리눅스 페이지 캐시에 대한 효율적인 관리 기법

이명진 차현욱 이수인 이은규 김명철
한국정보통신대학원대학교
{solma, chahu, elsie, ekyulee, mckim}@icu.ac.kr

An Efficient Management Scheme for Page Cache in Linux

Myung-jin Lee Hyun-uk Cha Soo-in Lee Eunkyu Lee Myungchul Kim
Information and Communications University

요 약

운영체제의 성능을 향상시키기 위해서는 효율적인 메모리 관리가 필수적이다. 시스템에서는 존재하는 크기의 메모리보다 더 큰 크기의 메모리 공간을 요구하는 경우가 빈번히 발생한다. 이 문제를 극복하기 위해서 운영체제에서는 가상 메모리(virtual memory)를 사용한다. 리눅스 가상 메모리 지원 방식에는 요구 페이징(demand paging), 스왑핑(swapping), 공유 가상 메모리(shared virtual memory) 가 있다. 이런 방식을 효율적으로 지원하기 위해서 리눅스는 캐시를 사용한다. 본 논문에서는 페이지 해시 리스트에서 페이지를 찾는 알고리즘을 수정하여 새로운 페이지 캐시 관리 기법을 제안한다. 이 방법은 탐색한 페이지를 해시 리스트의 헤드(head)로 옮김으로써 다음 탐색 때 그 페이지를 찾는데 필요한 탐색 회수를 줄일 수 있다는 장점을 갖는다. 또한 다른 프로세스에 의해서 동시에 많이 참조되는 페이지들은 탐색시간이 줄어들게 된다. 시뮬레이션 프로그램을 통해 본 논문에서 제안한 수정된 페이지 캐시 관리 기법을 이용하면, 기존의 방법에 비해서 페이지를 찾는데 필요한 탐색 회수와 탐색 시간의 측면에서 성능이 향상됨을 보인다.

1. 서론

프로세서의 이용률과 사용자에 대한 응답 속도를 향상시키기 위해서 컴퓨터는 메모리상에 여러 프로세스를 유지해야 한다. 효율적인 메모리 관리를 위하여 다양한 기법들이 제안되어 있으며, 많은 요소들을 고려하여 적절한 메모리 관리 기법이 선택되어 사용되어 진다. 가상 메모리는 시스템이 메모리를 필요로 하는 여러 프로세스에게 메모리를 공유하게 함으로써 실제 메모리가 가지고 있는 것보다 더 많은 메모리를 시스템이 가진 것처럼 보여주는 기법이다.

메모리 관리에서 가상 메모리는 캐시, 요구 페이징, 스왑핑과 같은 부분으로 세분화될 수 있다[3]. 리눅스는 메모리 상에 페이지 캐시를 유지하고 있으며 inode 가 할당된 모든 메모리 페이지들은 해시 리스트에 기록되고, 그것들은 프로세스에 의해서 실행되는 코드 또는 메모리에서 맵핑되는 파일의 내용으로 대응될 수 있다. 이 페이지 캐시는 캐시에 연결되는 페이지의 상태에 따라 동적으로 관리된다. inode가 할당된 페이지의 내용이 메모리로 불려질 때, 새로운 페이지가 할당되어 캐시에 넣어지고, 그 내용은 디스크로부터 읽혀지게 된다. 페이지 내용에 대해 다음에 접근할 때, 이미 메모리상에 있는 페이지를 다시 읽어올 필요가 없게 된다. 이 접근 방식에서는 페이지가 페이지 해시 테이블에 덧붙여질 때 해시 리스트의 테일(tail)에 덧붙여진다. 따라서 순차적 탐색을 하는 리스트의 특징 때문에 테일에 붙은 페이지를 탐색하는데 걸리는 시간이 길어지게 된다.

본 논문에서는 페이지 캐시에서 페이지를 탐색하는 함수를 수정하여 기존 방법의 문제점을 개선하였다. 즉, 한번 탐색된 페이지의 포인터를 해시 리스트의 헤드로 옮김으로써, 다음 탐색 때 그 페이지를 찾는데 걸리는 탐색 시간을 줄일 수 있도록 하였다.

시뮬레이션 프로그램을 통해 본 논문에서 제안한 수정된 페이지 캐시 관리 기법이 기존의 방법에 비해서 페이지를 찾는데 필요한 탐색 회수와 탐색 시간의 측면에서 성능이 향상되었음을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 리눅스의 메모리 관리 기법과 관련하여 캐시에 대해서 살펴 본다. 3장에서는 기존의 페이지 캐시 관리 기법과 본 논문에서 제안하는 페이지 캐시 기법을 소개한다. 4장에서는 기존의 방법과 3장에서 제시한 방법의 성능을 비교, 분석한다. 마지막으로 5장에서는 결론을 맺고 향후 연구에 대해 살펴본다.

2. 관련 연구

이 장에서는 시스템의 성능을 향상시키기 위해서 리눅스에서 사용되고 있는 캐시 관리 기법을 소개한다[6].

캐시의 종류로는 버퍼 캐시(buffer cache), 페이지 캐시(page cache), 스왑 캐시(swap cache), 그리고 하드웨어 캐시(hardware cache)가 있다. 버퍼 캐시는 블록 장치 드라이버가 사용하는 데이터 버퍼들을 가지고 있으며, 이들 버퍼는 고정된 크기로 블록 장치에서 읽거나 쓰는 자료의 블록을 가지고 있다. 페이지 캐시는 디스크상의 이미지와 데이터에 접근하는 속도를 높이기 위해 사용되며, 파일의 논리적인 내용을 페이지 단위로 저장한다. 디스크에서 메모리로 페이지를 읽어 들이면, 페이지들은 페이지 캐시에 저장된다. 스왑 캐시는 페이지가 더티(dirty) 페이지인지

를 구별하여 더티 페이지인 경우 스왑 파일에 저장한다. 이들 페이지가 스왑 파일에 기록된 다음 더 이상 변경되지 않는다면, 그 페이지가 다음에 스왑 아웃(swap out)될 때, 스왑 파일에 기록할 필요 없이 폐기하면 된다. 하드웨어 캐시는 프로세서 내부에 있는 페이지 테이블 엔트리의 캐시로서 주로 구현된다. 이 경우에 프로세서는 항상 페이지 테이블을 읽는 것이 아니라, 페이지 테이블 엔트리를 필요로 할 때마다 페이지에 대한 변환 결과를 참조한다. 이들은 변환 참조 버퍼(Translation Look-aside Buffer, TLB)라고 불리며 시스템의 여러 프로세스의 페이지 테이블 엔트리의 복사본을 가지고 있다[3].

높은 효율을 얻기 위해서 캐시를 관리하는데 많은 시간과 공간을 사용해야 한다는 점이 이들 캐시의 단점으로 지적된다. 본 논문에서는 페이지 캐시 탐색 알고리즘을 수정하여 페이지 캐시를 탐색하는 시간을 줄여 효율을 개선하고자 한다.

3. 제안 알고리즘

이 장에서는 페이지 캐시에 대한 기존의 관리 기법에 대해 살펴 보고, 새로운 페이지 캐시 관리 기법을 제안한다.

3.1. 기존의 페이지 캐시 관리 방법

이 절에서는 리눅스 시스템에서 사용되는 페이지 캐시와 관련된 기본적인 구조와 관리 기법을 소개한다[6].

리눅스에서 페이지 캐시의 역할은 디스크에 있는 파일에 대한 빠른 접근을 가능하게 하는 것이다. 메모리 맵핑된 파일들은 페이지 단위로 읽혀지고, 그림 1과 같이 해시 체이닝(hash chaining) 기법을 이용하여 이들 페이지들을 페이지 캐시에 저장한다.

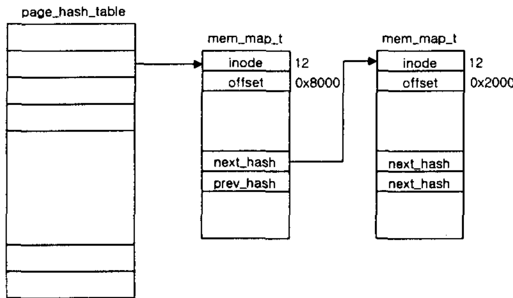

그림 1. 리눅스 페이지 캐시

리눅스에서 각각의 파일은 가상 파일 시스템(Virtual File System, VFS) inode 자료 구조에 의해서 식별된다. 각각의 VFS inode는 유일하고 오직 하나의 파일을 식별하도록 해준다. 페이지 테이블에서의 인덱스(index)는 파일의 VFS inode와 그 파일에서의 오프셋(offset)으로부터 유도된다.

페이지가 요구 페이징에 의해 메모리로 읽혀질 필요가 있고 메모리 맵핑된 파일로부터 읽혀질 때마다 페이지 캐시를 통해서 페이지는 읽혀진다. 만약 페이지가 캐시에 있다면, 그 페이지에 대한 포인터가 페이지 폴트(page fault) 처리 코드로 되돌려진다. 그렇지 않다면 이미지를 가지고 있는 파일 시스템에서 메모리로 페이지를 가져와야 한다. 리눅스는 물리적 페이지를 할당하고 디스크에 있는 파일로부터 페이지를 읽어온다.

아래의 그림 2는 페이지 캐시에 페이지가 존재하는지를 조사하는 역할을 하는 find_page() 함수의 수행 과정을 보여준다. 예를 들어 요구 페이징에 의해서 페이지를 메모리로 가져올 때, 페이지는 페이지 캐시를 통해서 읽혀진다. 페이지가 캐시에 없는 경우 페이지는 디스크에서 읽혀져 메모리로 가져가게 되며, 페이지 캐시 리스트의 테일에 붙여지게 된다. 참조되지 않는 페이지가 캐시에서 제거되어 캐시가 상태가 변하는 경우를 제외하면, 페이지의 위치가 고정되어 있으므로, 리스트의 테일에 있는 페이지를 자주 탐색하는 경우 탐색 시간이 오래 걸리게 된다.

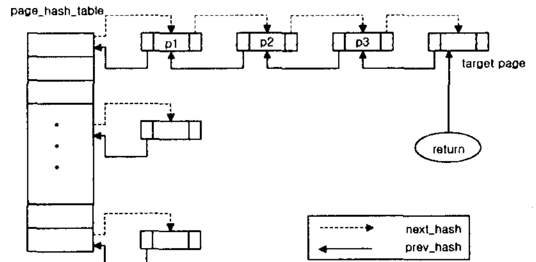

그림 2. 기존의 find_page() 함수

3.2. 수정된 페이지 캐시 관리 방법

본 논문에서는 페이지 캐시에 접근하는데 걸리는 시간을 줄이기 위해서 페이지 탐색 알고리즘을 개선하고자 하였다. 앞 절에서 살펴본 바와 같이, 페이지 캐시에서 사용되는 탐색 알고리즘은 탐색을 순차적으로 수행하고, 탐색된 페이지에 대해서 카운트 필드만을 증가시킨다. 페이지 폴트가 발생하고 페이지 캐시에서 해시 리스트를 탐색할 때, 만약 해당 페이지가 존재한다면, 우리는 그 페이지를 리스트의 헤드로 옮기는 동작을 수행한다. 위에서 제시된 방법을 수행함으로써 페이지 폴트 처리 시간을 줄일 수 있다.

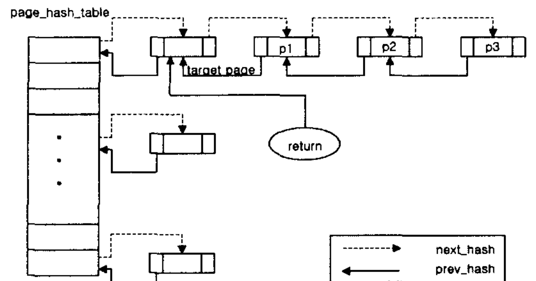

그림 3. 수정된 find_page() 함수

그림 3은 본 논문에서 제시한 페이지 캐시 관리 방법을 나타낸다. 탐색된 페이지는 리스트 상의 위치를 포인터의 값을 해시 리스트의 헤드의 값으로 바꾸어 주고, 바뀐 위치를 되돌려 준다. 따라서 페이지가 자주 참조되면, 페이지의 위치가 해시리스트의 헤드에 있으므로 페이지를 탐색하는데 걸리는 탐색 시간을 줄일 수 있게 된다.

4. 성능 평가

이 장에서는 시뮬레이션 결과를 통해 본 논문에서 제시한 페이지 캐시 관리 방법이 기존 방법에 비해 좋은 성능을 갖는다는 것을 보인다. 본 논문에서 수행한 시뮬레이션은 다음과 같은 가정하에 진행되었다.

가정 1 : 전체 101개의 파일이 존재한다
가정 2 : 찾고자 하는 파일의 크기는 29K 바이트이고, 나머지 100개의 파일은 임의의 크기를 갖는다.

멀티 스레드를 가지고 있는 시뮬레이션 프로그램을 구현하여, 캐시의 리스트를 통해서 프로세스가 탐색을 수행하

도록 하였다. 구현한 프로그램은 4개의 스레드가 생성되고 각각의 스레드가 101개의 파일 중에서 하나의 파일에 임의로 접근하는 방식의 수행을 한다. 만약 접근한 파일이 찾고자 하는 파일이 아닐 경우에는, 그 파일에서 1 바이트의 값을 순차적으로 읽어 들이고 포인터를 4K 바이트 옮긴다. 이 임의 접근과 읽기 과정을 스레드마다 10000번 수행한다. 지정한 파일의 크기가 29K 바이트이므로 각각의 스레드에 대해서 적어도 8번의 접근이 필요하다. 이 과정에서 필요한 탐색회수와 수행시간을 측정하여 기존 방법과 본 논문에서 제시한 방법의 성능을 비교하였다.

일반적으로 프로세스는 그 크기가 크지않은 데이터 파일이나 DLL 파일들을 참조하므로 표본 파일의 크기로 29K 바이트는 충분하다고 할 수 있다. 또, 100개의 더미 파일을 열고 읽는 과정을 통해 해시 리스트에 연결되는 페이지의 수를 테스트 하기에 충분히 큰 상황을 만들었다. 따라서, 우리의 가정은 타당성을 갖는다.

보다 정확한 결과를 얻기 위해 시뮬레이션을 반복적으로 실행하였다. 시뮬레이션에서 리눅스의 성능은 현재의 페이지 캐시의 상태에 의존한다. 시뮬레이션 프로그램이 필요로 하는 페이지가 해시 리스트의 헤드쪽에 위치해 있을 경우 수행 시간은 상대적으로 짧게 된다. 따라서 시뮬레이션 프로그램이 참조하는 페이지의 리스트에서의 위치가 가변적이 되도록 하기 위해 많은 페이지를 페이지 캐시에 붙였다.

표 1은 기존의 커널에서의 수행 시간을 나타내고, 표 2는 수정된 커널의 수행 시간을 나타낸다.

표 1. 기존 커널의 프로세스 수행 시간 측정

기존의 기법							
적은 수의 노드				많은 수의 노드			
탐색 회수	수행 시간(ms)			탐색 회수	수행 시간(ms)		
	최소	최대	평균		최소	최대	평균
1	31.4	56.1	40.3	1-4	33.5	68.8	42.4
1	29.9	55.4	39.4	1-4	33.5	68.6	42.3
1	31.0	32.9	31.7	1-4	33.4	59.1	42.1
1	30.0	61.7	44.5	1-4	33.2	67.7	42.3

표 2. 수정된 커널의 프로세스 수행 시간 측정

수정된 기법							
적은 수의 노드				많은 수의 노드			
탐색 회수	수행 시간(ms)			탐색 회수	수행 시간(ms)		
	최소	최대	평균		최소	최대	평균
1	30.9	33.0	31.6	1	30.8	64.1	39.4
1	31.0	63.3	39.3	1	31.3	33.2	31.8
1	31.3	66.4	40.8	1	31.2	62.3	39.1
1	29.7	56.6	38.7	1	31.1	62.8	39.2

해시 리스트에 연결된 노드의 수가 적을 경우에는 기존의 방법과 본 논문에서 제시한 방법간에 큰 차이를 보이지 않는다. 즉, 탐색 회수도 1회로 동일하고 수행시간도 비슷하다. 그러나, 본 논문에서 제시한 방법에서는 한 번 찾았던 페이지 캐시를 다시 찾기 위해 필요한 시간이 기존 방법에 비해 적기 때문에, 해시 리스트에 연결된 노드의 수가 많아짐에 따라 성능의 차이가 나타난다.

따라서, 해시 리스트에 많은 노드가 연결되어 있는 경우에는 기존의 방법의 경우 탐색 회수가 4번까지 필요하지만, 본 논문에서 제시한 방법의 경우 1번으로 탐색으로 충분하다. 수행 시간의 측면에서도 본 논문에서 제시한 방법의 성능이 기존 방법에 비해 6.4%에서 25%까지 개선되었다.

5. 결론

본 논문에서 우리는 리눅스 페이지 캐시 리스트에서 페이지를 탐색하는데 걸리는 시간을 줄이기 위한 새로운 기법을 소개하였다.

리눅스에서 기존의 방식은 페이지가 메모리로 읽혀질 때, 해시 리스트의 테일에 페이지가 덧붙여진다. 이 방법은 연산이 간단하다는 장점이 있지만, 자주 읽혀지는 페이지가 리스트의 테일에 있는 경우 탐색 시간이 오래 걸리는 단점이 있다. 본 논문에서는 기존 페이지 캐시 관리 방식의 성능을 개선시키기 위해서 우리는 페이지 탐색 알고리즘을 수정하였다. 탐색된 페이지를 해시 리스트의 헤드로 위치시켜서 다음 탐색 때 그 페이지를 찾는데 필요한 탐색 시간을 줄일 수 있도록 하였다.

페이지가 더 이상 어떤 프로세스에 의해서 사용되지 않는 경우에 페이지가 캐시에서 제거된다. 이런 페이지 캐시의 상태 변화에 대한 고려는 향후 과제로 남겨둔다.

6. 참고 문헌

[1] Olivier Temam, " Investigating Optimal Local Memory Performance" , ACM, 1998.

[2] Satyendra Bahadur, Vi swanathan Kalyanakrishnan, James Wetall, " An Empirical Study of the Effects of Careful Page Placement in Linux" , ACM, 1998.

[3] A. Silberschatz and P. B. Galvin, " Operating Systems Concepts 5th Edition, " Addison -Wesley Longman, 1998.

[4] M Beck, H Bohme, M Dziadzk a, U Kunitz, R Magnus, and D Verworner, " Linux Kernel Internals 2nd Edition" , Addison-Wesley, 1998.

[5] Scott Maxwell, " Linux Core Kernel Commentary " , CoriolisOpen Press, 1999.

[6] David A . Rusling , " The Linux Kernel " , Linux Documentation Project, 1998.

[7] Remy Card, Eric Dumas, and Franck Mevel, " the Linux Kernel book" , John Wiley & Sons Ltd, 1998