운영체제8-11장

2015253039 권진우

8장

1. (1) 가상메모리란 무엇이며 어떠한 장점이 있는가?

가상 메모리란 **프로그램을 논리 메모리와 물리 메모리를 분리시켜서 디스크의 일부 공간을 주 기억장치 메모리처럼 사용 가능하게 만든 공간을 디스크 가상 메모리 공간**이라고 하며 이를 이용하여 **메모리용량보다 더 많은 프로세스를 동시에 실행 시킬 수 있는 장점+프로세스들이 메모리와 파일 공유가 쉬워지는 장점**이 있고 페이지 단위 스와핑을 통해서 입출력시간 감소, 메모리 공간 절약, 빠른 사용자 응답시간, 향상된 멀티 프로그래밍 지원이 가능해졌다.

(2) 가상메모리를 요구페이징을 사용하여 구현하는 방법에 대해서 적으시오.

**요구 페이징을 이용하여 가상 메모리 구현하는 방법은 프로세스를 페이지 단위로 분할하여 메모리의 Page frame에 분할된 Page를 1:1로 할당하고 프로세스가 현재 필요한 페이지들만 메모리에 할당함으로써 메모리 공간을 절약하며 디스크와 메모리 사이에서 필요한 페이지가 있을 때 Swapping할 페이지들을 페이지 단위로 요구**하고 Swap in/out하며 가상메모리를 구현한다.

2. copy on write에 대해서 설명하시오. (연습문제 8.17 (9판 9.4) 참조)

**쓰기 시 복사로써** fork()를 이용하여 프로세스를 생성하면 **자식 프로세스는 부모 프로세스의 페이지 테이블만 복사하고 페이지는 복사하지 않는다**. 페이지 테이블만 복사하는 이유는 자식 프로세스가 공유 **페이지를 수정하려 할 때 복사한 페이지가 무효화 되는 것을 방지**하기 위함이며 공유 페이지 수정 시, **수정하려는 페이지만 복사해서 메모리의 다른 물리적 공간에 할당 후 수정**한다. -> 효율적인 프로세스 생성 법

3. (1) LRU page 교체 정책이란 무엇인가? 그리고 실제로 LRU 대신에 LRU 근사 알고리즘 을 사용하는 이유는 무엇인가?

LRU 페이지 **교체 정책이란** 페이지를 교체 할 때 메모리에 여유 페이지 frame공간이 없으면 메모리에서 디스크로 Swap out 될 **Victim 페이지를 선택**해야 하는데 이때 어떤 페이지를 교체 대상을 삼을지 선택하는 것이 교체 정책이다. **LRU 페이지 교체 방식은 최소 최근 사용 페이지 교체** 방식으로써 LRU알고리즘을 통하여 최근에 가장 사용이 되지 않은(가장 예전에 사용된) 페이지를 찾아서 해당 페이지를 Victim페이지로 삼는 정책이다. 하지만 LRU알고리즘은 H/W의 지원이 필요하고 Counter나 Stack을 사용하여 교체할 Page 선택을 위한 알고리즘을 수행하는데 이는 교체할 알고리즘을 선택하는데 걸리는 **오버헤드가 너무 크기** 때문에 **LRU근사 알고리즘을 통해 오버헤드를 줄인** 간단한 LRU 근사 알고리즘을 사용한다.

(2) 가장 간단한 하드웨어 지원으로 구현할 수 있는 LRU 근사 알고리즘은 무엇이며, 동작을 설명하시오.

-**참조비트** : 각 페이지 Entry에 참조비트를 추가하여 참조된 적이 없는 비트는 '0'이고 참조비트가 **'0'인 비트 중에서 랜덤으로 페이지 선택**

-**부가적 참조비트**(**History bit**) : 페이지 Entry에 참조비트가 아닌 참조 바이트를 추가하여 가장 왼쪽 비트를 참조비트, 우 7비트를 history bit로 사용하여 **일정시간 마다 비트를 Shift Right 함으로써 가장 오랫동안 사용하지 않은 페이지를 선택하는 LRU 근사 알고리즘이다**.(H/W의 지원이 필요하다.)

-2**차기회 알고리즘** : 참조 비트만 사용하고 Page를 **원형 큐로 구현하고** 페이지 교체 요청이 들어오면 해당 시점에서 원형 큐에서 참조비트가 '1'인 비트가 나오면 해당 비트는 0으로 바꾸고 참조비트가 '0'인 비트가 처음 나온 페이지를 교체대상으로 삼는 알고리즘이다.

-**개선된 LRU 근사 알고리즘** : **참조비트와 Modify비트를** 사용하며, 2비트를 통해 **4가지 조합을** 사용하여 참조되어 수정 된 적이 있는지 최근에 참조 된 적이 있는지를 확인하여 보다 효율적이고 가장 LRU알고리즘에 근사한 개선된 LRU 근사알고리즘이다.

\*Ref bit : 최근 참조된적이 있는지 / Modify bit : 수정된적이 있는지(과거에 참조된적)

\***페이지 buffering알고리즘 : 메모리 여유공간을 항상 두는 것, 지연된 Swap out**

5. Thrashing이란 무엇인가? 그리고 thrashing 발생하지 않도록 제어하는 방식을 설명하시오.

**쓰레싱(Thrashing)**

- 프로세스가 충분한 Page를 할당 받지 못하면 **Page fault가 너무 자주 발생**하여 Disk를 계속 액세스해야 하는 **I/O 인터럽트 및 작업이 많이 일어나서 문제**발생

1. 프로세스가 **Swap out/Swap in에 시간 소비**(오버헤드) = I/O burst 시간이 길어짐

2. **CPU 이용률이 낮아짐**

3. **OS**는 CPU이용률이 낮아서 **멀티프로그래밍 정도를 증가**시키도록 판단

4. **다른 프로세스**가 시스템(메모리)에 **추가**

5. 메모리가 부족해서 Page frame을 별로 할당 받지 못해서 일어난 문제인데 새 프로세스가 추가됨으로써 **Page fault가 더 자주 발생**

6. CPU이용률 더욱 저하 ---------------> ***악순환***

= 빈번한 page교체로 프로세스가 반복적으로 Swap in/out 하느라 바쁜 상황

해결법 : **지역성 모델을 통해서 각 프로세스에게 일정량 필요한 만큼의 Frame 수를 제공하는 방법** -> **PFF 방식** : 어떤 프로세스를 기준으로 상대적으로 Page fault rate가 높으면 frame 더 할당, Page fault rate가 낮으면 page frame 수 감소시킴

6. Memory mapped file에 대해서 설명하시오.

Memory Mapped File

- 디스크 블록을 메모리에 있는 Page에 맵핑

- mmap() system call 사용(in UNIX)

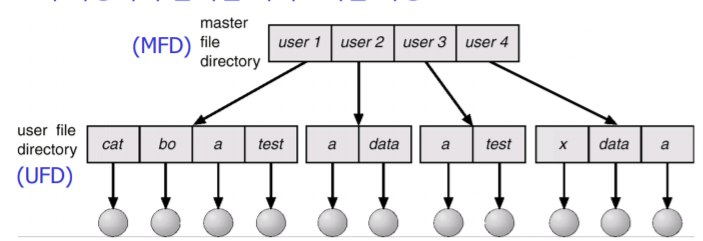
-> 여러 프로세스가 페이지를 공유하여서 같은 파일에 맵핑 가능

Disk는 가상 주소 공간에 프로세스를 Mapping하고 물리적 메모리로 적재하고 프로세스의 가상 논리 주소를 물리적 메모리로 복사(프로세스마다 서로 다른 논리 주소가 같은 물리적 메모리 주소를 공유할 수 있음)한다.

10장

1. 2단계의 open file table 구성에 대해서 설명하시오.

**Two** Level Directory



- 모두 분리된 디렉토리 사용

- 경로이름 예시 : /User/Test file

- 장점 : **분리된 Directory(디렉토리) 별로 같은 Directory이름 사용** 가능

효율적인 파일 탐색(2단계만 거치면 모든 파일 탐색이 가능하다.)

- 단점 : Grouping 단계가 2단계로 제한적이다.

2. (1) 디렉토리는 어떤 정보를 저장하는가?

디렉토리는 **해당 디렉토리 안에 있는 파일에 대한 정보(속성, 이름 등)을 저장**하고 있음

(2) 디렉토리 구조가 필요한 이유는 무엇인가? 그리고 디렉토리 구조를 설계할 때에 어떠한 점들을 고려해야 하는가?

**디렉토리** **구조를 통해서 우리는 파일을 체계적으로 단계별로 나누어서 구분지어 관리**할 수 있고 디렉토리 구조를 설계 할 때 고려할점?

3. 마운팅이란 무엇인가?

마운팅이란 **새로운 장치를** 루트를 가진 **디바이스가 마운팅 포인트를 제공함으로써 파일시스템에 접속(마운트) 시키는 것**이다. 이로써 파일 시스템의 확장하는 효과를 가져오게 된다. + 원격 지원

11장

1. 계층적 파일 시스템의 구성 및 각 계층의 동작에 대해서 설명하시오.

계층적 파일 **시스템은 App - 논리 파일 시스템 - 파일 조직화 모듈 - 기본 파일 시스템 - I/O 제어(disk driver) - device(장치**) 로 이루어진다.

각 계층에서의 동작은

-**Logical file sys**(논리 파일 시스템) : 메타데이터 정보 관리, FCB 보안과정 처리, 심볼릭 파일 이름 디렉토리 구조, 이름으로 된 파일의 실제 위치를 찾도록 해줌

-**file organization** **module**(파일 조직화 모듈) : 논리 블럭을 실제 Disk의 물리 블럭으로 변환시켜준다. -> Disk작동 가능

-**Basic file sys**(기본 파일 시스템) : 장치 드라이버에게 명령어를 제공, Disk의 메모리 버퍼와 캐시 관리

-**I/O 제어** : 장치 드라이버가 메모리와 디스크간 데이터 전송역할

2. (1) 파일 시스템을 구현할 때에 on-disk와 in-memory 자료 구조에 대해서 설명하시오.

파일시스템 구현 시 On-disk는 파일시스템이 디스크에 저장된 구조를 말하며 부트제어블럭, 볼륨제어블럭, 디렉토리 구조, 파일제어블록, 파일데이터로 구성되어있다.

1. **부트제어 블럭** : **파일 시스템과는 무관하며** **부팅에** 사용되는 정보가 있는 블럭(Disk의 첫 블럭+ 파티션당 1개씩있음)

2. **볼륨제어 블럭** : 해당 **파티션, 블럭에 대한 전체적인 정보**를 가짐(Unix : superblock/ Window : master file table)

3. **디렉토리 구**조 : 파일 이름과 같은 **inode 관련** 정보 저장

4. **파일제어 블**럭 : **inode에 있는 파일 제어 블럭(FCB)**

5. 파일 데이터

------------------------------------------------------------

**In-memory 자료구조는 Disk에서 파일을 실행(Open)하게 되면 inode의 정보를 메모리로 복사해와서 메모리에 적재하는데 메모리에 적재된 이 file(실행중인 프로세스의 자료구조)에 대한 자료구조가** In-memory 자료구조이다.

**1. 마운트 정보 2. 최근 사용한 디렉토리 정보 저장 3. inode(FCB)복사본**

**4. 프로세스당 open file table**

(2) 파일 read와 open 과정의 동작에 대해서 설명하시오.

-**최초로 디스크에서 파일이 open되어 실행 될 때, 사용자가 open명령어로 신호를 보내고 low level table에서 해당 file의 index값을 반환한다. -> 디렉토리 구조에서 열고자하는 파일의 디렉토리를 찾아가서 해당 디렉토리를 가져와서 디렉토리가 가리키는 위치의 FCB(inode)를 찾는다. -> FCB가 Sys-wide open file table에서 데이터 블럭들을 가리키고 디스크의 버퍼 캐시로 복사하고 버퍼로 옮겨지면 Open(read)이 완료된다.\*\*\*\*\***

\*FCB : 파일 제어 블록 : 파일에 대한 주요한 정보들을 가짐

3. (1) 파일에게 디스크 블록을 할당하는 기본적인 여러 방법에 대해서 설명하시오.

1.**연속(contiguous) 할당** : 디스크에 파일을 연속적인 메모리 공간에 할당

2.**링크(linked) 할당** : Linked list형식으로 블록들을 할당 하고 각 블록에는 다음 link에 대한 정보가 있음

3.**색인(indexed) 할당** : 인덱스 블록을 사용하여 각 디스크 블록들이 가지는 블록 Pointer를 저장하여 사용(파일 크기 제한)

(2) linked 할당방법과 FAT 방식은 어떠한 차이가 있는가?

Linked 할당은 **linked list형식으로 블록을 할당**하는데 각 블록에 다음 블록에 대한 link값이 저장 되어야 해서 각 **블록들이 메모리 낭비가 발생**하게 **되고 2^ 크기로 블럭에 저장할 수가 없는 문제**가 발생한다. / FAT 방식의 경우 **각 블록이 Link되어있는 정보를 가진 별도의 블록(File Allocation Table)을 저장**하여 파일 실행 시 **메모리에 FAT 블록을 복사**하여 찾으려는 블록을 빠르게 액세스 할 수 있게 됨

(3) UNIX의 파일시스템에서 사용하는 파일 공간 할당 방식에 대해서 설명하시오.

**Combined Scheme**으로 모드를 여러 개로 나누어(다단계 인덱스 Table 사용) **기본적으로 인덱스(색인) 할당 방식**을 사용하고 **Direct 블럭, 1단계, 2단계, 3단계 indirect 블럭을** 두어서 모드에 따라 파일 크기에 따라서 적절하게 할당하여 탐색 시 **파일 크기 특성에 따라** direct 블럭에서 찾을 지 Index 블럭을 2단계 찾아가는 double indirect index블럭을 사용 할 지 결정한다.

4. 로그구조 파일시스템에 대해서 간단히 설명하시오.

**복구를 위한 파일 시스템 구조인데** **파일 시스템에 대한 각 update를 transaction으로 기록하여** 모든 transaction은 로그에 비동기적으로 기록되고 **파일 시스템이 고장나면 로그를 통해서 복구 및 transaction은 다시 수행 될 수 있다.**

9장

1. 논리블록과 물리블록의 맵핑에 대해서 설명하시오.

**논리 블럭은 일반적으로 섹터단위 크기를 가지고 1차원 배열로 취급된다.**

**논리 블럭의 1차원 배열을 디스크의 섹터에 순차적으로 맵핑시킨다.**

**인접한 논리블럭 -> 인접한 물리블럭**

2. 디스크 스케줄링이란 무엇이며, 왜 하는가?

디스크의 파일을 불러 올 때 **OS는 하드웨어의 효율적인 사용을 위해서 디스크 데이터에 대한 접근 시간을 최소화하는 효율적인 스케쥴링이 필요**하게 되어 사용한다.

3. SCAN과 C-SCAN 스케줄링은 어떤 차이가 있는가?

SCAN은 디스크 Arm이 한쪽 끝에서 시작하여 다른 쪽 끝까지 이동하면서 가는 길의 모든 데이터 요청을 처리 -> 왔다 갔다 하면서 스캔 -> 가운데 트랙의 대기시간에 유리

C-SCAN은 SCAN이 가운데 블럭에 비중이 커지는 것을 방지하기 위해서 한쪽 방향에서 다른 한쪽 방향으로 한방향으로만 SCAN하여 처리하는 방식이며 SCAN보다 균일한 대기시간을 제공한다.

4. 여러 레벨 디스크 formatting에 대하여 설명하시오.

-**low level formatting** : **팩토리 공정**에서 **섹터를 구분**하기 위한 디스크를 적절한 자료구조로 기록 (sector = header + data + trailer)

-**Partition** : **여러개의 실린더로 나누어 구성**

-**Logical formatting** : **파일 시스템을 포맷**하는 작업

\*손상 블럭 처리방법: Sector sparing(해당 섹터 무시=스케쥴링 최적화에 영향을 줌) /

Sector slipping(밀어내기)