論 文
41~7~10

버퍼 캐쉬와 가상메모리 화일을 이용한 대형 데이타화일의 처리방법 설계 및 구현

A Design and Implementation on Large Data File Management Using Buffer Cache and Virtual Memory File

金 炳 喆*·申 秉 錫*·曺 東 燮**·黃 熙 隆***
(Byeong-Chul Kim · Byeong-Seok Shin · Dong-Sub Cho · Hee-Yeung Hwang)

Abstract - In this paper we design and implement a method for application programs to allow handling of large data files in DOS environment. In this method we use extended memory and hard disk as a data buffer. And we use a part of the conventional DOS memory as a buffer cache which allows the application program to use extended memory and hard disks transparently. Using buffer cache also allows us some speed improvement for the application program.

Key Words : Virtual Memory File(가상메모리 화일), Buffer Cache(버퍼 캐쉬), Extended Memory(연속확장 메모리)

1. 서 론

근래에 발표되는 대다수의 응용프로그램들의 특징은 기능이 복잡해지고, 처리하는 데이타의 양이 많아진다는 점이다. 따라서 보다 넓은 데이타 공간을 확보해주기 위해, DOS사용자들은 몇 KB의 데이타라도 더 적재할 수 있는 영역을 찾아내려 노력하고 있다.

DOS는 단일 사용자 단일 작업 (single user single tasking) 운영체제로서, 640KB라는 메모리 한계를 갖고 있다.[7] DOS가 처음 발표될 당시에는 640KB는 상당히 큰 메모리로 인식되었으나 그 후 PC 하드웨어의 가격이 저렴해지고 처리속도도 마이크로 컴퓨터의 수준까지 이르게 되자 소프트웨어 설계자들은 지금까지 여러가지 제약조건들 때문에 추가하지 못했던 각종 기능들을 응용프로그램에 추가시키기 시작하였고, 그 결과 1980년대

*正 會 員 : 서울大 大學院 컴퓨터工學科 博士課程
**正 會 員 : 梨花女大 電子計算學科 副敎授 · 工博
***正 會 員 : 서울大 工大 컴퓨터學科 敎授 · 工博
接受日字 : 1991年 11月 29日
1次修正 : 1992年 4月 15日
2次修正 : 1992年 6月 1日

중반에 들어서면서 DOS 사용자들은 필연적으로 메모리 공간의 부족에 직면하게 되었다.

본 연구에서는 제한된 메모리 공간에서의 DOS 응용 프로그램에 대해 보다 넓은 데이타 공간을 제공하기 위한 방법으로, 처리하고자 하는 대형 데이타 화일은 연속확장 메모리나 하드디스크에 저장해두고, 상용 메모리에는 버퍼캐쉬(buffer cache)[9, 13]를 이용하여 현재 처리중인 화일의 일부분만 저장하여 둠으로써 사용자가 연속확장 메모리와 하드디스크의 용량이 허용하는 범위에서 대형의 데이타 화일을 처리할 수 있도록 해주는 방법을 제시하고자 한다. 이 방법은 기억장소 확장의 기능적인 측면과 프로그래머가 쉽게 사용할 수 있도록 하는 편리성까지도 고려하고 있다.

2장에서는 DOS에서 어떻게 메모리를 사용하고 있는지, 그리고 현재 제한된 메모리 공간을 확장시키기 위해 어떤 방법을 사용하고 있는 지를 알아보고 그 방법들의 문제점을 제시하였다. 3장에서는 본 논문에서 설계 구현한 메모리 관리시스템의 기본 개념과 동작을 설명하였다. 4장에서는 이를 토대로 실험한 결과를 정리하였으며 5, 6장에서 앞으로 해결해야 할 문제점을 제시하고 결론을 맺었다.

2. DOS의 메모리 사용법과 기존의 해결책

2.1 DOS의 메모리 사용법

DOS에서 1MB의 메모리는 64KB로 된 16개의 세그먼트 (segment)로 구성된다. 이들 중 처음 10개의 세그먼트는 DOS, 디바이스 드라이버 (device driver) 및 응용 프로그램을 적재(load)하는 용도로 예약되어 있다. 그 다음 2개의 세그먼트인 A0000과 B0000은 비디오버퍼(video buffer)용으로 예약되어 있고 세그먼트 C0000에서부터는 어댑터 ROM (Read Only Memory)이 사용하는 공간으로 예약되어 있다. 세그먼트 E0000은 원칙상 시스템 BIOS(Basic Input/Output System)확장용으로 예약되어 있으나, 실제로는 빈 상태로 남아 있는 경우가 많다. 끝으로 시스템 BIOS가 F0000에서 시작되는 공간을 사용하고 있다.[7, 10]

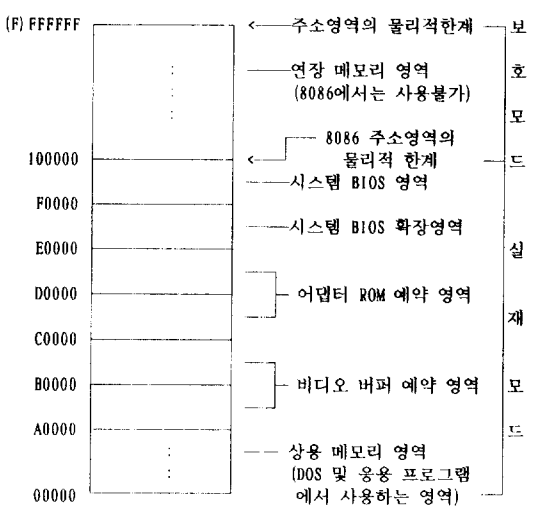


그림 1 DOS 메모리 맵
Fig. 1 DOS memory map

그림 1은 DOS에서 관리하는 메모리의 각 세그먼트위치와 해당 세그먼트의 용도를 설명한 것이다.

2.2 기존의 해결책

DOS 환경에서 부족한 메모리를 추가하기 위한 방법으로 그림 1의 메모리 맵상에서 1MB이내의 사용되지 않는 공간을 사용하는 백필(back fill)[11]이 있다. 그러나 이 방법을 사용하기 위해서는 하드웨어 지원 뿐만 아니라, 시스템 BIOS에서 이를 조사해서 DOS가 부팅되기 이전에 DOS가 알 수 있는 영역에 이를 표시하는 작업도 선행되어야 하며 그 크기가 최대 96KB 밖에 되지 않기때문에 메모리 부족의 근본적인 해결책이 될 수 없다.

페이징(paging) 또는 뱅크 스위칭(bank switching) 기법을 사용하는 증첩확장 메모리(expanded memory)[1~3]는, DOS 주소 공간의 사용되지 않는 상위 메모리 중에서 연속적인 64KB의 페이지 프레임(page frame)을 16KB 단위의 페이지로 분할하고, 여기에 증첩확장 메모리의 페이지들을 몇 개의 I/O명령어에 의해 전환(swich) 시킴으로써 액세스 가능하게 하는 방법이다. 그러나 증첩확장 메모리는 16KB 단위의 메모리 페이지가 동시에 스왑(swap)되기 때문에 주소 지정이 직선으로 되지 않고 16KB 단위의 주소공간으로 분할되게 되므로 멀티태스킹 소프트웨어를 지원하는데는 매우 유용하나, 연속적인 데이타를 저장하기에는 부적합한 면이 있고 증첩확장 메모리를 실현하기 위한 하드웨어로 인해 가격이 상승된다는 단점이 있다.

연속확장 메모리 (extended memory)[4]도 메모리 부족을 해결하기 위한 방법의 하나이다. 8086 이후에 나온 80286과 80386/80486은 주소 핸이 24개 및 32개 이므로, 단순히 메모리만을 추가함으로 해서 1MB이상에 위치하는 영역의 연속확장 메모리를 데이타 공간으로 확보할 수 있다. 그

러나 연속확장 메모리를 액세스 하려면, 조작모드를 실제모드에서 보호모드로 전환시켜야 하기 때문에, 비록 80286, 80386, 그리고 80486 프로세스들이 더 큰 주소 공간을 지정할 수 있는 능력이 있지만, 1MB까지만 직접적인 주소지정이 가능한 DOS 환경에서는 이를 사용하기가 그리 용이하지 않다.

80386이나 80486 CPU를 사용하는 시스템에서는 멀티태스커 (DOS multitasker)나 DOS 확장자 (DOS extender) [11]처럼 가상 -86 모드 (virtual -86 mode)를 이용해서 1MB 상위의 연속확장 메모리를 증첩확장 메모리로 사용할 수 있다. 그러나 이를 위해서는 가상-x86모드를 지원하는 80386/80486 시스템을 사용해야만 한다.

3. 본 연구에 의한 해결책

본 연구에서는 DOS메모리의 부족을 해결하기 위하여 버퍼캐쉬 (buffer cache) [9, 13]와 가상메모리 화일(virtual memory file)을 사용한다. 버퍼 캐쉬와 가상메모리 화일을 사용하면 화일 전체를 메모리(상용 메모리와 연속확장 메모리)에 적재시켜 사용하므로 처리속도를 향상시킬 수 있고, 연속확장 메모리와 디스크의 넓은 영역을 메모리 공간으로 사용할 수 있다는 장점이 있다. 또한 프로그래머는 상용 메모리, 연속확장 메모리, 하드디스크중에서 어떤 부분이 어떻게 사용되는지 알 필요없이 단지 10개의 함수들만으로 메모리의 할당/반환, 읽기/쓰기를 할 수 있다.

3.1 연속확장 메모리 관리자 (Extended Memory Manager)

1MB이상의 연속확장 메모리를 DOS에서 사용할 수 있도록 만들어진 규칙인 XMS(extended Memory Specification)에서는 실제 모드에서 연속확장 메모리를 사용하기 위해 필요한 연속확장 메모리의 구성및 사용방법, 데이타의 저장방법, 제약조건등이 명시되어 있다.

연속확장 메모리 관리자(XMM : extended Memory Manager)는 XMS에 따라 연속확장 메모리를 사용하기 위해 만들어진 디바이스 드라이버이다. XMM이 하는 일은 응용 프로그램이 메모리를 필요로 할 때 연속확장 메모리의 일부를 할당해주고, 반환 받는 것과 상용 메모리와 연속확장 메모리간의 데이타를 이동시켜주는 것이다. 위의 두가지 기능을 이용하여 연속확장 메모리의 데이타를 처리한다.

3.2 버퍼캐쉬

캐싱(caching)은 자주 사용하는 데이타를 메모리의 일부분인 버퍼(buffer)에 저장하여 둠으로서 빈번한 디스크 액세스를 막아 데이타의 처리속도를 향상시키는 방법이다. 장착된 메모리의 크기보다 큰 화일을 처리하기 위해서는 필연적으로 디스크를 사용해야 하는데, 이 때 메모리에 비해 상대적으로 저속장치인 디스크는 가능한 사용하지 않아야 처리속도가 향상된다.

버퍼캐쉬를 사용할 때 같은 내용이 두개의 buffer에 저장되어 데이타의 불일치가 발생하는 일이 없도록 해야하며, 또 사용중인 버퍼와 사용하지 않는 버퍼에 "busy", "free"의 표시를 하여 구분이 명확하도록 해야 한다. 사용하지 않는 버퍼는 free 버퍼 리스트(free buffer)의 인덱스로 참조할 수 있게 해준다. 이렇게 하면 해쉬 함수를 이용하여 해당 버퍼를 쉽게 찾아갈 수 있게 된다.

3.3 기본 개념

첫째, 취급하려는 화일은 가상메모리 화일의 형태로 변환된다. 가상메모리 화일은 디스크에서 읽어들인 데이타화일을 연속확장 메모리와 하드 디스크상에 재구성한 것으로 그림 2와 같이 (크기, 내용)의 조합으로 되어 연속확장 메모리나 하드 디스크상에서의 offset을 가지고 내용을 쉽게 참조할 수 있다. 연속확장 메모리가 장착된 경우는 데이타를 디스크가 아닌 고속의 연속확장 메모리에서 다루기 때문에 연속확장 메모리의 크기보다 작은 화일을 다루는 경우는 처리속도가 빠르다. 만약 화일의 크기가 연속확장 메모리의 한계를 벗어날 때는 가상메모리의 화일이 하드디스크로 연장되는데 하드디스크에 저장되는 부분은 단순한 텍스트 화일의 형태가 아니라 연속확장 메모리에 저장된 것과 같은 형식을 가지므로 화일의 크기가 커져도 일관되게 처리할 수 있다. 연속확장 메모리는 반드시 필요한 것이 아니며, 연속확장 메모리가 없어도 버퍼캐쉬를 사용하므로 고속처리가 가능하다.

가상메모리 화일에 대한 액세스는 가상 주소(virtual address)를 이용해서 하는데 가상주소는 가상메모리 화일의 시작점으로부터 해당 데이타가 위치한 곳까지의 offset을 나타내는 4byte의 주소를 가리킨다. 연속확장 메모리가 장착된 경우는 할당받는 연속확장 메모리의 시작점이 offset 0가 되고, 하드 디스크만 사용하는 경우에는 디스크상의 임시화일의 시작점이 offset 0가 된다. 연속확장 메모리가 장착된 경우 가상주소의 상위 부분은

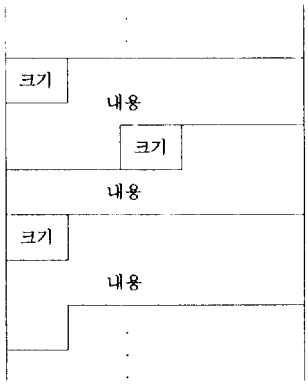


그림 2 가상메모리 화일의 구조
Fig. 2 Virtual memory file structure

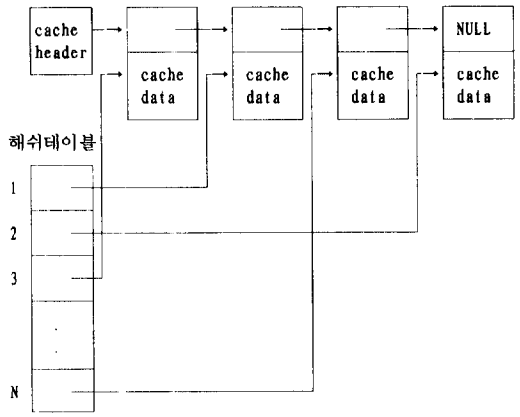


그림 3 버퍼캐쉬 서브시스템의 구조
Fig. 3 Buffer cache subsystem structure

연속확장 메모리에 사상(mapping)되어 있고, 하위 부분은 디스크에 사상된다. 따라서 가상주소가 연속확장에 메모리의 크기를 초과하는지 검사하여 초과하지 않는 경우는 연속확장 메모리에 읽기/쓰기를 하고, 초과하는 경우는 디스크에 읽기/쓰기를 한다. 물론 읽거나 쓰는 내용은 버퍼캐쉬를 경유하여 응용 프로그램에 전달된다.

그림 2는 가상메모리 화일이 연장메모리나 하드디스크상에 구현되었을 때, 할당받은 각 블럭의 논리적인 구조를 나타낸 그림으로 각 블럭은 (크기, 내용)의 조합으로 구성된다. 크기 field는 4 byte로 되어있다.

둘째, 데이타의 조작은 전적으로 버퍼 캐쉬를 통해서 이루어진다. 데이타의 일부를 읽을 필요가 있을 때는 버퍼로부터 읽어들이고, 버퍼에 필요한 데이타가 없을 때는 가상메모리 화일로부터 읽어들인다. 또 데이타를 쓸 경우에는 버퍼에 쓰고, 버퍼의 변형플래그를 표시하여 나중에 버퍼가 free될때 그 내용이 가상메모리 화일에 갱신되도록 한다.

본 연구에서 사용한 버퍼캐쉬 시스템에서 하나의 버퍼는 2KB의 데이타 영역과 링크 영역, 그리고 버퍼의 type을 나타내주는 플래그로 이루어져 있다. 캐쉬 헤더는 버퍼들의 링크 리스트를 관리하며, 해쉬 테이블은 입력된 가상주소를 변환하여 해당 버퍼의 캐쉬 데이타 영역으로 사상해준다. 본 연구에서는 모두 400개의 항목을 가지는 해쉬테이블을 사용하며 각각의 항목은 해당 버퍼의 캐쉬 데이타 영역을 가리키는 포인터로 되어 있다. 따라서 버퍼가 새로 할당되거나 반환된 개수로 나누는 방법을 사용한다. 한가지 특기할 사항은 버퍼의 free list가 없다는 점인데, 이것은 초기에 일정한 양의 메모리를 버퍼캐쉬 시스템에 예약(reserve)해서 사용하지 않고, 응용프로그램이 필요에 따라 할당/반환을 할 수 있도록 해주었기 때문이다. 그러나 최소한 1개 이상의 버퍼는 항상 유지된다.

만약 상용 메모리에 더이상 버퍼로 사용할 메모리가 없을 경우에는 LRU(least recenty used) 알고리즘에 의해 버퍼를 flush시키고 다른 데이타를 저장하는데 사용한다.

그림 3은 본 연구에서 사용한 버퍼캐쉬 시스템의 구조를 나타낸 것이다.

가상메모리 화일과 버퍼캐쉬가 결합된 메모리관리 시스템의 도움으로 사용자는 연속확장 메모리나 디스크에 대한 특별한 고려가 없이도 메모리를 임의로 할당받고, 데이타를 읽고 쓰는 것이 가능하다. 또 버퍼캐쉬의 사용은 가상메모리 화일이 연속확장 메모리나 디스크의 액세스 횟수를 줄여서 사용자가 상용 메모리를 사용할 때에 비해 속도차이가 거의 없도록 해준다.

3.4 시스템의 동작

본 연구에서 구현한 메모리 관리 시스템의 구성과 동작은 다음의 그림 4와 같다.

그림 4는 본 연구에서 구현한 메모리 관리 시스템의 구성 및 각 부분이 담당하는 역할을 설명한 것이다.

데이타를 읽을 때는 읽고자하는 데이타가 저장된 주소를 가지고 READ 명령을 내리면 된다. 먼저 버퍼에서 해당 데이타를 찾는데 주소는 가상주소로 주어지게 되며, 이것은 해쉬 함수에 의해 해당되는 버퍼의 캐쉬데이타 영역의 주소로 바뀐다.

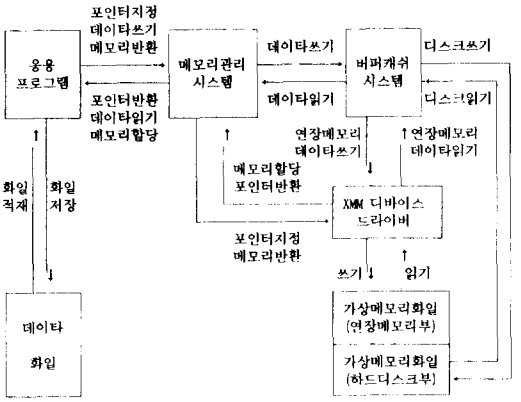


그림 4 메모리 관리시스템의 개괄적 구성
Fig. 4 Memory management system overview

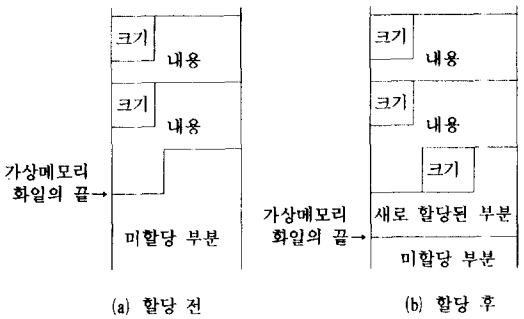


그림 5 메모리 할당시 가상메모리 화일의 변화
Fig. 5 Change on virtual memory file structure when allocated

데이타가 버퍼캐쉬의 특정 버퍼 내에 있으면 그 값을 넘겨주면 된다. 그러나 데이타가 들어있는 버퍼가 없는 경우는 그 데이타를 직접 가상메모리 화일에서 읽어야 한다. 읽은 데이타는 캐쉬 버퍼를 경유하여 응용프로그램으로 넘겨진다.

데이타를 쓰는 것은 읽는 동작과 유사하다. 다만 데이타를 쓸 때는 버퍼캐쉬에 변형플래그를 표시해두어서 나중에 버퍼를 다른 용도로 사용해야 할 경우, 또는 버퍼를 반환할 때 그 내용으로 가상메모리 화일이 갱신되도록 해주어야 한다.

메모리의 할당은 가상메모리 화일에서 이루어진다. 먼저 가상메모리 화일의 크기를 검사하여 연속확장 메모리의 범위를 넘지 않는 경우는 연속확장 메모리를, 넘는 경우는 디스크의 영역을 할당해준다. 할당 전후의 메모리상의 구성은 그림 5와 같다. 메모리가 할당될 때는 요구한 크기의 데이타 블럭에 크기를 나타내는 정보를 추가 해준다.

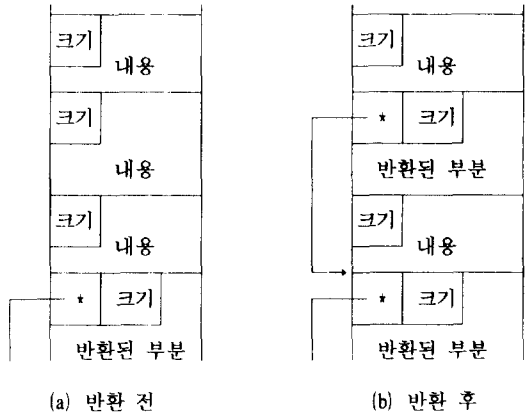


그림 6 메모리 반환시 가상메모리 화일의 변화
Fig. 6 Change on virtual memory file structure when released

표 1 메모리 관리용 사용자 함수
Table 1 User function for memory management

함수명	기 능
SetupCache	버퍼캐쉬 시스템 초기화
RemoveVMF	가상메모리 화일을 연속확장 메모리와 디스크에서 삭제
FreeBufCache	1개의 캐쉬버퍼용 메모리를 반환
FreeAllCache	모든 캐쉬버퍼용 메모리를 반환
WriteToVM	가상메모리 화일에 써넣기(블럭단위)
SetToVM	가상메모리 화일에 써넣기(4byte단위)
ReadFromVM	가상메모리 화일에서 읽기(블럭단위)
CetFromVM	가상메모리 화일에서 읽기(4byte단위)
AllocVirtual	가상메모리 할당
ReleasVirtual	가상메모리 반환

그림 5는 메모리의 할당 전후의 가상메모리 화일의 변화를 나타낸 것이다.

가상메모리 화일에서 사용중인 부분이 반환되면 그 부분은 free list에 삽입된다. free list는 반환된 메모리 공간들이 link list로 연결된 것으로 이를 위해서 반환된 메모리 블럭에는 link 영역이 추가된다. 인접한 두개의 메모리 블럭이 모두 free이면 두개의 블럭은 통합(coalescing)된다. [12]

그림 6은 메모리 반환전후의 가상메모리 화일의 변화를 나타낸 것이다.

본 논문의 메모리 관리시스템이 제공하는 사용자 함수는 다음 〈표 1〉과 같다.

3.5 다른 메모리 관리 프로그램과의 비교

DOS에서 제공하는 메모리 관리 프로그램으로 RAMDRIVE. SYS[14]와 SMARTDRV. SYS[14]가 있다. 이 두개의 프로그램은 설치가능(installable)한 디바이스 드라이버들로써 구조나 기능면에서 본 논문에서 제시하는 방법과 유사한 점들을 가지고 있다.

RAMDRIVE. SYS는 메모리의 일부를 하드 디스크처럼 사용할 수 있도록 해주는 디바이스 드라이버로서 여기서 사용하는 메모리 영역을 특별히 RAM DISK라 부른다. RAM DKSK는 정보를 하드 디스크보다 고속인 메모리에 저장하고 사용하기 때문에 속도가 매우 빠르다는 장점을 가진다. 또 연속확장 메모리나 중첩확장 메모리가 장착된 경우는 이 부분을 이용하여 여러개의 RAM DISK를 만들어 쓸 수 있다. 그러나 이 방법은 대형 데이타 화일을 사용할 경우 데이타의 처리는 여전히 좁은 상용 메모리 공간에서 해야하므로 기존의 방법에서 사용했던 바와 같이 오버레이를 하는 수밖에 없다. 따라서 처리의 복잡함은 줄어들지 않고 다만 하드디스크 대신에 메모리를 사용함으로 해서 속도만 향상될 뿐이다. 또한 연속확장 메모리가 장착되어 있지 않은 경우는 RAM DISK를 상용 메모리상에 만들기 때문에 기억장소 확장의 의미가 없어진다.

SMARTDRV. SYS는 연속확장 메모리또는 중첩확장 메모리가 장착된 시스템에서 사용되는 하드 디스크 캐쉬 프로그램으로서 하드 디스크의 액세스 횟수를 줄여서 전체적인 처리속도를 향상시키는 프로그램이다. SMARTDRY. SYS는 스웹이 빈번히 발생하는 프로그램에 사용하면 좋다. 이 프로그램은, 본 논문에서 제안한 방법이 상용 메모리를 이용하여 캐싱하는 것과는 달리 연속확장 메모리나 중첩확장 메모리로 캐싱을 한다. 이 방법의 문제점은 대형 데이타 화일을 처리할 때 RAM DISK와 같이 복잡한 오버레이 기법을 사용해야만 한다는 점이다. 또 본 논문에서 제안한 방법은 연속확장메모리가 장착되어 있지 않더라도 사용가능한데 비해서 SMARTDRY-SYS는 연속확장 메모리나 중첩확장 메모리가 없으면 사용할 수 없는 문제가 있다.

4. 실험결과

실험을 위해 두개의 영문용 editor 프로그램인 RM. EXE와 VM. EXE를 만들었다. RM. EXE는 상용 메모리를 사용하면서, 메모리가 부족할 때 하드 디스크로 오버레이를 하는 것이고, VM. EXE는 본 논문에서 제시한 메모리 관리시스템을 사용한 것이다. 비교를 위해 메모리 관리용 함수를 제외한 모든 부분은 동수의 실행문과 변수를 사용하였다. 실험대상 시스템으로는 80386 CPU를 탑재하고 640KB의 상용 메모리와 4096KB의 연속확장 메모리를 가진 시스템을 사용했다.

4.1 처리 데이타의 양

새로운 화일을 생성하여 그 화일을 가상메모리 화일내에서 조작하면 하드 디스크의 잔여용량을 모두 사용할 때까지 처리가 가능하다. 그러나 이런 경우 조작을 마친 후에 다시 일반 화일로 저장할 수 없게 된다. 따라서 본 시스템이 처리할 수 있는 화일의 크기는 하드디스크 잔여량의 절반정도가 된다. 실제 가상메모리 화일에는 크기, 포인터를 저장하는 영역이 있으므로 처리할 수 있는 화일의 크기는 이보다 작아진다.

R : 하드디스크의 잔여량
E : 가용 연속확장 메모리 크기
M : 최대 처리가능 화일의 크기
V : 가상메모리 화일의 크기
L : '크기' field크기

$$V = M + L \tag{1}$$

$$R + E = M + V \tag{2}$$

1)을 2)에 대입하여 정리하면

$M = (R + E - L)/2$가 된다.

예를 들어 연속확장 메모리가 4MB장착되어있고 하드디스크의 잔여량이 10MB이고 화일의 행수가 10000행인 경우 처리가능한 화일의 크기는 약 7 MB정도가 된다.

$$M = (4000000 + 10000000 - 4 \times 10000)/2$$
$$= 6980\text{KB}$$

상용 메모리만을 사용하는 경우는 응용 프로그램을 적재하고 남은 공간만을 데이타 처리에 사용하기 때문에, 응용 프로그램, DOS, 디바이스 드라이버를 적재하고나면 수십내지 수백 KB정도의 화일을 처리할 수 있을 뿐이다.

4.2 속도 개선

본 시스템은 상용 메모리에 비해 처리과정이 복잡한 연속확장 메모리와 저속의 하드 디스크를 사용하기 때문에 상용 메모리만을 사용하는 시스템보다 빠를 수는 없다. 문제는 상용 메모리를 사용하는 경우에 비해 처리속도가 크게 떨어지지 않도

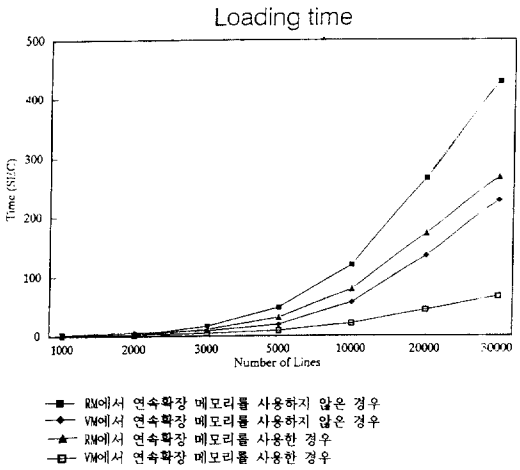


그림 7 적재시간 비교
Fig. 7 Loading time comparison

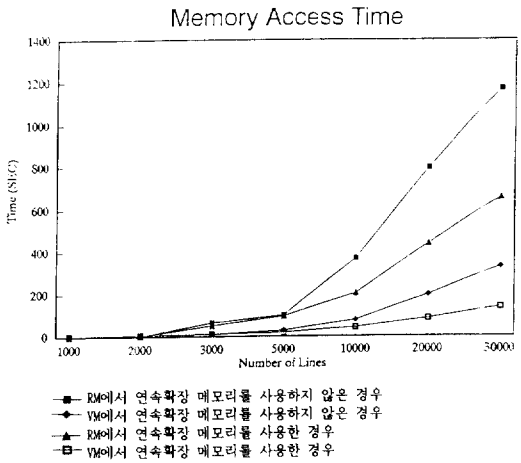


그림 8 가상주소 참조시간 비교
Fig. 8 Access time comparison

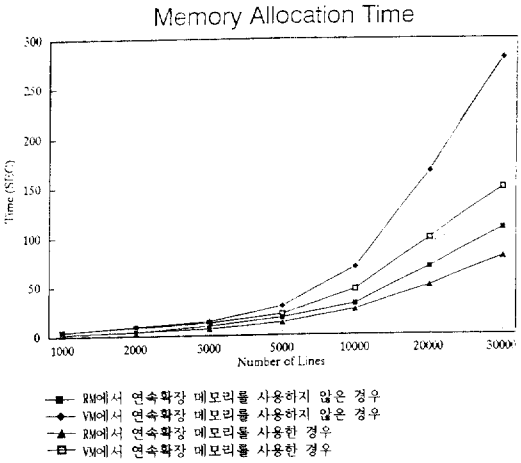


그림 9 메모리 할당시간 비교
Fig. 9 Memory allocation tiem comparison

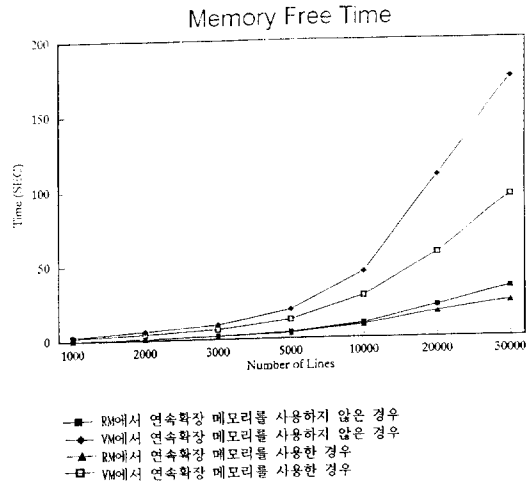


그림 10 메모리 반환시간 비교
Fig. 10 Memory release time comparison

록 해주는 것이다.

다음의 그림들은 상용메모리만을 사용하여 오버레이 기법으로 대형 데이나 화일을 처리하는 경우와 본 논문에서 제시한 메모리 관리 시스템을 사용한 두가지 경우에 대해 화일의 적재 속도, 특정 부분의 참조 속도, 메모리 할당/반환 속도를 비교한 것이다. 먼저 연속확장 메모리가 장착되어 있지 않을 경우 하드 디스크에 오버레이를 하는 RM과 본 논문의 메모리 관리 시스템을 사용한 VM을 비교하고, 다음에는 연속확장 메모리를 장착하여 RAM DISK에 오버레이를 하는 RM과 VM을 비교하였다.

그림 7에서 행수가 2000행 정도의 작은 화일인 경우까지는 오버레이가 발생하지 않기 때문에 RM의 속도가 약 1.5배~2.5배가 빠르다. 그러나 화일의 크기가 커서 오버레이가 발생하게 되면 VM이 RM보다 2배~4배 정도 빨라지는 것을 볼 수 있다. 또 같은 VM도 연속확장 메모리를 사용하는 것이 하드 디스크만 사용할 때보다 2배가량 빠른 것을 볼 수 있다.

그림 8에서는 화일내의 특정한 부분을 임의로 참조하는 경우에 소요되는 시간을 비교하고 있다. 오버레이를 하는 경우 한 부분을 집중적으로 참조하면 한번도 오버레이가 발생하지 않을 수 있으나

최악의 경우는 매번 참조할 때마다 오버레이가 발생할 수도 있다. 그래서 이 실험에서는 임의의 참조 패턴을 정하고 그 패턴을 4가지 경우에 적용하는 방법으로 비교하였다. 화일 적재시와 마찬가지로 작은 화일에서는 오버레이가 발생하지 않아서 RM의 속도가 VM보다 4배가량 빨랐지만 대형 화일의 경우는 VM이 RM보다 4배~5배 빠른 것을 볼 수 있다.

그림 9에서 메모리 할당시간을 비교해 보면 화일의 크기에 따라 다소간의 차이는 있지만 RM이 VM보다 1.5배~2.5배 가량 빠른 것을 볼 수 있다. 이것은 메모리 할당의 경우는 오버레이가 거의 일어나지 않기 때문이며, VM내의 메모리 관리 시스템에서 최초 적합(first fit) 알고리즘에 의해 가장 적합한 블럭을 찾고 크기와 링크정보를 갱신해주는 시간이 추가되기 때문이다.

그림 10에서 메모리 반환시간은 RM의 속도가 VM의 속도에 비해 3배~7배까지 빠른 것을 볼 수 있는데 이것 역시 메모리 반환시에는 오버레이가 거의 발생하지 않는 이유도 있고, VM내의 메모리 관리 시스템에서 인접한 두개의 free 블럭을 통합해주는 작업과 블럭의 크기및 링크정보를 갱신해주는 작업을 하기 때문이다. 특히 연속확장 메모리를 사용하지 않는 경우는 하드 디스크상에는 통합작업과, 크기, 링크정보 갱신작업을 해야하므로 상당히 느려진다.

이런 결과를 살펴보면 화일의 크기가 작아서 상용 메모리내에 적재하여 사용할 수 있는 경우나, 오버레이가 거의 발생하지 않는 메모리의 할당/반환 조작의 경우는 RM의 처리속도가 빠르지만 화일이 상용 메모리에 적재될 수 없을 정도로 크거나 화일의 적재, 내용 참조시에는 VM의 속도가 빠른 것을 볼 수 있다.

5. 개선할 점

첫째로 버퍼 캐쉬의 크기를 적절히 결정해야 한다. 버퍼 캐쉬의 크기는 처리속도나 성능에 큰 영향을 미치게 된다. 버퍼의 크기가 작으면 데이타가 적중하는 비율(hit ratio)이 떨어지기 때문에 디스크 또는 연속확장 메모리부분에 대한 액세스 횟수가 증가하게 된다. 반면에 버퍼의 크기를 너무 크게하면 응용프로그램이 화일의 데이타 이외의 다른 데이타를 저장하는데 필요한 메모리의 크기가 감소하여 동작이 어렵게 된다. 따라서 적절한 버퍼크기를 구하는 것은 매우 중요한 일이다.

둘째로 화일의 적재 시간을 줄여야 한다. 가상메모리 화일을 사용하는 시스템에서는 화일을 읽어서 가상메모리 화일의 형태로 재구성해야 하기 때문에 적재속도는 상당히 느려진다. 표 2는 상당히 큰 화일의 적재시간을 비교한 것이다.

결과에서 행수가 많아질수록 행당 평균 적재시간이 길어지는 것을 볼 수 있는데 이것은 가상메모리 화일이 하드디스크 영역으로 확장되면서 발생하는 문제이다. 현재로써는 초기적재시에만 소용되는 시간이기 때문에 대형데이타를 취급하기위한 오버헤드(overhead)로 간주되고 있지만 앞으로 연구를 통해 개선해야할 부분이다.

6. 결 론

본 연구에서 가상메모리 화일과 버퍼캐쉬를 사용하는 메모리 관리 시스템을 설계 구현하고 이를 응용프로그램에 적용시켜 DOS의 메모리 한계를 넘는 대량의 데이타를 처리할 수 있으며 처리속도가 상용메모리를 사용하는 경우에 비해 크게 떨어지지 않는 것을 보였다.

DOS 시스템에서 640KB 이상의 메모리를 액세스하는 것은 가능한 일이다. 다만 어떻게 하면 적은 비용으로, 사용자에게 불편을 주지 않으면서, 고속으로 대량의 데이타를 처리할 수 있느냐가 관

표 2 대형화일의 적재시간

Table 2 Loading time for large date file

행 수	가상메모리 크 기	연속확장 메모리 비사용		연속확장 메모리 사용	
		총 적재시간 (sec)	행당 적재시간 (msec)	총 적재시간 (sec)	행당 적재시간 (msec)
5000	520KB	19.40	3.88	9.11	1.82
10000	1.04MB	56.09	5.61	20.87	2.09
20000	2.08MB	135.05	6.75	43.52	2.18
30000	3.12MB	227.88	7.59	66.15	2.21

건이 된다. 또한가지 간과할 수 없는 것은 프로그래머들이 복잡한 메모리나 디스크의 조작법을 모르고도 사용하기 쉽도록 해주어 한다는 점이다. 본 연구에서 제시하는 방법도 이러한 여러가지 욕구를 충족시키기 위한 것이다.

본 연구는 현재 메모리 부족으로 고심하고 있는 많은 분야에 좋은 해결책을 제공해 줄 뿐 아니라, 앞으로의 응용 프로그램 개발자들이 DOS의 한계를 고려하지 않고 응용프로그램들을 개발할 수 있는 길을 열어줄 것이다.

참 고 문 헌

[1] Lotus, Intel, Microsoft, "LOTUS, INTEL, MICROSOFT ANNOUNCE MAJOR UPGRADE TO EXPANDED MEMORY SPECIFICATION," Microsoft, August 19, 1987, pp. 1~7

[2] Lotus, Intel, Microsoft, "Background Information On The Lotus/Intel/Microsoft Expanded Memory Specification Version 4.0," Microsoft, August 1987, pp. 1~10

[3] Lotus, Intel, Microsoft, "QUESTIONS AND ANSWERS ABOUT THE LOTUS/INTEL/MICROSOFT EXPANDED MEMORY SPECIFICATION VERSION 4.0," Microsoft, August 1987, pp. 1~10

[4] Microsoft Corporation, "eXtended Memory Specification (XMS) ver 2.0," Microsoft July 19. 1988, pp. 1~15

[5] Intel, "80386 Hardware Reference Manual," Intel, 1986, Chapt 6, pp. 1~27

[6] Intel, "MICROPROCESSORS," Intel, 1990, Chapt 3, pp. 1~61

[7] Microsoft, "The MS-DOS Encyclopedia," 1988, pp. 1~103

[8] Rakesh K. Agarwal, "80x86 ARCHITECTURE AND PROGRAMMING," Prentice-Hall International 1991, pp. 11~207

[9] MAURICE J. BACH, "The Design of the UNIX Operating System," Prentice-Hall International, 1986, pp. 38~59

[10] 퍼스널 컴퓨터 편집부, "PC에서의 메모리 관리," 퍼스널 컴퓨터, 1991년 7월호, pp. 130~141

[11] PC 어드밴스 편집부, "메모리 부족의 해결," PC 어드밴스, 1991년 2월호, pp. 148~152

[12] 조유근, 고전, "운영체제론," 홍릉출판사, 1990, pp. 159~251

[13] Samuel J. Leffler외 3인, "The Design and Implementation of the 4. 3BSD UNIX Operating System," Addison-Wesley Publishing Company, 1989, pp. 170~223

[14] Microsoft, "MS-DOS User's Guide and User's Reference," Microsoft Corporation, 1988, pp. 310~313

[15] John H. Crawford and Patrick P. Gelsinger, "Programming the 80386," SYBEX, 1987, pp. 655~667

저 자 소 개

김병철(金炳喆)

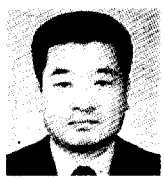
1959년 7월 20일생. 1982년 서울대 공대 컴퓨터 공학과졸업. 1984년 서울대 대학원 컴퓨터공학과 졸업(석사). 현재 서울대 대학원 컴퓨터공학과 박사과정.

조동섭(曺東燮)

1955년 10월 7일생. 1979년 서울대 공대 전기공학과 졸업. 1981년 서울대 대학원 전기공학과 졸업(석사). 1986년 서울대 대학원 컴퓨터공학과 졸업(공박). 현재 이화여자대학교 전자계산학과 부교수. 당학회 평의원.

신병석(申秉錫)

1967년 11월 7일생. 1990년 서울대 공대 컴퓨터공학과 졸업. 1992년 서울대 대학원 컴퓨터공학과 졸업(석사). 현재 서울대 대학원 컴퓨터공학과 박사과정.



황희융(黃熙隆)

1934년 3월 3일생. 1964년 서울대 공대 전기공학과 졸업 1974년 동 대학원 전기공학과 졸업. 현재 서울대 공대 컴퓨터공학과 교수(공박). 당학회 평의원.