Linux 운영체제에서 가상메모리 압축 기법에 대한 연구

정진우[0] 장승주
동의대학교 컴퓨터공학과
jinwoo@dongeui.ac.kr, sjjang@dongeui.ac.kr

Virtual Memory Compression System on Linux Operating System

Jin-Woo Jeong[0] Seoung-Ju Jang
Dept. of Computer, DongEui University

요 약

가상 메모리 관리에서 가장 큰 문제점은 느린 보조기억 장치의 속도와 빠른 주기억장치의 속도 차이에서 나타나는 성능 저하라고 할 수 있다. 요구 페이징 기법에서 Page Fault가 일어나면 페이지 교체 정책에 의해 필요 없는 page들을 swap device로 이동을 시킨다. 이때 느린 보조기억장치의 접근 속도로 인한 응답시간의 지연은 전체적인 시스템 성능의 저하를 초래한다. 그래서 Swap Device로의 접근 횟수와 페이지의 크기를 줄일 수 있다면 Page Out되는 응답시간을 높일 수 있을 것이다. 따라서 본 논문에서는 가상 메모리 압축 시스템을 설계하여 Swap Out 되는 시간을 줄여 시스템의 전체적인 성능 향상을 위한 시스템을 구현한다.

1. 서 론

운영체제에서 메모리 관리 서브시스템은 가장 중요한 부분 중 하나이며, 초창기의 컴퓨터에서부터 시스템에 물리적으로 존재하는 것보다 더 많은 양의 메모리를 필요로 해왔다. 물리적인 메모리의 한계를 극복하기 위한 여러 기법들이 개발되었는데, 가상 메모리 기법이 가장 성공적이다. 가상 메모리 시스템에서의 문제점은 페이지 폴트(fault)가 일어났을 때, 느린 스왑 디바이스로 페이지들이 이동하면서 성능 저하가 발생한다는 것이다. 페이지 폴트가 일어나면 프로세스는 I/O 오퍼레이션이 일어난다. 그리고 프로세스는 Run 상태에서 Sleep 상태로 전의하면서 CPU의 점유권을 상실하게 되며, 페이지 교체 정책에 의해 주기억장치 내의 페이지들을 스왑 디바이스로 이동을 시킨다. 이후에 이 페이지들이 다시 필요하면 스왑 디바이스에서 주기억장치로 읽어 들이게 된다. 이렇게 페이지 폴트가 일어났을 때 이들 페이지들을 스왑 디바이스로 이동시키는 과정에서 I/O 오퍼레이션으로 인한 속도저하와 성능 저하가 일어난다[5].

가상 메모리 압축 시스템은 스왑 아웃되는 페이지들을 압축하여, 주기억장치 내의 압축된 캐시 영역에 저장하여 스왑 디바이스로 이동하는 시간과 횟수를 감소시켜 페이지 폴트 응답시간을 줄이며, 전제적인 시스템 성능을 향상 시킬 수 있다[1].

2. 관련 연구

가상 메모리 압축 시스템은 1993년에 Fred Douglis에 의해 이미 연구가 되었다. Douglis의 연구는 DECstation 5000/200 workstation에 Sprite OS를 사용하여 가상 메모리 압축 시스템을 실험했다. 그는 유동적인 크기의 원형 버퍼의 구조를 가진 압축된 캐시 영역을 만들고, 이 영역에 스왑 아웃되어 스왑 디바이스로 이동하는 페이지들을 저장하여 스왑 디바이스로의 I/O 횟수를 줄임으로써 시스템의 성능을 향상시키고자 하였다[4].

Douglis의 연구 결과 압축 캐시의 성능은 어플리케이션의 동작에 영향을 받으며, 압축과 I/O 작용의 상대적인 비용에 의존한다는 것을 증명하였다. 그리고 압축 캐시를 사용하지 않은 시스템보다 2~3배 정도의 속도 향상을 할 수 있다는 것을 증명하였다. 그러나 그의 연구는 특정 어플리케이션에서만 스왑 아웃될 때의 traffic을 감소시키거나 제거 할 수가 있었으며, 대부분의 모든 응용 프로그램에서는 적용되지 못했다.

Douglis의 연구 이후 Doug Banks와 Mark Stemm은 포터블 디바이스 내에서의 압축된 메모리 시스템을 제안했는데 Douglis의 연구에서의 문제점으로 Desktop PC나 Workstation에서는 압축된 메모리 시스템에서의 시스템 성능향상은 기대할 수 없으므로 포터블 디바이스 내에서의 압축된 메모리 시스템을 고안하였다. Doug Banks와 Mark Stemm의 연구는 PDA(Personal Digital Assistant)나 Laptop과 같은 포터블 디바이스가 느린 통신 수단으로 연결되어 크기가 큰 응용 프로그램에 접근하는 경우이다. 그래서 그들의 실험은 느린 스왑 디바이스로의 접근하는 양을 줄여 시스템의 성능 향상을 증명하였다[3].

3. 압축 메모리 시스템의 설계

페이지 스왑핑(Swaping) 메커니즘은 가상 메모리 시스템의 일부로서 더티(dirty) 페이지들을 분류하여 스왑 디바이스로 페이지들을 보냄으로서 물리 메모리의 공간 확보를 위한 것이다. 압축 메모리 시스템의 동작은 크게 스왑아웃(Swap Out)과 스왑인(Swap In)으로 이루어진다.

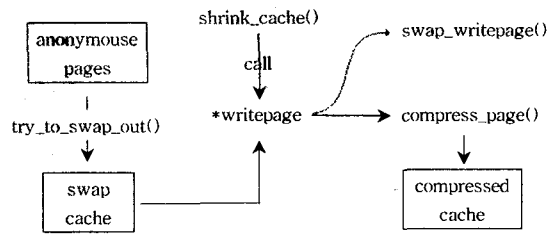

그림 1. Swap out의 과정

그림 1은 스왑 아웃의 전체적인 과정을 보여주고 있다. 스왑 캐시(Swap Cache)로 들어오는 페이지들은 shrink_cache()의 writepage 호출에 의해 스왑 디바이스로 보내지게 되는데, writepage는 함수 포인터 구조로 되어 있고, 이 함수 포인터는 swap_writepage() 함수의 주소를 가리키고 있다. 스왑 디바이스로의 이동은 swap_writepage()에서 호출하는 rw_swap_page()에 의해서 각 디바이스에 따라 다르게 구현된 코드에 의해 처리가 된다. 여기서 writepage가 compress_page() 함수의 주소를 지정하게 하고, 압축된 페이지를 압축 메모리 영역으로 이동시키도록 하면, 압축 메모리 시스템에서의 스왑 아웃 과정이 이루어지게 된다.

스왑인(Swap In) 과정은 매우 간단하게 동작을 하는데, 페이지 폴트가 발생하면 그림 2에서 보는 것과 같이 스왑 캐시에서 페이지를 찾게 되고, 만약 스왑 캐시에 없다면 압축 캐시에서 찾는다. 마지막으로 압축 캐시에도 없으면 스왑 디바이스에서 찾게 된다.

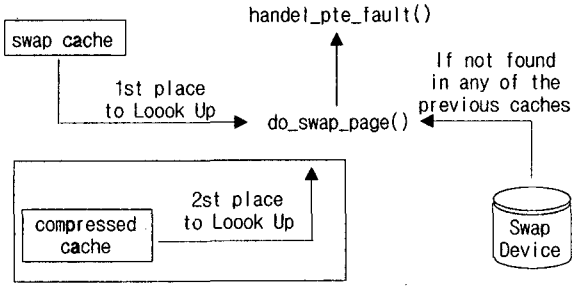

그림 2. Swap In의 과정

4. 압축 알고리즘

압축 기술에서 가장 중요하게 고려해야 할 사항은 압축 알고리즘의 압축율과 실행 속도 그리고 메모리의 요구량을 우선적으로 생각해야 한다. 기존의 압축 알고리즘들은 대부분 파일을 압축하기 위해서 고안되었다. 이런 압축 알고리즘들은 1byte의 문자 단위로 압축을 처리하는데, 대부분의 압축 알고리즘들이 연속된 문자들의 반복된 형태를 찾아서 압축을 한다. 그러나 메모리 내에서의 데이터들은 거의 byte 단위의 문자들을 포함하지 않는다. 메모리 내의 데이터들의 특징은 프로세서에 의해 빠르게 처리 될 수 있도록 Double-Word 단위나 Word 단위로 처리되며, 메모리 내의 데이터이기 때문에 이진 데이터들이 특성을 가지고 있으며, 메모리 영역에 대한 동일한 포인터들을 다수 포함하고 있다. 그리고 압축할 데이터의 크기가 항상 페이지 크기인 4Kbyte라는 특징도 가지고 있다[8].

본 논문에서는 복잡한 수행 과정 없이, 간단하며 빠르게 동작하면서 압축률이 좋은 LZ(Lempel -Ziv) 기반의 알고리즘을 변형한 압축 알고리즘을 고안하였다. LZ 기반 알고리즘은 압축할 자료를 코드화 하면서 기억장치 내에 문자열에 대한 색인표(Index Table)를 생성하여, 한 문자열씩 확인하여 그 내용이 같은 문자열을 다시 만나면 그것을 미리 간직해둔 하나의 색인을 갖게 된다[6,7]. 그렇게 중복되는 모든 부분들은 색인표를 가리키는 하나의 Index를 가지게 돼 그 크기를 줄일 수 있다.

본 논문의 압축 알고리즘은 전체 비트가 0과 1로 구성된 데이터를 적은 비트로 압축하고 사전에 등록된 데이터와 사전에 등록되지 않은 데이터들을 제어 비트에 의해 처리되도록 구현한다. 그리고 사전의 운영은 해쉬를 사용하여 구현한다.

5. 가상 메모리 압축 시스템 구현

메모리에 관련된 모든 소스는 커널 소스 디렉토리의 " mm/" 에 있다. 그래서 압축 메모리 시스템의 구현은 " mm/" 디렉토리의 소스들의 수정과 기능 추가를 위해 새로운 소스들을 생성하여 구현하였다.

압축 메모리 시스템에서 가장 핵심적인 부분이 커널 스왑 데몬(kswapd)의 구현 부분으로, kswapd는 kswapd_init()에 의해서 시작이 된다. 그림 3은 압축 메모리 시스템의 스왑 아웃 과정의 전체적인 흐름을 보여준다.

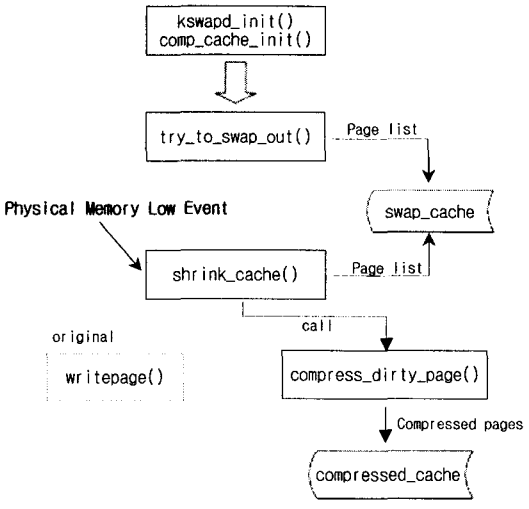

그림 3. Swap Out의 흐름도

압축 메모리 시스템은 커널 스왑 데몬이 시작하는 kswap_init()과 함께 초기화가 된다. 그리고 메모리 관

리 시스템에 의해 스왑 아웃될 페이지들이 swap_cache로 들어오게 된다[2]. 이때, 주기억장치의 공간이 모자라게 되면 커널 스왑 데몬은 주기억장치의 공간을 늘리기 위해 스왑 아웃을 수행하게 되는데, 스왑 아웃이 되는 writepage() 함수를 compress_drity_page로 수정하여, 압축 메모리 시스템을 구현한다. 이 함수는 스왑 캐시로 들어오는 페이지들을 압축하여, 주기억장치 내에 이미 할당된 압축 캐시 영역에 저장하였다가 스왑 인이 발생하면, 다시 압축을 풀어서 프로세스에게 페이지를 돌려준다.

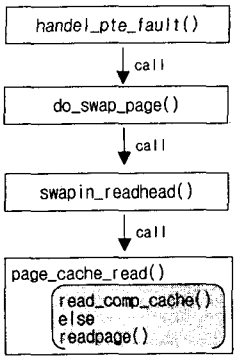

그림 4. Swap In의 흐름도

스왑 인의 과정은 그림 4에서 보는 것처럼 최종 실행되는 곳이 readpage() 함수이다. readpage()함수는 스왑 디바이스에서 페이지를 가져오게 되는데, 이 작업이 이루어지기 전에 read_comp_cache()에 의해 압축 영역에서 필요한 페이지를 가져오게 하면 된다.

6. 실험 및 성능 평가

실험에 사용된 시스템은 인텔 CPU 933MHz와 384MB의 주기억장치, 256KB의 시스템 캐시를 갖는다. 실험 방법은 주기억장치의 메모리를 강제 할당 및 해제를 수행하여, 스와핑이 일어나도록 하는 실험을 하였다.

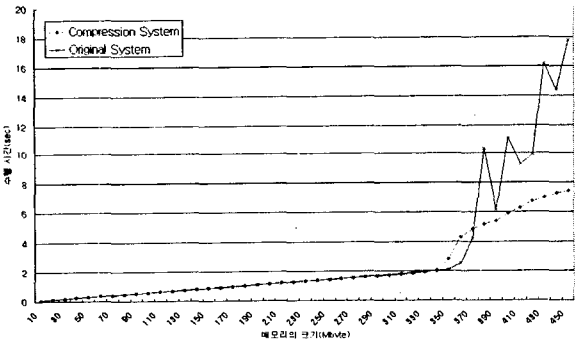

그림 5. 메모리양에 따른 수행 시간 측정

수행 시간 측정 실험으로 주기억장치의 남아있는 메모리 공간보다 더 많은 메모리의 할당을 요구할 경우에 효율성을 얻을 수 있다는 것을 알 수가 있는데, 이것은 그림 5의 그래프에서 주기억장치의 크기보다 크게 메모리를 할당하고 해제하는 시점에서, 시스템에 스왑이 발생하고, 스왑이 더욱더 많이 발생할수록 시스템의 수행 시간을 많이 소모한다는 것을 알 수 있었다. 이렇게 가상 메모리 압축 시스템을 적용할 경우 시스템의 주기억장치와 스왑 디바이스와의 접근에서 발생하는 지연 시간을 감소시켜 수행 시간의 효율성을 얻을 수 있었다.

7. 결 론

본 논문에서는 주기억장치를 압축되지 않은 영역과 압축된 영역으로 나누어서 Page Fault가 발생 했을 때 느린 Swap Device로 이동하는 페이지들을 압축된 메모리 영역에 저장하고, 이후에 이 페이지들이 프로세스에 의해 필요로 될 때 느린 Swap Device가 아닌 압축된 메모리 영역에서 가지고 옴으로써 보조기억장치로의 접근 횟수를 줄일 수 있도록 구현 하였다. 그리고 본 논문의 실험 결과 가상 메모리 압축 기법을 적용한 시스템이 주기억장치의 최대 가용 메모리 양보다 많은 메모리를 필요로 할 때 수행 속도의 성능이 좋아지는 것을 보았다.

본 논문에서는 Swap Out되는 페이지들을 압축하여 주기억장치 내의 압축 영역에 저장하는 메모리 관리 기법인 가상 메모리 압축 시스템을 구현함으로써 Swap Out되는 시간을 줄여서 전체적인 시스템의 성능 향상을 할 수 있음을 보였다.

향후 연구로는 좀더 효율적인 압축 알고리즘의 고안과 시스템의 상태에 따른 압축 영역의 크기를 유동적으로 변경하는 기법을 계획하고 있다.

참 고 문 헌

[1] Caroline Benveniste, "Cache-Memory Interfaces in Compressed Memory System", 2001 IEEE

[2] David A Rusling, "The Linux Kernel", January 25, 1999

[3] Dong Banks, Mark Stemm, "Investigating Virtual Memory Compression on Portable Architectures", January 19, 1995

[4] Fred Douglis. "The compression cache: Using on-line compression to extend physical memory". USENIX Proceedings, Winter 1993

[5] Ian McDonald, "The use of a Compressed Cache in an Operating System supporting Self-Paging", September 8, 1999

[6] Ross N. Williams, "An Extremely Fast Ziv-Lempel Data Compression Algorithm", Data Compression conference, 1991

[7] S. Kwong and Y. F. Ho, "A statitical Lempel-Ziv Compression Algorithm for Personal Digital Assistant(PDA)", 2001 IEEE

[8] Scott F. Kaplan, "Compressed Caching and Modern Virtual Memory", 1999