제18회 한국정보처리학회 추계학술발표대회 논문집 제9권 제2호 (2002.11)

우선순위에 기반한 메모리 고립화 기법

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

Priority based Memory Isolation Method

Young-Woong Ko*, Cheol-Ho Hong, Young-Pill Kim, Chuck Yoo
Department of Computer Science, Korea University

요 약

대부분의 멀티미디어 응용은 제한된 시간내에 작업이 수행되어야 하는 연성 실시간 특성을 가지고 있으며, 일반적으로 윈도우즈 또는 유닉스와 같은 범용 운영체제 상에서 수행되고 있다. 하지만, 범용 운영체제는 요구 페이징에 기반한 가상 메모리 시스템을 근간으로 하고 있으므로, 연성 실시간 태스크가 요구하는 제약조건을 처리하는데 문제점을 가지고 있다. 본 논문에서는 범용 운영체제가 연성 실시간 태스크를 원활히 지원할 수 있도록 가상 메모리 시스템을 개선시키는 방법을 제시하고 있다. 주요 아이디어는 실시간 태스크가 사용하는 메모리에 대해서 태스크의 우선순위에 기반한 고립화(isolation)를 제공하는 것이며, 이를 통해서 메모리 제약 조건하에서 연성 실시간 태스크가 제한 시간을 만족시키며 수행됨을 보이고 있다.

1. 서론

범용 시스템 환경에서 연성 실시간 태스크인 멀티미디어 응용을 효과적으로 지원하기 위한 연구가 활발히 진행되고 있으며, 특히 프로세스 스케줄링 분야에서 SMART, Raialto, Hierarchical CPU Scheduler, Resource Reservation[1] 등의 연구 결과들이 제시되었다. 그러나 연성 실시간 태스크가 제한시간을 지키면서 수행되기 위해서는 프로세스 스케줄링만으로는 한계가 있다. 왜냐하면 프로세스가 제대로 수행되기 위해서는 프로세서 이외에 물리 메모리 및 디스크 대역폭과 같은 자원이 충분히 확보되어 있어야 하기 때문이다. 따라서 연성 실시간 태스크가 원활하게 수행되기 위해서 태스크가 필요로 하는 자원들에 대하여 예측 가능한 사용을 보장해줄 수 있어야 한다.

본 연구에서는 연성 실시간 태스크가 제한된 시간 안에 수행될 수 있도록 하기 위해서 필요한 메모리 자원을 보장받고 사용할 수 있도록 해주는 고립화(isolation) 방법을 제안하고 있다. 메모리 고립화 방법은 기존에 활발한 연구가 이루어졌던 자원 예약(resource reservation) 방법[1]과는 큰 차이가 존재한다. 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 특성을 가진다. 따라서 어떤 태스크가 프로세서를 할당 받아서 수행하고자 할 때, 프로그램 수행에 필요한 코드, 데이터, 힙(heap) 그리고 스택(stack) 등이 물리 메모리 상에 존재할 수 있도록 관리가 이루어져야 한다. 본 연구에서 제시하는 메모리 고립화 방법은 태스크의 우선순위에 기반하고 있으며, 이를 통해서 다양한 우선순위를 가지고 수행되는 태스크들의 요구조건을 만족시킬 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 가상 메모리 시스템의 문제점을 해결할 수 있는 기존의 연구 내용을 기술하며, 3장에서는 우선순위에 기반을 둔 메모리 고립화 모델에 대한 내용 기술 및 리눅스 시스템에 구현한 우선순위 기반 메모리 고립화 모델에 대해서 설명한다. 4장에서는 실험을 통한 본 논문의 유용성을 보이고 있으며, 5장에서 결론을 내린다.

2. 관련 연구

Nemesis 운영체제에 구현된 self-paging[2] 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 및 L4와 같은 마이크로 커널의 external pager와 접근 방법이 유사하나, 페이저가 독립 서버로 존재하지 않고, 각 응용 프로그램이 개별적으로 페이저를 가지고 있다는 점에서 차이가 난다. Harty의 V++ external page-cache 관리 기법[3]은 각 응용 프로그램이 자신에게 할당된 물리 페이지를 관리한다는 측면에서 self-paging과 유사하며, external pager가 응용 프로그램의 주소 공간에 포함되거나 또는 독립적인 모듈로 존재할 수 있는 방안을 제시하였다.

일반적인 UNIX 시스템에서 제공하는 mlock 시스템 호출[4]은 특정 메모리 영역을 강제적으로 고정(pinning)시킴으로서 해당 영역의 페이지가 교체되어 나가지 않도록 하고 있다. 따라서 제한된 시간 내에 작업을 수행하는 실시간 태스크를 지원할 수 있는 간단하며 효과적인 기법이다. 하지만, 과도한 mlock 시스템 호출은 전체 메모리 관리 효율을 떨어뜨리며, 쓰레싱(thrashing)을 발생시킬 수 있는 문제점을 가지고 있다.

madvice 시스템 호출[4]은 응용 프로그램에서 힌트를 주는 방식으로 가상 메모리를 효율적으로 사용할 수 있게 한다. 즉, madvice 시스템 호출은 응용 프로그램이 특정한 메모리 영역에 대해서 순차 접근을 할 것인지, 임의 접근을 할 것인지 또는 LRU에 기반한 일반적인 페이지 정책을 사용할 것인지에 대한 힌트를 제공하고 있다. 하지만, madvice는 특정한 메모리 영역에 대한 메모리 관리 최적화 기법이며, 사용할 수 있는 정책이 제한적이다.

3. 우선순위 기반 메모리 고립화

본 논문에서 제시하는 우선순위에 기반을 둔 메모리 고립화 모델은 연성 실시간 태스크와 범용 태스크가 공존하는 환경에서 연성 실시간 태스크의 제한 시간을 만족시키고 수행될 수 있는 것을 목적으로 하고 있다. 즉 메모리 과부하 발생하더라도, 연성 실시간 태스크가 제한된 시간 안에 작업을 끝마치기 위해서 필요한 물리 메모리가 보장되어야 한다. 이를 위해서 메모리 과부하 발생시에 연성 실시간 태스크가 사용하는 RSS가 페이징에 의해서 교체되어 나가는 것을 막아주는 방법이 필요하다.

본 논문에서 메모리 고립화를 위한 설계 원칙은 크게 세 가지이다. 첫 번째는 기존의 가상 메모리 시스템의 구조에 크게 벗어나지 않게 함으로써, 다양한 시스템에 쉽게 포팅(porting)할 수 있게 한다. 두 번째는 런타임 상황에 유동적으로 대처할 수 있는 구조를 취한다. mlock과 같이 프로그램이 구현된 이후에 무조건적인 메모리 락킹을 통한 메모리 자원의 확보는 시스템에 치명적인 결점을 줄 수 있으므로, 메모리 자원의 현재 상황을 모니터링하고 가용한 범위 내에서 메모리 자원을 할당해줄 수 있어야 한다. 세 번째는 시스템에 최소한의 오버헤드를 줄 수 있도록 설계한다.

지금까지 메모리 고립화 방법은 페이지의 속성에 따른 전역적인 페이지 교체 방법이나 특정 페이지를 락킹시키는 방법이 대부분이라고 할 수 있다. 본 연구는 기존의 알고리즘은 그대로 두고, 페이지 교체에 몇 가지 정책을 추가하는 방법을 사용함으로써, 메모리 고립화를 제공하고 이를 통하여 가상 메모리 시스템에 실시간 특성을 부여하려고 한다.

우선순위 기반 메모리 고립화 모델은 메모리 사용 현황에 대한 시스템 정보를 이용하여, 각 태스크가 보유할 수 있는 예상 RSS 값을 계속해서 조정하는 방법을 취한다. 따라서 페이지 교체를 수행하는 모듈에서 특정 태스크의 RSS의 값이 태스크의 예상 RSS 값을 유지할 수 있도록 조정을 하게 되며, 이때, 현재 태스크의 RSS 값이 예상 RSS 값보다 큰 경우에만 페이지 교체가 이루어지게 하고, 그렇지 않은 경우는 다른 페이지를 교체 대상으로 선정하게 된다. 태스크의 예상 RSS 보유량은 태스크의 우선순위, 페이지 폴트의 비율, 태스크가 수행되는 동안 할당 받았던 최대 RSS의 값, 그리고 현재 메모리 상태 정보를 통해서 계산한다.

다음 그림은 본 논문에서 사용하는 용어를 기술하고 있다.

task.exp_rss : 태스크의 예상 RSS 값
task.cur_rss : 태스크의 현재 RSS 값
task.max_rss : 태스크가 수행되는 동안 받은 최대 RSS 값
task.priority : 태스크의 우선순위
task.pagefault : 태스크가 받은 페이지 폴트 횟수
system.mem_status : 시스템의 메모리 가용 상태

[그림 1] 우선순위 기반 메모리 고립화 모델 용어

task.exp_rss는 태스크의 RSS 예상 값을 보관하고 있으며, 메모리 상태에 따라서 값이 증가되거나 또는 감소될 수 있다. 이때, 메모리 상태를 나타내는 system.mem_status는 메모리가 여유가 있는지 또는 부족한지를 나타내주는 값이다. 대부분의 운영체제에서는 시스템 전역 변수로 메모리 상태에 대한 정보를 기록하고 있다. task.cur_rss는 현재 RSS의 값을 보관하고 있는 변수이며, task.max_rss는 태스크가 수행된 이후에, 가장 많은 물리 메모리 용량을 차지했을 때의 정보를 담고 있다. task.priority는 태스크의 우선순위를 나타내며, 메모리 과부하 상황에서 태스크의 RSS 용량을 고립화하는데 사용하는 값이다. 본 논문에서 우선순위 값은 0보다 크고 1보다 작은 실수 값을 사용하고 있다. task.pagefault는 단위 시간에 태스크가 받은 페이지 폴트 횟수를 의미하며, 태스크가 페이지 폴트를 많이 받게 되는 경우,

워킹 세트(working set)에서 벗어나 있다고 판단을 하게 되며, 이때는 RSS의 값을 증가시키려고 시도하게 된다.

메모리 과부하시 예상 RSS 값은 최소한의 워킹 세트를 유지할 수 있도록 감소가 되어야 한다. 이때, 태스크의 우선순위에 의해서 고정적으로 주어지는 예상 RSS 값과 시스템의 상황을 피드백 받아서 예상 RSS 값을 주는 두 가지 방식을 사용한다. 식 (1)과 (2)에서 x, y 변수는 시스템의 특성에 맞게 결정되는 파라미터이며, 어느 정도로 RSS 값을 고립화 시킬 것인지 결정하는 가중치로 사용된다.

첫 번째의 경우에 해당되는 조건은 다음과 같은 식 (1)이 된다.

task.exp_rss = x * (task.max_rss * task.priority) ---------- (1)

즉, 예상 RSS값을 결정함에 있어서, 태스크의 최대 RSS 값과 1보다 작은 태스크 우선순위를 곱한 후에, 가중치 x를 곱하게 되면, 태스크가 메모리에 차지할 수 있는 예상 RSS 값이 도출된다. 수식을 통해서 태스크의 우선순위에 비례하는 RSS 값을 유지할 수 있게 된다.

두 번째의 경우는 다음과 같은 식 (2)가 된다.

task.exp_rss = task.cur_rss + y * task.priority * task.pagefault) ----------(2)

즉, 태스크가 워킹 세트를 벗어나서 지속적인 페이지 폴트가 발생하는 경우가 된다. 이때, 페이지 폴트를 줄이기 위해서는 RSS의 값을 증가시켜야 하며, 그 기준으로는 task.priority 값과 페이지 폴트가 발생하는 비율 그리고 가중치 y를 곱한 후에, task.cur_rss에 더한다. 이와 같은 계산이 필요한 이유는 동일한 페이지 폴트가 발생하여도, 우선순위가 높은 태스크가 빠른 시간 안에 워킹 세트를 복구할 수 있게 하기 위함이다.

다음은 메모리 과부하가 발생한 경우에, 예상 RSS 값을 구하는 식을 정리해서 보이고 있다.

task.exp_rss = MIN(x * (task.max_rss * task.priority), (task.cur_rss + y * task.priority * task.pagefault)) ----------(3)

4. 실험 결과

본 논문에서 제시하는 우선순위 기반 메모리 기법의 유용성을 검증하기 리눅스 시스템 상에서 실험을 진행 하였다. 실험에 사용된 운영체제는 리눅스 커널 버전 2.4이고 하드웨어 사양은 Intel Pentium 500MHz, 64M RAM이다. 실험의 목적은 리눅스 운영체제 상에서 다양한 범용 태스크와 멀티미디어 태스크를 혼합하여 수행시키면서, 멀티미디어 태스크가 제한된 시간에 수행이 완료되는지를 확인하는 것이다. 실험에 사용된 태스크는 mpeg_play, cscope gcc 그리고 fault_make이다.

◇mpeg_play : 버클리 대학에서 개발한 MPEG-1 디코더이며, 주어진 비디오 데이터를 일정한 주기마다 한번씩 디코딩하는 연성 실시간 태스크이다. mpeg_play에서 사용된 비디오 데이터는 비틀즈 뮤직 비디오 파일이며, 시작부분에서 1000 프레임까지 대략 33초간 수행된다.

◇fault_make : 자체적으로 작성한 프로그램이며, 메모리 과부하를 제공해주는 목적으로 사용한다. fault_make는 주어진 실험환경에서 0.5 퍼센트의 프로세서 부하를 발생시키며, 10 MByte의 메모리 공간에 대해서 랜덤(random)하게 read/write를 한다.

◇cscope : 소스 분석을 위한 툴로써, 소스에서 발견되는 심볼(symbol)들 즉, 변수, 함수 그리고 문자열에 대해서 상호 참조할 수 있는 소스 데이터베이스를 구성함으로, 쉽게 소스 코드를 분석할 수 있게 도와주는 프로그램이다.

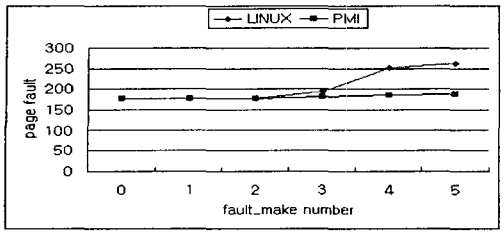
◇gcc : mpeg_play 소스 코드를 컴파일 한다. 주로 프로세서를 활용하는 작업이 대부분이므로 CPU 집중적인(intensive) 특성을 가지고 있으며, 많은 물리 메모리를 필요로 한다.

다음 [그림 2]는 실험에 사용된 프로그램의 특성을 보이고 있다. 여기서 user time, system time, elapsed time은 각각 사용자 영역과 커널 영역에서 mpeg_play가 수행한 시간과 전체 수행 시간을 의미하며, 단위는 초(second)이다. 그리고 cpu load, page fault, RSS(Resident Set Size)는 각각 프로세서 부하(단위: 퍼센트), 페이지 폴트 개수, 메모리에 존재하는 물리 프레임의 개수를 의미한다.

	user time	system time	elapsed time	cpu load	page fault	RSS
mpeg_play	3.35	0.03	33.19	10.1	178	1488
cscope	29.6	2.38	69.7	45.8	180	1612
gcc	28.32	1.91	31	97.3	28250	14420
fault_make	n/a	n/a	n/a	0.5	n/a	10000

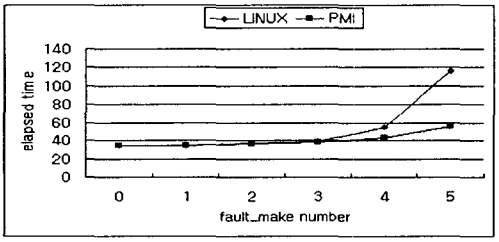
[그림 2] 실험 프로그램의 특성

제안한 우선순위 기반 메모리 고립화 기법의 유용성을 보이기 위해서 mpeg_play의 메모리 고립화를 측정하였다. 이를 위해서 mpeg_play의 우선순위를 1.0으로 할당하고 나머지 gcc, cscope의 우선순위는 0.8로 고정적으로 할당 하였다. 첫 번째 실험은 페이지 폴트의 발생 횟수를 비교하며, 기존의 리눅스 시스템을 LINUX로 우선순위 기반 메모리 고립화 기법은 PMI로 표기하였다.



[그림 3] 페이지폴트 횟수 비교

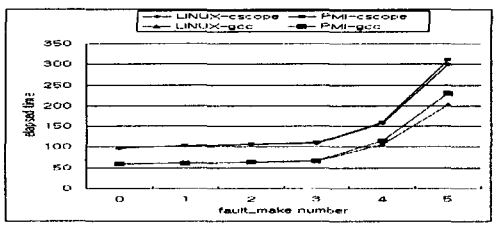
[그림 3]에서 알 수 있듯이, mpeg_play의 우선순위를 높여줌으로써, mpeg_play 태스크의 페이지 폴트 횟수가 일정한 범위 이하로 유지할 수 있다. 실험 결과에서 메모리 과부하가 발생하더라도 mpeg_play는 큰 영향을 받지 않고, RSS가 일정한 값을 유지하는 것을 확인할 수 있었다.



[그림 4] 수행 완료 시간 비교

[그림 4]에서 mpeg_play의 수행 완료 시간을 비교하였다. 실험 결과에서 알 수 있듯이, 메모리 고립화를 통해서 페이지 폴트 횟수가 줄어들고, RSS의 값이 일정하게 유지된 결과, mpeg_play의 수행 완료 시간이 완만한 증가를 보이고 있다. 이때, mpeg_play의 제한 시간 실패 횟수가 fault_make의 값이 5일때, 기존의 리눅스는 1000 프레임 중에서 540 프레임이 실패를 한 것에 비해, 우선순위 기반 메모리 고립화 기법에서는 87 프레임 정도가 실패한 것으로 나타났다.

다음 [그림 5]은 우선순위 고립화를 통해서 우선순위가 떨어지는 gcc, cscope의 수행 완료 시간이 어떻게 변화되었는지 측정해 보았다.



[그림 5] gcc, cscope의 영향 분석

[그림 5]에서 mpeg_play의 우선순위를 높여준 결과 gcc와 cscope 프로그램의 경우, 기존의 리눅스에 비해서 수행 완료시간이 약간 늘어난 것을 확인할 수 있다. 즉 mpeg_play의 메모리 고립화에 의한 연성 실시간 태스크의 메모리 사용량 확보는 결과적으로 다른 태스크의 수행에 영향을 미치게 되는 것이다. 하지만, 본 논문에서 주장하는 것은 모든 태스크의 성능 향상이 아니며, 연성 실시간 태스크의 예측 가능한 수행을 보장하려는 것이기 때문에, 당연한 결과로 받아들여진다.

5. 결론 및 향후 계획

본 논문에서는 메모리 제약 조건하에서 멀티미디어 응용과 같은 연성 실시간 태스크가 제한된 시간 내에 수행될 수 있는 우선순위 기반 메모리 고립화 기법을 제시하였다. 우선순위 기반 메모리 고립화 기법은 태스크의 실시간 우선순위에 따라서 메모리 상에 존재하는 RSS의 값을 보장시켜주는 방법이며, 이를 통하여 연성 실시간 태스크의 페이지 폴트 횟수를 감소시키고, 제한 시간을 잘 지켜주는 것을 목표로 하고 있다. 본 논문에서는 메모리 고립화 기법을 널리 사용되고 있는 리눅스 시스템에 구현하고, 성능 분석을 통해서 유용성을 입증하고 있다. 실험 결과에서 알 수 있듯이, 메모리 고립화를 통해서 연성 실시간 태스크가 제한된 시간 내에 작업이 수행되는 것을 확인하였다. 또한 메모리 고립화가 다른 태스크에게 미치는 영향을 분석하였다.

향후, 본 연구 내용을 한층 발전시켜서, 메모리 관리 기법과 프로세스 스케줄러와 상호 연동한 자원 관리 시스템에 대한 연구를 진행할 계획이다. 이러한 연구 결과는 개별적인 자원 관리를 통합화함으로써, 시스템의 전체 성능 향상과 연성 실시간 태스크의 예측 가능한 자원 확보에 도움을 줄 수 있을 것이다.

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

[3] Kieran Harty and David R. Cheriton, "Application-controlled physical memory using external page-cache management." SIGOPS Operating Systems Review Special Issue, volume 26, page 187, Boston, MA, October 12-15 1992

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