] Steven M. Hand, "Self-Paging in the Nemesis Operating System", Operating Systems Design and Implementation,73-86,1999

[2] Kieran Harty and David R. Cheriton,"Application-controlled physical memory using external page-cache management",ASPLOS volume 27,1992

[3] Dawson R. Engler and M. Frans Kaashoek and James O'Toole,"Exokernel: An Operating System Architecture for Application-Level Resource Management",SOSP 251-266,1995

[4] Jim Mauro and Richard Mc Dougall, SOLARIS Internals, SUN Microsystems, Inc, 2001