가상메모리 화일과 버퍼캐쉬를 이용한 대형 데이타 화일의 처리에 관한 연구

김병철 , 신병석* , 황희융

서울대학교 컴퓨터공학과

(A Study on Large Data File Management Using Buffer Cache and Virtual Memory File)

Kim Byeongchul , Shin Byeongseok* , Hwang Heeyeung

Seoul National University Computer Engineering

< Abstract >

In this paper we have designed and implemented a method of using extended memory and hard disk space as a data buffer for application programs to allow handling of large data files in DOS environment. 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 some speed improvement for the application program. We have also implemented a number of functions to allow easier handling of pointer operations used by application programs.

1. 서론

근래에 발표되는 대다수의 응용프로그램들의 특징은 기능이 복잡해지고, 처리하는 데이타의 양이 많아진다는 점이다. 특히 워드프로세서나 화상처리 프로그램등은 많은 정보를 저장할 공간을 필요로 한다. 따라서 보다 넓은 데이타 공간을 확보해주기 위해, DOS 사용자들은 몇 KB의 데이타라도 더 적재해둘 수 있는 영역을 찾아내려 노력하고 있다.

DOS는 단일 사용자 단일 작업 (single user single tasking) 운영체제로서, 640KB라는 메모리 한계를 갖고 있다. 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 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 연장메모리나 하드디스크에 저장해두고, 상용 메모리에는 버퍼캐쉬(buffer cache)를 이용하여 현재 처리중인 화일의 일부분만 저장하여 둠으로써 사용자가 연장메모리와 하드디스크의 용량이 허용하는 범위에서 대형의 데이타 화일을 처리할 수 있도록 해주는 방법을 제시하고자 한다.

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 비디오 버퍼(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.

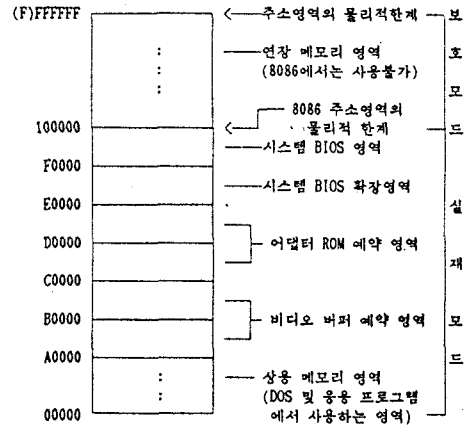

그림 1. DOS 메모리 맵

2.2. 기존의 해결책

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

페이징(paging) 또는 뱅크 스위칭(bank switching) 기법을 사용하는 확장메모리(Expanded Memory)는, DOS 주소 공간의 사용되지 않는 상위 메모리 중에서 연속적인 64KB의 윈도우를 16KB 단위의 프레임 (frame)으로 분할하고, 여기에 확장 메모리의 프레임들을 몇 개의 I/O 명령어에 의해 전환(switch) 시킴으로 해서 액세스 가능하게 하는 방법이다. 그러나 확장

메모리는 16KB 단위의 메모리 프레임이 동시에 스캡(swap)되기 때문에 주소 지정이 직선으로 되지 않고 16KB 단위의 주소공간으로 분할되게 되므로 멀티태스킹 소프트웨어를 지원하는데는 매우 유용하나, 연속적인 데이타를 저장하기에는 부적합한 면이 있고 확장 메모리를 실현하기 위한 하드웨어로 인해 가격이 상승된다는 단점이 있다.

연장메모리 (Extended memory)도 DOS의 메모리 부족을 해결하기 위한 방법의 하나이다. 8086 이후에 나온 80286과 80386/80486은 주소 핀이 24개 및 32개 이므로, 단순히 메모리만을 추가함으로 해서 1MB이상에 위치하는 영역의 연장 메모리를 데이타 공간으로 확보할 수 있다. 그러나 연장 메모리를 액세스 하려면, 조작모드를 실제모드 에서 보호모드로 전환시켜야 하기 때문에, 비록 80286, 80386, 그리고 80486 프로세서들이 더 큰 주소 공간을 지정할 수 있는 능력이 있지만, DOS 환경에서 이를 사용하기가 그리 용이하지 않다.

80386을 사용하는 시스템에서는 DOS 멀티태스커 (DOS Multitasker)나 DOS 확장자(DOS Extender)처럼 가상-86 모드 (virtual-86 mode)를 이용해서 1MB 상위의 연장메모리를 확장메모리로 사용할 수 있다. 그러나 이를 위해서는 가상-86모드를 지원하는 80386/80486 시스템을 사용해야만 한다.

기존의 메모리를 운영하는 방법을 개선하여 부족한 메모리 공간을 확보하는 방법도 있다. 예로써, 운영체제(Operating System)를 UNIX나 OS/2와 같은 멀티 태스킹(multi-tasking) OS로 채택하는 것이다. 일반적으로 멀티 태스킹 OS에서는 가상메모리 (virtual memory)를 지원하기 때문에 DOS에서와 같은 메모리의 한계를 갖기 않게 된다. 그러나 OS를 변경하기 위해서는 새로운 시스템(system)을 필요로 하거나 혹은 기존의 시스템에 새로운 설비의 추가를 필요로 한다. 그리고 OS를 변경하게 되면, 현재 사용중인 대다수 DOS 환경의 응용 프로그램을 사용할 수가 없게 된다는 문제도 있다.

3. 본 연구에 의한 해결책

본 연구에서는 DOS메모리의 부족을 해결하기 위하여 버퍼캐쉬 (buffer cache)와 가상메모리화일 (virtual memory file)을 사용한다. 버퍼 캐쉬와 가상메모리화일을 사용하면 화일전체를 메모리(상용 메모리와 연장 메모리)에 적재시켜 사용 하므로 처리속도를 향상시킬 수 있고, 연장 메모리와 디스크의 넓은 영역을 메모리 공간으로 사용할 수 있다는 장점이 있다.

3.1. Extended Memory Manager

1MB이상의 연장메모리를 DOS에서 사용할 수 있도록 만들어진 규칙인 XMS(eXtened Memory Specification)에는 실재 모드에서 연장메모리를 사용하기 위해 필요한 연장메모리의 구성및 사용방법, 데이타의 저장방법, 제약조건등이 명시되어 있다.

eXtended Memory Manager(XMM)은 XMS에 따라 연장메모리를 사용하기 위해 만들어진 device driver이다. XMM이 하는 일은 응용 프로그램이 메모리를 필요로 할 때 연장메모리의 일부를 할당해주고, 반환 받는 것과 상용메모리와 연장메모리간의 데이타를 이동시켜주는 것이다. 한다. 위의 두가지 기능을 이용하여 연장메모리의 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. 이를 위해서 메모리의 일부를 버퍼로 확보해두고, 데이타를 읽을 때는 먼저 버퍼를 확인하여 읽고자하는 데이타가 버퍼에 있을 경우에는 직접 읽고, 버퍼에 없을 때만 디스크를 액세스 한다.

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 list)에 보관한다. 또 버퍼 캐쉬에서 사용하는 모든 버퍼는 해쉬 테이블(hash table)의 인덱스로 참조할 수 있게 해준다. 이렇게 하면 해쉬 펑션을 이용하여 해당 버퍼를 쉽게 찾아갈 수 있게 된다.

3.3. 기본 개념

첫째, 취급하려는 화일은 가상메모리화일 (Virtual Memory File)의 형태로 변환된다. 가상메모리 화일은 디스크에서 읽어들인 데이타화일을 연장메모리상에 재구성한 것으로 그림 2와 같이 (크기, 내용)의 조합으로 되어있다. 따라서 연장메모리 상에서의 offset을 가지고 내용을 쉽게 참조할 수 있고, 데이타를 디스크가 아닌 고속의 연장메모리에서 다루기 때문에 화일의 크기가 연장메모리의 크기보다 작은 경우는 처리속도가 빠르다. 만약 화일의 크기가 연장메모리의 한계를 벗어날 때는 가상메모리 화일이 하드디스크로 연장되는데 하드디스크에 저장되는 부분은 단순한 텍스트 화일의 형태가 아니라 연장메모리에 저장된것과 같은 형식을 가지므로 화일의 크기가 커져도 일관되게 처리할 수 있다.

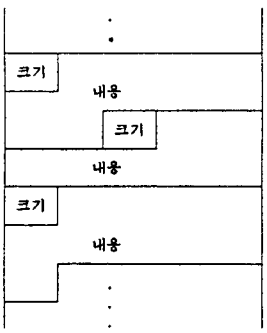

그림 2. 가상메모리 화일의 구조

가상메모리 화일에 대한 액세스는 특정한 위치의 데이타를 읽거나 쓸때 발생하는데, 이때 '특정한 위치'는 가상주소(virtual address)로 주어진다. 가상주소의 상위 부분은 연장메모리에 사상(mappig)되어 있고, 하위 부분은 디스크에 사상된다. 따라서 가상주소가 연장메모리의 크기를 초과하는지 검사하여 초과지 않는 경우는 연장메모리에 읽기/쓰기를 하고, 초과하는 경우는 디스크에 읽기/쓰기를 한다. 물론 읽거나 쓰는 내용은 버퍼캐쉬를 경유하여 응용 프로그램에 전달된다.

둘째, 데이타의 조작은 전적으로 버퍼 캐쉬를 통해서 이루어진다. 데이타의 일부를 읽을 필요가 있을 때는 버퍼로부터 읽어들이고, 버퍼에 필요한 데이타가 없을 때는 가상메모리화일로부터 읽어들인다. 또 데이타를 쓸 경우에는 버퍼에 쓰고, 버퍼의 변형 플래그를 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. 버퍼가 새로 할당되거나 반환된 경우는 해쉬테이블의 내용이 변경된다.

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.

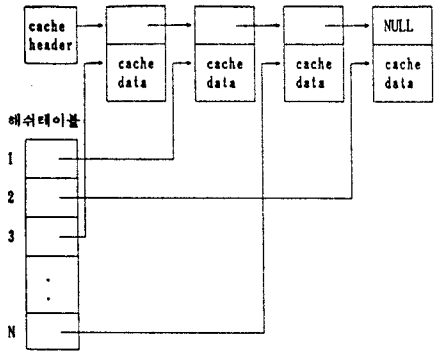

그림 3. 버퍼캐쉬 서브시스템의 구조

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

3.4. 시스템의 동작

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

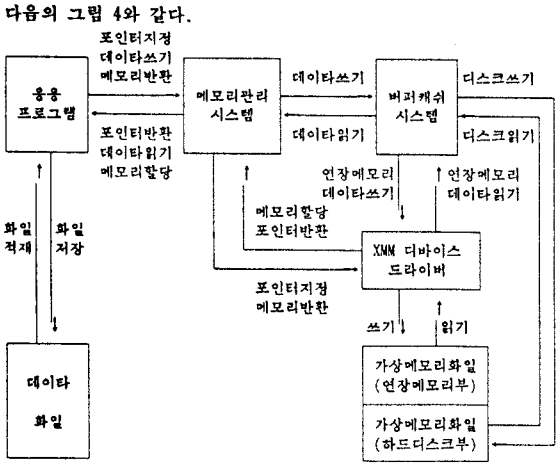

그림 4. 시스템의 개관적 구성

데이타를 읽을 때는 읽고자하는 데이타가 저장된 주소를 가지고 READ 명령을 내리면 된다. 먼저 버퍼에서 해당 데이타를 찾는데 주소는 가상주소로 주어지게 되며, 이것은 해쉬 펑션에 의해 해당되는 buffer의 캐쉬데이타 영역의 주소로 바뀐다. 데이타가 버퍼캐쉬의 특정 버퍼내에 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 쓸때는 버퍼캐쉬에 변형플래그를 표시해두어서 나중에 버퍼를 다른 용도로 사용해야할 경우, 또는 버퍼를 반환할 때 그 내용으로 가상메모리 화일이 갱신되도록 해주어야 한다.

메모리의 할당은 가상메모리 화일에서 이루어진다. 먼저 가상메모리화일의 크기를 검사하여 연장메모리의 범위를 넘지 않는 경우는 연장메모리를, 넘는 경우는 디스크의 영역을 할당해준다. 할당 전후의 메모리상의 구성은 다음 그림과 같다.

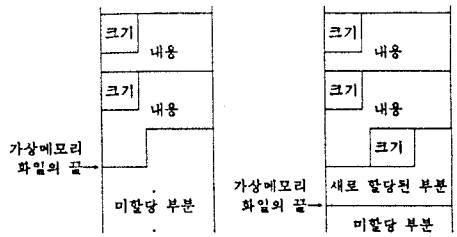

그림 5 메모리 할당시 가상메모리 화일의 변화

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 free이며 두개의 블럭은 통합(coalescing)된다.

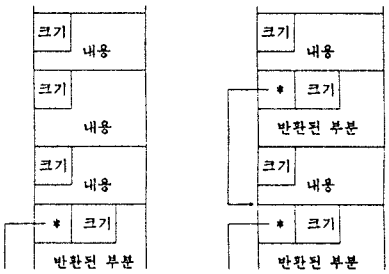

그림 6. 메모리 반환시 가상메모리 화일의 변화

4. 실험결과

실험을 위해 두개의 영문용 editor 프로그램인 RM.EXE와 VM.EXE를 만들었다. RM.EXE는 가상메모리 화일과 버퍼 캐쉬를 사용하지 않은 것이고, VM.EXE는 사용한 것이다. 정확한 비교를 위해 메모리 관리부를 제외한 모든 부분은 동수의 실행문과 변수를 사용하였다. 실험대상 시스템은 다음표와 같다.

	CPU	상용메모리	연장메모리
SYSTEM1	80386	640KB	384KB
SYSTEM2	80486	640KB	1024KB

표 1. 실험대상 시스템

4.1. 처리 데이타의 양

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

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

4.2. 속도 개선

본 시스템은 상용메모리에 비해 처리과정이 복잡한 연장메모리와 저속의 하드디스크를 사용하기 때문에 상용메모리만을 사용하는 시스템보다 빠를 수는 없다. 문제는 상용메모리를 사용하는 경우에 비해 처리속도가 크게 떨어지지 않도록 해주는 것이다.

본 시스템의 처리속도는 화일의 크기에 따라 달라진다. 화일의 크기가 연장메모리의 크기보다 작을 경우는 연장메모리만을 참조하기 때문에 처리속도가 상용메모리만을 사용하는 경우에 비해 크게 떨어지지 않지만, 가상메모리 화일이 하드디스크로 확장되는 경우는 디스크 액세스 횟수에 따라 처리속도가 떨어지게 된다. 그러나 버퍼 캐쉬 시스템을 사용하여 연장메모리나 디스크의 액세스 횟수를 최소화 시키기 때문에 속도가 크게 떨어지지는 않는다.

다음표들은 SYSTEM1과 SYSTEM2에서 각각의 크기의 화일을 RM과 VM에 적재해두고 가상주소 참조시간, 메모리 할당시간, 그리고 메모리 반환시간을 측정한 것이다.

	SYSTEM 1			SYSTEM 2		
행수	RM (sec)	VM(sec)	상대시간	RM (sec)	VM(sec)	상대시간
1000	1.09	1.58	1.45	0.38	0.44	1.15
2000	2.41	2.91	1.21	0.79	0.82	1.03
3000	3.84	4.51	1.17	1.04	1.21	1.16
4000	6.15	7.63	1.24	1.42	1.70	1.19

표 2. 적재시간 비교

표2에서 VM의 적재속도가 RM에 비해 크게 떨어지지 않는 것을 볼수 있다. RM의 경우 화일의 적재속도는 디스크와 상용메모리간의 전송시간만으로 결정되나, VM에서는 여기에 가상메모리화일 구성시간과 버퍼캐쉬 setup시간이 추가로 소모되기 때문에 VM의 적재시간이 길어진다.

	SYSTEM 1			SYSTEM 2		
행수	RM (sec)	VM(sec)	상대시간	RM (sec)	VM(sec)	상대시간
1000	1.00	3.74	3.74	0.45	0.60	1.33
2000	2.24	7.42	3.31	0.73	1.16	1.59
3000	3.35	11.26	3.36	1.11	1.83	1.65
4000	4.80	15.22	3.17	1.55	2.49	1.61

표 3. 가상주소 참조시간 비교

표3에서 SYSTEM에 따라 VM의 가상주소 참조시간이 1.5 - 3배가량 느린것을 볼 수 있다.

	SYSTEM 1			SYSTEM 2		
행수	RM (sec)	VM(sec)	상대시간	RM (sec)	VM(sec)	상대시간
1000	2.13	4.13	1.94	0.38	0.68	1.79
2000	4.10	8.57	2.09	0.79	1.45	1.86
3000	6.20	13.00	2.09	1.04	2.26	2.17
4000	8.85	17.20	1.94	1.42	2.29	2.04

표 4. 메모리 할당시간 비교

표4에서 VM의 메모리 할당시간이 RM보다 약 2배가량 느린 것을 볼 수 있는데 이것은 VM내의 메모리 관리 시스템에서 first fit 알고리즘에 의해 가장 적합한 블럭을 찾고 크기와 링크정보를 갱신해주는 시간이 추가되기 때문이다.

표5에서 VM의 메모리 반환시간이 RM보다 3-5배가량 느린 것을 볼 수 있는데 이것은 VM내의 메모리 관리 시스템에서 인접한 두개의 free 블럭을 통합해주는 작업과 블럭의 크기및 링크정보를 갱신해주는 작업을 하기 때문이다.

	[illegible]			[illegible]		
행수	RM (sec)	VM(sec)	상대시간	RM (sec)	VM(sec)	상대시간
1000	0.38	2.02	5.31	0.13	0.33	2.54
2000	1.00	4.40	4.40	0.22	0.78	3.55
3000	1.57	6.85	4.36	0.32	1.07	3.34
4000	2.32	10.15	4.47	0.43	1.30	3.02

표 5. 메모리 반환시간 비교

5. 결론

본 연구에서는 가상메모리 화일과 버퍼캐쉬를 사용하는 메모리 관리 시스템을 쉽게 구현하고 이를 응용프로그램에 적용시켜 DOS의 메모리 한계를 넘는 대량의 데이타를 처리할 수 있으며 처리속도가 상용메모리를 사용하는 경우에비해 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 관건이 된다. 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 막강한 기능을 가진 응용프로그램들을 개발할 수 있는 길을 열어줄 것이다.

6. 참고문헌

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

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

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

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

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

[6] MAURICE J.BACH,"The Design of the UNIX Operating System", pp. 38-59, 1986

[7] Samuel J.Leffler, "The Design and Implemetation of the 4.3BSD UNIX Operating System", pp. 170-223, 1989

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

[9] Intel, "MICROPROCESSORS" , pp. 3-1 ~ 3-61, 1990

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

[11] PC 어드밴스, "메모리 부족의 해결", pp. 148-152, 1991.2

[12] 퍼스널 컴퓨터, "PC에서의 메모리 관리", pp. 130-141. 1991.7

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