리눅스 기반의 연성 실시간 시스템을 위한 메모리 대체 기법

서의성[0] 오승택 이준원
한국과학기술원 전자전산학과 전산학전공
{ses, stoh, joon}@camars.kaist.ac.kr

Memory Replacement Scheme for Linux-based Soft Real-time System

Eui-Seong Seo[0] Seung-Taek Oh Joonwon Lee
Div. of Computer Science, Dept. of EECS, KAIST

요 약

Linux는 페이지 기반의 가상 메모리 시스템이다. 따라서 메모리가 부족할 때에는 페이지 대치 알고리즘(page replacement algorithm)에 의해 선택된 페이지가 하드디스크로 대치되게 된다. 실시간 시스템에서 이와 같은 페이지 대치가 발생하면 실시간 제약조건을 만족하지 못할 가능성이 크므로 실시간 시스템에서는 이에 맞는 대치 알고리즘이 개발되어야 한다. 본 논문에서는 연성 실시간 시스템에 적합한 N-Chance 기법을 이용한 새로운 페이지 대치 알고리즘을 제안하고 성능을 평가하였다. 새로운 페이지 대치 알고리즘은 기존의 Linux에서 사용하는 second chance 알고리즘을 수정한 것이다. 기존의 알고리즘은 페이지를 대치함에 있어서 사용되지 않는 페이지에 2번의 기회를 준 후 하드디스크로 쫓아내는 방법인데 반하여 본 논문에서 제안하는 방법은 페이지를 사용하는 프로세스가 실시간 프로세스인지 아닌지에 따라서 기회를 주는 횟수를 달리하는 방법이다. N-chance 알고리즘을 사용했을 경우 실시간 제약 조건을 비교적 충족시키면서도 무조건적인 lock으로 인한 메모리 사용의 부담을 줄일 수 있다.

1. 서론

Linux는 페이지 기반의 가상 메모리 시스템이다. 따라서 메모리가 부족할 때에는 페이지 대치 알고리즘(page replacement algorithm)에 의해 선택된 페이지가 하드디스크로 대치되게 된다. 그러나 실시간 작업의 페이지가 이와 같이 하드디스크로 대치되게 된다면 실시간 요구조건을 만족시킬 수 없음은 명백하다. 이와 같은 문제를 방지하고 최대한의 시스템 성능을 제공하도록 메모리 관리 모듈을 설계하여야 한다.

기존의 가상 메모리 시스템의 대치 알고리즘은 LRU(Least Recently Used)에 근접한 알고리즘을 사용한다. 이 것은 최근에 접근된 메모리 블록이 근시간 안에 다시 접근 확률이 크며, 오랫동안 접근이 안된 메모리 블록을 앞으로도 접근되지 않을 가능성이 크다는 것에 기반 한다. 이 알고리즘은 일반적인 운영체계에서는 매우 효율적인 것으로 알려져 있다[1]. 그러나 이 방법은 실시간에 관련된 요소가 전혀 고려되어 있지 않았기 때문에 실시간 운영체계에 사용될 때는 문제가 발생하게 되며, 그 결과로 실시간 프로그램의 메모리 페이지가 하드디스크에 대치되어 프로그램의 시간을 맞추어 주지 못하는 경우가 발생한다.

이를 보완하기 위해 기존의 실시간 운영체계에서는 대치되어서는 안 되는 메모리 페이지에 대하여 메모리 locking 방법을 사용하고 있다. 이 방법은 경성 실시간(hard real-time) 시스템에서는 좋은 성능을 보이나, 인터넷 서버와 주문형 비디오 서버와 같은 연성 실시간(soft real-time) 시스템에서는 모든 메모리를 locking시켜 놓는 것이 매우 큰 부담으로 작용한다. 따라서 본 과제에서는 연성 실시간 시스템에 적합한 우선 순위(priority)에 기반한 n-chance 메모리 대치 알고리즘을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 Linux를 연성 실시간 운영체제로 사용할 때 발생하는 메모리 대치의 문제점을 설명하고 3장에서는 이를 극

*본 연구는 정보통신부 대학 정보통신 연구센터 육성 지원 사업의 지원으로 이루어졌음.

복한 N-chance 알고리즘에 대하여 기술한다. 4장에서는 N-chance 알고리즘의 구현에 대하여 기술하며, 5장에서는 구현된 N-chance 알고리즘의 성능을 평가하고, 마지막으로 6장에서 향후 과제를 기술하고 결론을 맺는다.

2. 연성 실시간 시스템으로서의 Linux의 문제점

특정 메모리에 대하여 locking을 걸어 놓는 방법을 사용하면 실시간 프로그램에서 사용되는 중요한 메모리가 원하지 않는 상황에서 하드디스크로 대치되어 프로그램의 시간을 맞추지 못하는 문제를 해결할 수 있다. 메모리 locking에 대해서는 실시간에 관한 표준인 POSIX1.b에서 필요한 API(mlock(), munlock(), mlockall(), munlockall() 등)와 각 API의 성질들에 관해 정의되어 있다[2]. LINUX에서도 실시간 프로그램을 지원하기 위해서 1.3.43 이후의 버전에서는 메모리 locking이 구현되어 있다[3].

그러나 메모리 locking만으로는 실시간 운영체계가 효율적으로 동작할 수 없다. 그 이유는 다음과 같다. 첫째로 메모리 locking은 아주 많은 비용이 드는 작업으로서 많은 양의 메모리에 대하여 locking하는 것은 전체 성능의 저하를 가져올 수 있다. 만약에 실시간 프로그램의 전체를 locking한다면 이는 다른 프로세스가 사용할 수 있는 메모리가 줄어들어 전체 시스템의 성능에 매우 큰 저하를 가져올 수 있다. 또한 그 반대로 실시간 프로그램에서 중요한 일부만 locking하는 방법이 있을 수 있는데, 이 경우 locking하지 않은 메모리는 실시간성을 보장 받을 수 있을 뿐만 아니라 사용자가 어느 부분을 locking해야 할지 정해주는 번거로움이 있다.

3. 연성 실시간 시스템을 위한 N-Chance 알고리즘

Linux에서는 second chance 알고리즘의 일종인 clock 알고리즘을 이용하여 메모리 대치를 한다. 현재 사용중인 모든 페이지 블록을 시계의 눈금에 놓고 시계 바늘을 돌려 가는 것과 비슷한 방법으로 동작한다[그림 1]. 각 블록은 accessed bit라는 것을 가지고 있는데, 이 bit는 페이지 block이 사용될 때마다 1로 세팅 되어져서 최근에 사용되었다는 것을 알게 한다. 메모리가 부족하여 대치가 일어날 때는 시계 바늘이 돌면서 대치할 페이지를 찾는다. Accessed bit가 1인 페이지를 만나면 최근에 사용되었다는 것을 알므로 대치하지 않고 accessed bit을 0으로만 만든다. Accessed bit가 0인 페이지를 만나면 시계가 한 바퀴 돌 동안에 사용되지 않은 페이지므로 하드 디스크로 대치시킨다.

이와 같은 방법은 실시간 프로세스와 일반 프로세스에 동일하게 적용된다. 하지만 실시간 프로세스에게 좀 더 높은 우선순위를 주어서 실시간 프로세스가 사용하

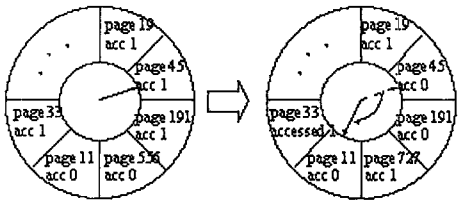


그림 1. 기존의 clock 알고리즘

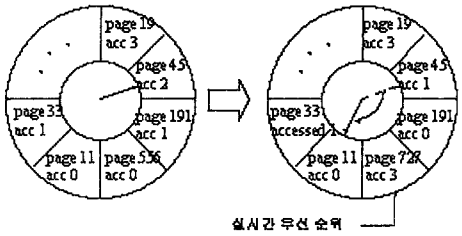


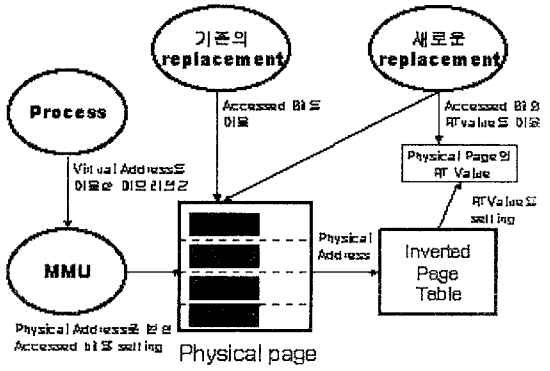
그림 2. 새로운 n-chance 알고리즘

는 메모리를 좀 더 늦게 대치시킬 수 있다면 좀 더 효율적인 연성 실시간 시스템을 구현할 수 있다. 이 것은 각 페이지가 사용되었을 때 변화하는 accessed 값을 다르게 줌으로써 가능하다[그림 2]. 이런 새로운 알고리즘은 실시간 프로세스가 사용하는 메모리를 되도록 늦게 대치함으로써 실시간 요건을 충족시킬 가능성을 높여주고, 메모리 locking에 대한 부담을 줄여주며, 비 실시간 프로그램에 대한 지나친 메모리 대치를 막아줌으로써 시스템의 지나친 성능 저하를 방지한다.

4. N-Chance 알고리즘의 구현

페이지 테이블 entry의 structure(struct pte_t)에 있는 ACCESSED bit은 user 프로세서나 커널이 해당 되는 페이지를 참조하면 CPU의 MMU부분이 하드웨어 상에 구현된 프로그램에 의해서 자동으로 1로 만들어 주거나, 커널의 페이지 테이블 entry의 fault로 인해서 구동하게 되는 부분에 의해서 1로 바뀐다. 이 ACCESSED bit을 확장해서 32bit의 ACCESSED byte를 만들고 실시간 특성이 요구 되는 프로세스들이 메모리를 사용했을 때는 그 정도에 비례해서 큰 값을 주게 하였다. 따라서 second chance가 아닌 가변적으로 변하는 n-chance 알고리즘으로 메모리를 대치한다.

RISC CPU의 경우는 대부분의 ACCESSED bit 조작이 MMU의 지원이 불가능하므로 계속된 페이지 테이블 entry fault를 통해서 커널이 수정을 하게 되지만, x86에서는 MMU가 이 기능을 완벽하게 지원하고 있다. 따라서, MMU의 지원을 포기하고 커널에서 ACCESSED bit을 수정하려 하는 것은 상당한 효율의 감소가 생기게 된다. 결국 페이지 테이블 entry의 fault로 인한 트랩 상태에서 swap out 되는 희생양을 고르기 전에 ACCESSED bit을 확인해서, 사용된 경우는 ACCESSED



byte에 process가 갖는 RT 특성 값인 rt-value를 세팅하게 해주어, 하드웨어 지원으로 인한 효율 증가에 해를 끼치지 않고, 같은 효과를 얻을 수 있게 하였다.

물리적 메모리의 마지막 부분에 inverted 페이지 테이블을 만들었다. Inverted 페이지 테이블은 가상 주소를 이용해 물리적 주소를 찾는 페이지 테이블 방식의 반대 과정으로 작동하는 구조로서, 물리적 주소에 해당하는 가상 페이지의 정보를 찾을 수 있도록 구현되었다. 여기서는, ipt_val과 ipt_org라는 두 개의 배열이 각각 현재의 ACCESSED byte의 값과 메모리가 사용되었을 때 바뀔 ACCESSED byte의 값을 저장하고 있다.

각 task는 메모리의 우선순위를 나타내는 rtval이라는 값을 가지고 있으며, 이 task의 요청에 의해 가상 메모리가 물리적 페이지로 적재 될 때 rtval이 ipt_org의 해당되는 위치에 복사 된다. set_rtval()과 get_rtval() 이라는 시스템 콜의 구현하여 rtval을 조작할 수 있도록 하였다.

ipt_val은 페이지 테이블 entry fault가 났을 경우 second chance 알고리즘이 수행되는 부분에서 ACCESSED bit 대신 검사 된다. N-chance 알고리즘에서는 ACCESSED bit이 아닌 ipt_val의 값을 감소시키면서 0이 되는 것을 swap-out의 희생양으로 삼게 된다. 따라서, rtval의 값을 조정하는 것으로서 그 process가 swap-out의 희생양이 될 가능성을 줄일 수 있다.

5. 성능 평가

N-chance 알고리즘의 성능을 평가하기 위해서 P-III 700MHz, 128MB의 메모리를 갖는 시스템에 Linux 2.2 kernel을 이용하여 실험하였다.

실험을 하기 위해 2종류의 프로세서를 만들었다.

Process A : 15000개의 페이지를 할당한 후에, 3초의 간격으로 10000개의 페이지를 무작위로 access한다(dead-line이 3초인 연성 실시간 프로세스임.)

Process B : 4600개의 페이지를 할당한 후에, 4600개 페이지를 순차적으로 접근하는 것을 500회 반복하는 것을 1초 간격으로 수행한다. (일반 프로세스로 가정. 3개의 동일한 프로세스를 동시에 실행.)

각각의 페이지의 크기는 4Kbyte이다.

	RT process	Other process
second-chance	5.36	0.07
n-chance	2.54	0.08
mlocked	0.01	1.38(1 panic)

표 1. 각 프로세스가 1회 실행되는 평균 시간

실행결과는 표 1.과 같다. 첫번째 실험은 아무런 방법도 쓰지 않고 RT 프로세스와 일반 프로세스 모두 second chance 방법을 적용해 메모리를 대치한 것이고, 두 번째 방법은 RT 프로세스에 n-chance 알고리즘(n=2)을 적용한 것이며, 세 번째 방법은 RT 프로세스에서 필요한 메모리를 mlocked()를 이용하여 locking 시켜서 대치되지 못하도록 한 것이다.

Second-chance 알고리즘을 사용했을 경우는 많은 RT 프로세스가 dead-line을 지키지 못한 것을 알 수 있으나 n-chance 알고리즘은 대부분의 RT 프로세스가 dead-line을 지킬 수 있도록 해 주었다. 또한 mlocked를 이용하면 RT 프로세스의 결과는 매우 좋았지만 그로 인해 다른 프로세스가 거의 실행되지 못하며, 3개의 other process중 하나는 실행이 멈추는 일이 발생하였다.

6. 결론

본 논문에서는 연성 실시간 시스템에 적합한 메모리 대치 알고리즘인 n-chance 알고리즘을 구현하고 성능을 평가하였다. N-chance 알고리즘은 second-chance 알고리즘의 변형으로 페이지가 물리적 메모리에서 대치될 때, 실시간을 요구하는 프로세스가 사용하는 메모리에 대하여는 n번의 기회를 준 다음 대치시키는 알고리즘이다. N-chance 알고리즘을 사용하면, 실시간 프로세스의 성능은 향상되어 dead-line을 지킬 수 있으면서도 메모리를 과다하게 사용하여 다른 프로세스가 동작하지 못하게 되는 현상을 방지할 수 있음을 확인하였다.

7. 참고 문헌

[1] John L. Hennessy and David A Patterson, Computer Architecture A Quantitative Approach second ed. Morgan Kaufmann Publishers Inc. 1996, pp. 373-427

[2] Bill O. Gallmeister, POSIX.4 -- Programming for the Real World, O'Reilly & Associates, 1995

[3] A Vision for Linux 2.2 -- POSIX.1b Compatibility and Real-Time Support, ftp://ftp.informatik.uni-erlangen.de/local/cip/mskuhn/misc/linux- posix.1b