지연 특성이 없는 스토리지를 위한 선택적 문맥 교환

이광호
한양대학교 전자컴퓨터통신학과
e-mail: kanlee@ece.hanyang.ac.kr

The selective context switch for the storage which has no latency

Kwangho Lee
Dept. of Electronics & Computer Engineering, Hanyang University

요 약

본 논문에서는 기존 하드 디스크에만 최적화 되어있는 운영체제의 블록 서브시스템을 수정하여 하드 디스크와 지연 특성이 없는 플래시 메모리나 비휘발성 메모리가 동시에 사용 되었을 때에도 그 특성에 맞게 최적화하여 일반적으로 큰 비용이 수반되는 문맥 교환을 선택적으로 구현함으로써 성능 향상을 보였다.

▸ Keyword : 문맥 교환(context switch), 하드디스크(HDD), 블록 서브시스템(Block Subsystem)

I. 서 론

현재까지 스토리지로 사용되고 있는 하드 디스크는 속도와 용량, 가격 면에서 최고로 우수한 장점을 가지고 있어 십 수 년간 널리 사용되고 있으며 많은 운영체제들이 하드 디스크를 효율적으로 사용하고 보다 좋은 성능을 낼 수 있도록 최적화 되어 있다. 그러나 하드 디스크는 근본적으로 기계 장치를 이용하여 정보를 저장하거나 읽기 때문에 기계적 특성으로 발생되는 임의 접근에 대한 입출력 지연을 제거 할 수 없는 문제를 안고 있다. 따라서 최근 스토리지는 이러한 문제점을 극복하기 위하여 플래시 메모리, SSD, 비휘발성 메모리 등으로 지연 특성이 없고 하드 디스크의 속도와 용량, 가격에 경쟁할 수 있는 새로운 스토리지 개발에 많은 노력을 하고 있다.

하드디스크는 자료가 저장된 플래터를 헤드로 읽는 기계적 특성으로 인하여 탐색 시간과 회전 지연 시간이 존재하는데 이는 최대 20ms에 이르며 이는 현대의 고속 전송에 매우 큰 악영향을 미치게 된다. 따라서 운영체제는 이 악영향을 최소화하기 위하여 임의로 접근되는 읽기와 쓰기의 요청을 모아서 지연을 최소화 할 수 있도록, 모아두었던 읽기와 쓰기 요청을 알맞은 순간에 한꺼번에 수행하도록 한다. 이 때 현재 읽기와 쓰기를 요청한 작업의 실행을 해당 요청이 수행되어 완료될 때까지 연기하고 이 동안 수행할 수 있는 다른 작업을 골라 실행하게 되는데 이는 지연 시간 동안 단순히 대기함으로써 시스템 자원을 낭비하는 것이 아니라 나중에 처리해야 할 작업을 그 동안 실행함으로써 지연 시간이 긴 장치의 사용에 있어 자원 낭비를 완화 시켜준다.

그러나 다른 작업으로 전환할 때는 문맥 교환이라는 과정을 거쳐야 하는데 각 작업마다 필요한 정보들을 추후 다시 사용하기 위하여 다른 곳에 저장하고 다음에 실행될 작업에 필요한 정보들을 다시 읽어드리는 제반 사항이 필요하게 된다. 문맥 교환도 그에 해당하는 비용을 지불해야 하는데 다음에 실행해야 할 작업을 선택해야 하고 문맥을 저장하고 복구해야 하며 현재 작업이 실행되고 있는 pipeline을 비우고 TLB와 page table도 새로운 것으로 교체해야 한다. 이러한 문맥 교환을 위한 직접 비용 이외에도 문맥 교환 이후에 발생하는 캐시 다시 읽어 들이기는 상황에 따

라서 매우 높은 비용을 지불해야 하는 간접비용에 속하며 중앙처리장치와 캐시의 성능이 낮은 내장형시스템의 경우는 그 영향이 커지는 결과를 볼 수 있다.
그렇다면 지연 시간이 없는, 앞서 언급된 플래시메모리, 비휘발성 메모리 등을 사용한다면 이 지연 시간을 무시하고 다른 작업으로 전환하지 않음으로써 문맥 교환의 비용도 소비하지 않을 수 있지 않을까? 그러나 현대의 운영체제는 이에 대한 고려가 되어 있지 않다. 플래시 메모리나 비휘발성 메모리는 하드 디스크와 같이 기계적인 장치를 이용하여 정보를 저장하거나 읽지 않기 때문에 임의 접근에 대해서도 지연 시간이 없이 정보의 저장과 출력이 가능하다. 운영체제는 이러한 각 스토리지의 특성에 맞는, 즉 지연 시간이 매우 긴 장치와 지연 시간이 없는 장치를 모두 고려하여 입출력을 수행하도록 수정할 필요가 있다.
다음 장에서는 관련 연구에 대해서 알아보고 리눅스 운영체제를 사용하여 지연 시간이 없는 스토리지와 지연 시간이 긴 하드 디스크를 동시에 처리하는 입출력 시스템을 어떻게 수정할 수 있는지 살펴본 후 실제 실험을 통하여 불필요한 문맥교환을 제거함으로써 얻어지는 이득에 대하여 보일 것이다.

II. 관련 연구

1. 관련연구

1.1 해외 동향

Ousterhaut[5]는 여러 시스템에서 문맥 교환의 측정으로 하드웨어 속도 증가에 비례하여 전체적인 성능은 증가하지 않는 다는 것을 보이면서 이와 관련된 중요한 하드웨어 요인이 메모리 대역폭과 문맥 교환에 소비되는 비용이라고 언급하였지만 문맥 교환이 왜 비용이 많이 드는지 자세히 언급하지는 않았다. Makoto[6]는 문맥 교환으로 인하여 다른 작업이 기존 작업의 캐시 영역을 대체할 수 있기 때문에 캐시의 지역 참조성이 방해 받고 적중률이 떨어지는 것을 3개의 서로 다른 특성의 작업부하에 대하여 보여주었다. Mogul과 Borg[7]는 문맥 교환 후에 캐시 적중율을 추적하여 캐시 성능이 문맥 교환에 미치는 영향을 실험함으로써 일반적인 문맥 교환의 절차보다 문맥 교환 후 캐시를 다시 채우는 것이 문맥 교환에서 많은 시간을 소비하게 하는 요인임을 보였다. Johan과 Lars[8]은 이러한 캐시 간섭 비용을 내장형 시스템의 실시간 운영체제에서 Simulator와 MiBench를 이용하여 각 캐시 설정에 따른 간섭 비용을 보여주었다. 특히 이 연구는 상이한 메모리 계층 구조에서 내장형 시스템의 동작 형태의 관찰에 의의가 있다. Fang Liu[9]는 뚜렷이 정의가 되지 못했던 문맥 교환의 간접비용에 대해서 여러 가지 요인을 포함하여 설명하였고 이에 대한 수학적 모델도 제시하였다. Lmbench[10]는 문맥 교환의 측정 방법을 제시하였는데 “cache footprint”라는 각 프로세스마다 할당된 메모리 영역을 접근하게 함으로써 프로세스가 접근하는 캐시 영역에 따른 문맥 교환의 비용을 측정 가능하게 하였다.
앞에서 보인 연구들은 문맥 교환이 일어남으로써 발생하는 비용과 그 측정 방법을 보이고 있으나 실제적으로 이를 개선하여 성능을 향상 시킬 수 있는 대안을 제시하고 있지는 않다. 본 연구는 현대의 지연 특성이 없는 스토리지를 사용할 때 블록 입출력에서 발생하는 문맥 교환을 제거함으로써 보다 향상된 성능을 얻을 수 있다는 것을 보여주고 있다.

III. 본 론

1. 리눅스 운영체제의 블록 입출력 구현 방식

앞서 설명한 바와 같이 근래의 운영체제는 긴 시간 동안 보조 기억장치로 사용된 하드디스크에 대해서 최적화 되어 있기 때문에 임의로 접근되는 읽기와 쓰기를 모아서 문맥 교환과 함께 처리한다. 아래 그림 1.은 리눅스에서 그 과정을 간략히 나타낸다. 리눅스의 블록 입출력 서브시스템은 각 읽기, 쓰기 요청을 블록으로 나누고 이를 bio(Block Input Output)의 구조체 단위로 정리하여 엘리베이터 알고리즘에 의해서 근접한 위치로, 다시 말하면 하드디스크의 기계적 동작이 최소로 발생하도록 모으는데 엘리베이터 알고리즘은 각 동작 특성에 따라 Anticipatory나 Deadline 또는 CFQ로 구분된다. 엘리베이터 알고리즘 동작은 __make_request()에서 이루어지는데 __make_request()는 bio의 특성에 따라 엘리베이터 알고리즘을 수행하거나 새롭게 request 구조체를 생성하고 엘리베이터 queue가 비었으면 blk_plug_device()를 수행하여 특정 시간 동안 입출력 요청이 없는 경우 현재의 입출력 요청이 처리 되도록 한다. 이때 bio는 실제적으로 정보의 전송이 이루어지는 메모리의 위치를 가리키고 있다.
request queue 구조체는 각 물리적 장치마다 할당되며 각 장치의 request 자료구조체의 리스트를 포함한다. 앞에서 언급한 엘리베이터 알고리즘 같은 입출력 스케쥴링을 하는 것은 이 request 자료구조체에 바탕을 두고 동작한다. 또한 각 request 자료구조체는 bio들을 관리하는 수단이 되기도 한다. 각 request 자료구조체는 새로운 bio를 위하여 할당되고 입출력 스케쥴에 의해서 기존 bio나 새로운 bio가 합쳐질 때 갱신된다.

이 후 해당 프로세스를 대기 상태로 전환하고 스케쥴러에서 선택된 다른 프로세스로 문맥 교환을 수행하게 된다. 요청된 읽기, 쓰기의 실제적 수행은 다른 읽기, 쓰기의 요청이 적당히 모아진 상태이거나 제한된 시간이 경과하면 generic_unplug_device()를 호출함으로써 수행되는데 이는 현재 프로세스의 문맥이 아닌 커널의 문맥에서 수행되며 완료 후 다시 읽기, 쓰기가 요청된 프로세스로 전환해야 하기 때문에 다수의 문맥교환이 발생하게 된다.

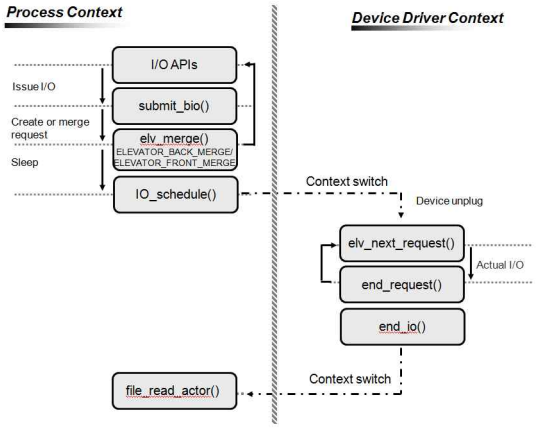

그림 1. 리눅스의 블록 입출력 처리
Fig. 1. Block I/O processing of Linux

위에서 보이는 것처럼 한 번의 읽기, 쓰기를 수행하기 위해서는 최소한 두 번 이상의 문맥 교환이 수행되어야 한다. 하드 디스크의 경우에는 이러한 긴 지연 시간은 중앙처리장치 자원의 효율성을 고려하여 반드시 문맥 교환이 고려되어야 하지만 최근 지연 시간이 없는 장치들은 해당 입출력에 대한 소요 시간이 최소한 두 번의 문맥 교환에 대한 시간보다 적을 경우 현 프로세스에서 문맥 교환 없이 직접 입출력을 하는 것이 더욱 효율적이 된다.

3. 스토리지에 따른 선택적 문맥 교환

최근 개발되고 있는 플래시 메모리나 비휘발성 메모리도 아직 하드 디스크의 속도나 용량, 가격 면에서 모두 우수한 특성을 가지고 있지 않기 때문에 동시에 같이 사용될 가능성이 많다. 따라서 요청된 읽기와 쓰기에 대하여 그 대상이 하드 디스크인지 플래시 메모리나 비휘발성 메모리 같은 지연 특성이 없는 스토리지인지 검출하여 하드 디스크의 경우 기존에 최적화된 절차를 수행하고 지연 특성이 없는 스토리지인 경우 문맥 교환을 하지 않고 바로 요청된 문맥에서 읽기와 쓰기 동작을 수행하도록 선별하여야 한다.

그 선별 작업은 엘리베이터 알고리즘을 구성하고 문맥 교환을 수행하기 전인 __make_request()에서 이루어진다. 우선적으로 각 장치마다 입출력 스케쥴링 정책을 하드 디스크인 경우 기본 값으로 유지 시키고 지연 특성이 없는 스토리지인 경우 근접한 읽기 쓰기 요청으로 재구성되어 처리되지 않도록 장치가 초기화 될 때 NOOP으로 설정을 변경 한다. 또한 장치가 초기화 될 때 request queue의 queue_flags를 지연 특성이 없는 스토리지를 나타낼 수 있도록 flag를 추가하고 __make_request()에서 이를 검출 하도록 한다.

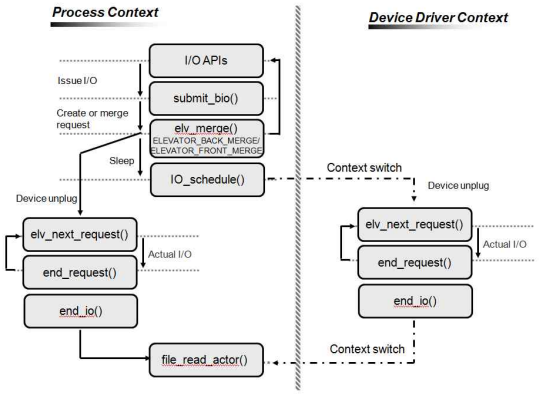

그림 2. 리눅스의 선택적 블록 입출력 처리
Fig. 2. Selective block I/O processing of Linux

4. 실험

4.1 실험 환경

표 1. 시스템 환경
Table 1. System Environment

항목	값
CPU 종류 및 처리 속도	MIPS24KEc 350MHz
캐시 크기	I 캐시: 32KB, D 캐시: 32KB
캐시 종류	4 way set-associative
메모리 크기	256 MB
DDR 속도	1066 Mbps
Flash	OneNAND 1Gbit
Linux kernel	2.6.21

4.2 실험 방법

선택적 문맥 교환을 수행하도록 플래시 메모리 request queue에 FAST_DEVICE flag를 추가한 뒤 장치 초기화에서 입출력 스케쥴링을 NOOP으로 전환하고 __make_request()에서 이를 검출하여 문맥 교환이 이루어지지 않고 요청된 프로세스에서 바로 처리 될 수 있도록 수정 한다. 문맥 교환이 수행되는데 걸리는 시간을 작업 공간내에서 접근하는 정보의 존재 유무로 비교하기 위하여 lmbench[10]의 lat_ctx를 이용하여 각 작업의 개수에 따른 문맥 교환의 소요 시간을 측정하여 문맥 교환이 성능에 미치는 영향을 알아보고 일정 크기(블록 크기)로 임의 읽기와 순차 읽기를 선택적 문맥 교환으로 수정하기 전과 수정한 후에 대해서 측정해보고 수학 계산을 수행하는 10개 프로세스와 입출력만을 수행하는 10개 프로세스를 함께 동작 시킨 후 각 환경에서 측정함으로써 선택적 문맥 교환이 읽기 성능에 미치는 영향을 측정하였다. 마지막으로 부팅 시간에도 어떠한 향상이 있는지 측정 하였다.

4.3 실험 결과

각 프로세스의 개수에 따른 작업 공간내 접근 정보의 유무에 따른 문맥 교환의 소요 시간의 측정 결과는 아래와 같다. 접근 정보는 캐시의 크기를 넘도록 임의로 100KB로 설정하였다. 각 결과 값은 micro second를 나타낸다.

표 2. 프로세스 개수 및 접근 정보에 따른 문맥 교환 시간

Table 2. Context switch time w.r.t. the number of processes and working set

프로세스의 개수	접근 정보 없음(us)	접근 정보 있음(us)
2	6.94	19.90
4	12.63	17.40
6	12.74	20.90
8	13.35	19.27

위의 결과로 문맥 교환 시 접근해야 하는 정보가 있다면 문맥 교환의 소요 시간은 증가하는 것을 알 수 있다. 대부분의 작업들은 수행 중 정보를 접근 및 가공하므로 앞서 설명한 문맥 교환의 간접비용은 대부분의 문맥 교환에서 발생하게 된다.

아래는 수학 계산을 수행하는 10개 프로세스와 입출력만을 수행하는 10개 프로세스를 함께 동작 시킨 후 각 환경에서 임의 읽기와 순차 읽기를 수행하여 선택적 문맥 교환으로 수정하기 전과 수정한 후에 대해서 그 소요 시간을 측정하였다. 페이지 캐시에 적재되어 버퍼링 되는 것을 막기 위해 umount와 mount를 측정 시 포함하였다.

표 3. 각 작업 부하에 따른 읽기 수행 시간

Table 3. The time of reading w.r.t. the workloads

작업부하종류	수정 전(ms)	수정 후(ms)	향상치
단일 임의 읽기	46300	43330	6.41%
단일 순차 읽기	46090	43136	6.41%
수학 임의 읽기	386473	349073	9.68%
수학 순차 읽기	375778	301404	19.79%
입출력 임의읽기	385642	347193	9.97%
입출력 순차읽기	376710	301312	20.01%

선택적 문맥 교환의 수정 전후의 부팅 시간은 아래와 같이 측정 되었으며 약 8%의 성능 향상을 보였다.

표 4. 선택적 문맥 교환에 따른 부팅 시간

Table 4. The booting time of the selective context switch

작업부하종류	수정 전(ms)	수정 후(ms)	향상치
부팅 시간	12156	11187	7.97%

IV. 결 론

기존 하드 디스크에만 최적화 되어있는 운영체제의 블록 서브시스템을 수정하여 지연 특성이 없는 플래시 메모리나 비휘발성 메모리가 동시에 사용이 되었을 때에도 그 이점을 살릴 수 있도록 선택적 문맥 교환을 구현함으로써 성능 향상을 보였다. 하드 디스크의 동시 사용 실험이나 비휘발성 메모리에 대한 실험은 차후에 이루어질 것 이다.

실험 결과는 선택적 문맥 교환의 장점이 확실하게 보이도록 작업부하를 설정하였기에 다른 작업부하로 실험한다면 그 이점이 감소될 여지는 있다. 그러나 다른 작업부하에서 성능 저하를 보이지 않으면서 특정 작업부하에서 성능 향상을 보인다면 이는 시스템 성능을 향상 시킨 것이라고 볼 수 있을 것이다. 특히 부팅 시간이 단축되는 결과는 확실한 시스템의 성능 향상의 요인으로 볼 수 있으며 실적용에도 많은 이점을 얻을 수 있을 것이다.

앞으로 다중 중앙처리장치와 각 캐시 설정에 따른 영향도 고려하여 각 장치의 종류뿐만 아니라 각 읽기, 쓰기 요청의 특성에 따른 적응적 문맥교환을 수행 할 수 있도록 연구할 것이다.

참고문헌

[1] HENNESSY, J., AND PATTERSON, D. Computer Architecture: A Quantitative Approach. Morgan Kaufmann Publishers, Inc, San Francisco, CA, 1996.

[2] Wikipedia, Flash memory, http://en.wikipedia.org/wiki/Flash_memory

[3] Bez, R., Camerlenghi, E., Modelli, A., Visconti, A., Introduction to Flash Memory", Proceedings of the IEEE, 91(4), pp.489~502, 2003

[4] Wikipedia, Non-volatile random access memory http://en.wikipedia.org/wiki/Non-volatile_RAM

[5] OUSTERHOUT, J. K. Why aren't operating systems getting faster as fast as hardware? In Proceedings of the USENIX Summer Conference (Anaheim, CA, June 1990).

[6] MAKOTO KOBAYASHI, An Empirical Study of Task Switching Locality in MVS. In IEEE 1985

[7] MOGUL, J. C., AND BORG, A. The effect of context switches on cache performance. In Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (Santa Clara, CA, Apr. 1991), pp. 75 - 84.

[8] Johan Starner and Lars Asplund, Measuring the Cache Interference Cost, In ACM 2004

[9] Fang Liu at el, Characterizing and modeling the behavior of context switch misses, In ACM 2008

[10] MCVOY, L., AND STAELIN, C. lmbench: Portable tools for performance analysis. In The Proceedings of the USENIX 1996 Annual Technical Conference (San Diego, CA, Jan. 1996), pp. 279 - 294.

[11] Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman, "Linux device drivers", 2005