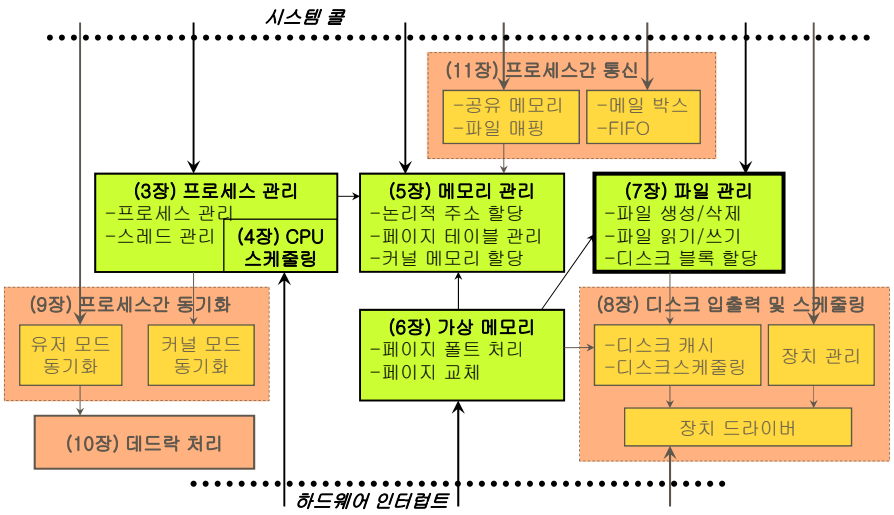
파일 관리

- ✓파티션과 파일시스템
- ✓파일 보호를 위한 정보관리
- ✓파일제어블록과 접근권한 검사
- ✓디렉터리 구조
- ✓파일별 데이터 블록 할당
- ✓빈 블록들의 목록관리
- ✓파티션 구성과 파일시스템 관리
- ✓파일시스템의 구현 예

관련 운영체제 구성 모듈

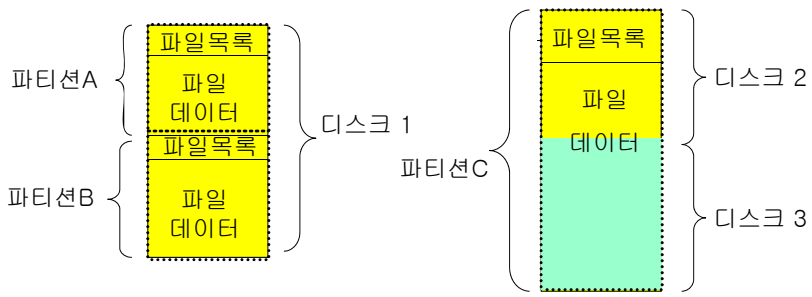


이해하고 넘어가야 할 내용들

- ✓ 파일, 파일시스템, 물리적 디스크, 논리적 파티션에 대한 이해 및 이들 간의 연관성에 대한 이해
- ✓ 디스크 상에 디렉토리 및 파일들의 배치 방법에 대한 이해 및 이들을 유닉스, 윈도즈 등에서 적용하고 있는 방식에 대한 이해
- ✓ 다중 사용자 시스템에서 파일 보호를 위해 정보를 관리하는 방법 및 프로세스가 파일을 접근하려 할 때 접근 권한을 검사하는 방법에 대한 이해
- ✓ 파일시스템을 구현하는 방법에 대한 이해

디스크와 파티션

- ✓ **하드디스크의 파티션 (partition)**
  - ✓ 디스크를 운영체제 내부에서 사용하는 논리적 파티션으로 구분
  - ✓ 파티션 별로 파일목록 부분과 파일데이터 부분으로 구성
- ✓ **디스크와 파티션**



파일과 파일시스템

- ✓ **파일 (file)**
  - ✓ 사용자의 정보를 저장하는 기록 단위로서 논리적으로 연속된 하나의 저장 내용
  - ✓ 파일과 실제 저장매체 기록 위치 간의 매핑은 운영체제 (파일시스템)에서 처리
  - ✓ 바이트 단위로 기록 (하나의 파일은 하나의 바이트 스트림)
- ✓ **파일시스템 (file system)**
  - ✓ 저장매체상에 파일들을 체계적으로 기록하고 관리하는 방식
  - ✓ 파일 목록과 파일 데이터들의 배치 및
  - ✓ 파일에 대한 생성/삭제/읽기/쓰기 등의 작업처리 등 포함
  - ✓ 파티션 별로 특정 파일시스템 적용
  - ✓ 예: 리눅스의 Ext2/3/4, 윈도즈의 NTFS/FAT32/ExFAT, 유닉스 파일시스템 등

파일 속성과 파일 작업

- ✓ **파일 속성 (attribute)**
  - ✓ 이름, 종류, 파일 보호 정보 (소유자, 읽기/쓰기/실행 가능 여부 등), 시간 (생성, 최종 수정, 최종 접근, 속성 변경 등)
  - ✓ 크기, 데이터 블록 (파일의 데이터들이 기록된 블록 번호) 등
- ✓ **파일에 대한 작업 (시스템 콜 함수)**
  - ✓ 생성, 삭제
  - ✓ 읽기, 쓰기, 열기, 닫기, 파일 내에서 읽기/쓰기 위치 변경
  - ✓ 크기 조정, 속성 수정
- ✓ **파일 접근 방식**
  - ✓ **순차 접근** (sequential access): 첫 바이트부터 차례대로 읽기/쓰기 작업 진행
  - ✓ 직접 접근 (direct access): 읽기/쓰기 요청 시에 파일 내의 위치를 지정

순차 접근과 직접 접근

작업 내용	순차 접근	직접 접근

처음부터 100바이트씩 파일 전체 읽기	`while (1) {`
	`  n = readSeq(file,`
	`  buffer, 100);`
	`  if (n <= 0)`
	`  break;`
	`}`
		`offset = 0;`
		`while (1) {`
		`  n = readDirect(file,`
		`  offset, buffer, 100);`
		`  if (n <= 0)`
		`  break;`
		`  offset = offset + n;`
		`}`

특정 offset 위치부터 읽기	`lseek(file, offset);`
	`readSeq(file,`
	`  buffer, 100);`
		`readDirect(file,`
		`  offset, buffer, 100);`

파일보호를 위한 정보관리

- ✓ **유닉스의 파일 보호 정보**
  - ✓ 권한을 소유자/소유그룹/기타 의 3가지 수준으로 구분
  - ✓ 각 수준별로 읽기/쓰기/실행 의 가능 여부를 표시
- ✓ **접근 권한의 검사**
  - ✓ 파일 열기를 실행할 때 요청한 프로세스의 사용자/그룹 정보와 대상 파일의 보호 정보를 비교하여 허용 여부 결정
- ✓ **유니스 파일 속성 예**
  - ✓ ㈜ '**`ls -l`**' 실행 결과

```
type/mode   link owner group   size     time        name
-rw-r--r--   1   kim    staff   4382  Aug 3  14:02  prog.c
-r--r-----   1   lee    staff   2346  Feb 15 11:17  message
drwxrwxr-x   3   park   student  512  Jul 8  09:35  work
```

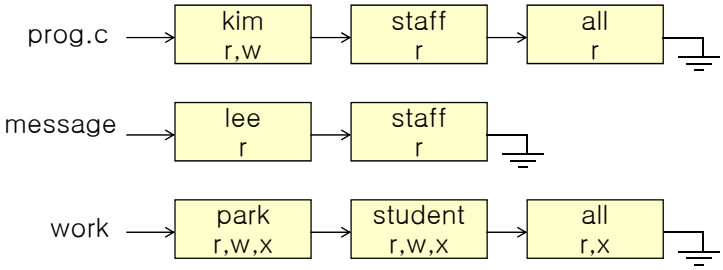
접근권한과 접근행렬

- ✓ **도메인 (domain)과 객체 (object)**
  - ✓ 객체: 접근 대상인 파일/디렉터리/장치 등
  - ✓ 도메인: 객체에 접근을 요청한 주체
- ✓ **접근 행렬 (access matrix)**
  - ✓ 모든 객체와 모든 도메인들에 대하여 접근 권한을 행렬 형태로 표시
  - ✓ 접근 행렬 예

도메인 객체	kim	lee	park	staff	student	all
prog.c	r,w	-	-	r	-	r
message	-	r	-	r	-	-
work	-	-	r,w,x	-	r,w,x	r,x

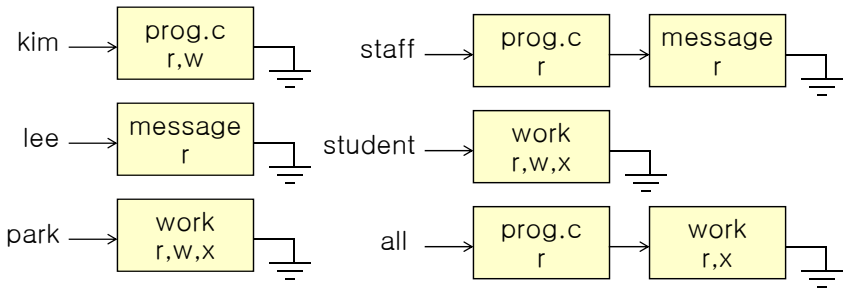
접근권한과 객체별 접근목록

- ✓ **접근 권한 정보를 파일 시스템 내에 기록해야 함**
- ✓ **접근행렬 (2차원 배열로 기록)**
  - ✓ 파일 갯수: 수십만~수백만개, 사용자 수: 수백~수만명
  - ✓ 지나치게 행렬의 크기가 크고 대부분 빈칸임
- ✓ **객체 별 접근목록 (Object access list, Access control list)**
  - ✓ 각 객체 별로 접근 가능한 도메인들을 목록으로 표시



접근권한과 도메인별 권한목록

- ✓ **도메인 별 권한 목록 (Domain capability list)**
  - ✓ 각 도메인 별로 접근 가능한 객체들을 목록으로 표시



접근 권한 기록 방법

- ✓ **객체별 접근 목록의 장점**
  - ✓ 일반적으로 도메인 수에 비해서 객체수가 월등히 많음 ➔ 도메인 별 권한 목록보다 목록 길이가 짧음
  - ✓ 도메인에 비해서 객체들의 생성 삭제가 빈번함 ➔ 도메인별 권한 목록에서는 관련된 모든 도메인들의 권한목록을 수정해야 함
  - ✓ 객체별 속성을 기록하는 부분에 접근 목록만 추가하면 됨 ➔ 객체별 접근목록이 구현하기에 단순함
- ✓ **유닉스의 객체별 접근 목록**
  - ✓ 객체당 도메인을 소유자, 소유 그룹, 모든 사용자로 3개만 허용
  - ✓ 도메인별로 권한은 읽기/쓰기/실행의 3가지만 적용

  ➔ 표현의 단순성 (소유자 ID, 소유그룹 ID, 권한 9비트)

파일제어블록

- ✓ **파일제어블록 (FCB: File Control Block)**
  - ✓ 파일 별로 속성들을 일괄적으로 정리하여 기록하는 영역
- ✓ **FCB의 예**

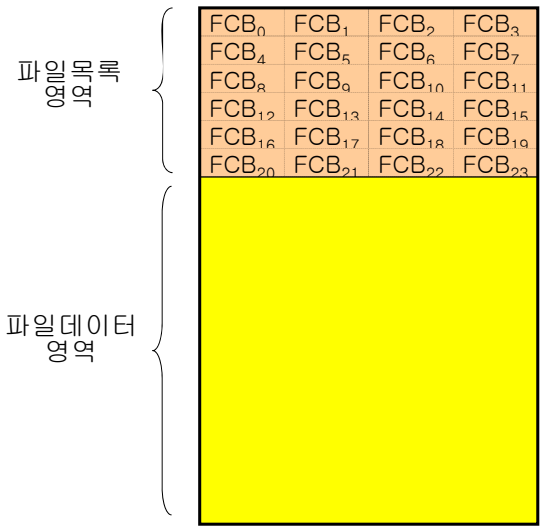
Name	
Type	Protection
Owner	Group
AccessTime	ModifyTime
ChangeTime	Size
DataBlockNumber	

```
struct FileControlBlock {
  char Name[16];
  int Type;
  int Protection
  int Owner, Group;
  int AccessTime, ModifyTime,
      ChangeTime;
  int Size;
  int DataBlock[20];
}
```

- FCB의 크기: 128 바이트 (int 4바이트, char 1바이트 가정)
- 평면구조 파일 시스템 가정

파티션의 파일목록 영역

- ✓ **일정한 크기의 FCB들의 배열로 기록**
- ✓ **파일 생성: 사용하지 않는 FCB를 찾아서 각종 속성들을 기록**
- ✓ (참고) 파일접근 속도를 높이기 위해 파일 목록영역을 몇 개로 분산하기도 함

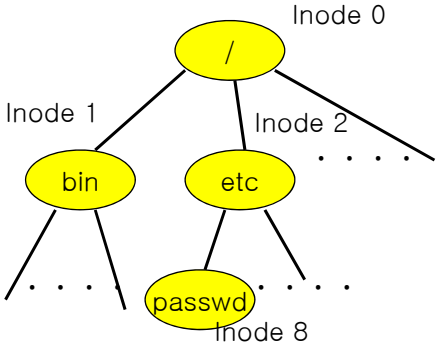


디렉터리 구조

- ✓ **디렉터리 구조**
  - ✓ 파일들을 체계적으로 분류하고 관리하는 체계
  - ✓ 평면구조, 트리 구조, 임의 구조
  - ✓ 대부분의 운영체제에서 트리 구조 사용 ← 사람의 인식체계
- ✓ **트리 구조 표현**
  - ✓ 디렉터리는 그 데이터 블록에 하위 파일/디렉터리들을 <이름, 고유번호> 쌍으로 등록
  - ➔ FCB에는 이름을 기록하지 않음
  - ➔ 고유번호는 FCB 들의 배열 인덱스를 활용

유닉스의 트리 구조 표현

- ✓ **inode (index node)**
  - ✓ FCB에 해당하며 전체 FCB 배열의 인덱스 번호가 파일별 고유번호임 ← inode 번호로 바로 해당 FCB 결정
- ✓ **유닉스의 트리 구조 예**



Inode 0 의 데이터 블록

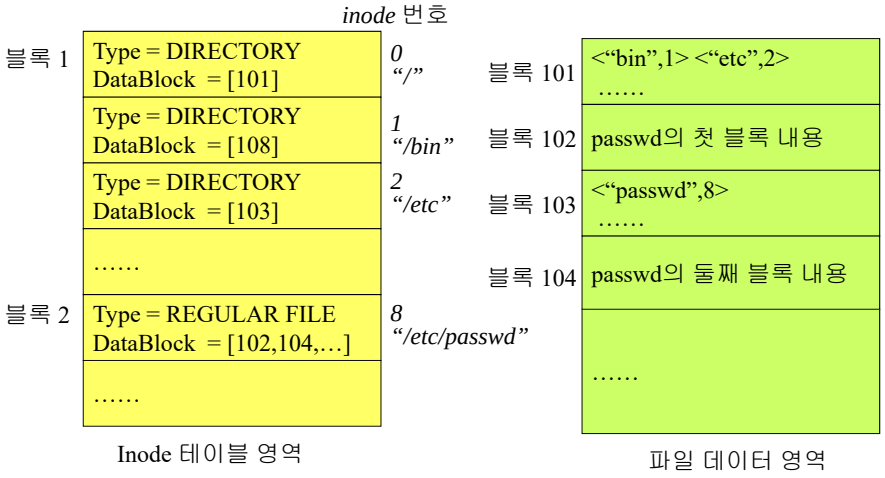
<“.”, ?> <“..”, ?>
<“bin”, 1> <“etc”, 2>
……

Inode 2 의 데이터 블록

<“.”, ?> <“..”, ?>
<“passwd”, 8>
……

유닉스의 inode 테이블

- ✓ 트리 구조를 반영한 inode 테이블과 파일 데이터 내용
- ✓ 가정: 1~100 블록 inode 테이블 영역, 101~끝 블록 데이터 영역

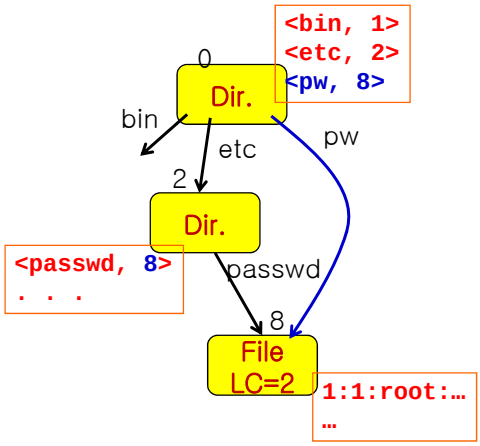


하드링크와 심볼릭링크

- ✓ UNIX의 트리 구조에서 부분적으로 임의구조 형태를 반영할 수 있는 기능
- ✓ 하드링크 (hard link)
  - ✓ 하나의 파일에 복수의 검색 경로허용 ← 복수의 디렉터리에 등록
  - ✓ 각 경로는 동일한 입장
- ✓ 심볼릭링크 (symbolic link)
  - ✓ 별도의 파일로 등록하고 그 데이터 블록에는 실제 파일의 경로를 기록 (파일 형식을 '심볼릭링크'로 구별)
  - ✓ 파일 접근시 '심볼릭링크' 형식의 파일이면 그 데이터 블록에 기록된 실재 파일의 경로로 대체하여 처리
  - ✓ (참고) 윈도즈는 '바로가기'

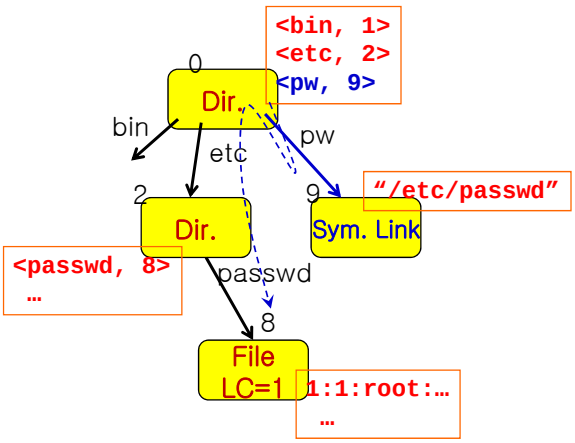
하드링크의 예

"/pw" 와 "/etc/passwd" 는 같은 파일의 하드 링크
(참고) ln /etc/passwd /pw



심볼릭 링크의 예

"/pw" 는 "/etc/passwd" 로의 심볼릭 링크
(참고) ln –s /etc/passwd /pw

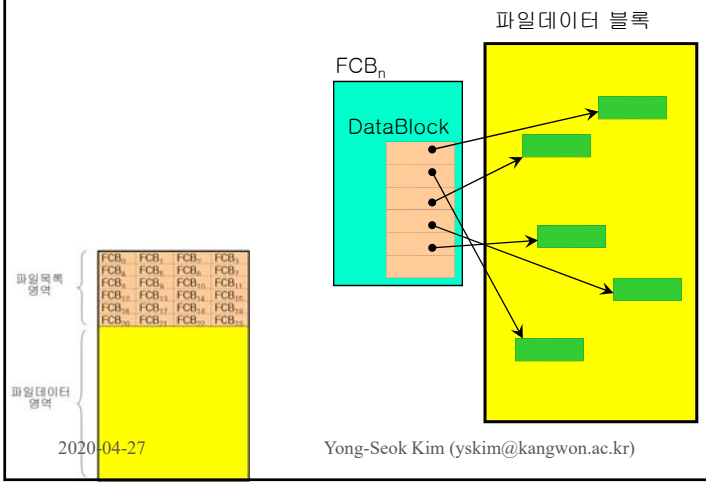


파일별 데이터 블록 할당

- ✓ **블록의 크기**
  - ✓ 섹터(sector): 디스크 입출력의 최소 단위 (512바이트)
  - ✓ 블록(block): 운영체제에서 연속된 몇 개의 섹터들을 묶어 하나의 처리단위로 사용
  - ✓ 블록의 크기는 보통 1KB, 4KB 등을 사용 (디스크 용량이 커지면서 블록 크기도 증가)
- ✓ **블록 할당 (block allocation)**
  - ✓ 파일별로 데이터 블록들을 할당하고 그 정보를 FCB에 기록하는 방식
  - ✓ 대표적인 방식: 인덱스 블록할당, 연속된 블록할당, 연결된 블록할당
- ✓ **블록할당 방식에서 고려할 항목들**
  - ✓ 파일내의 임의위치 접근의 효율성
  - ✓ 전체 디스크 공간의 효율적인 활용
  - ✓ 디스크 입출력 속도
  - ✓ 파일 크기의 제한 여부
  - ✓ 파일의 안정성

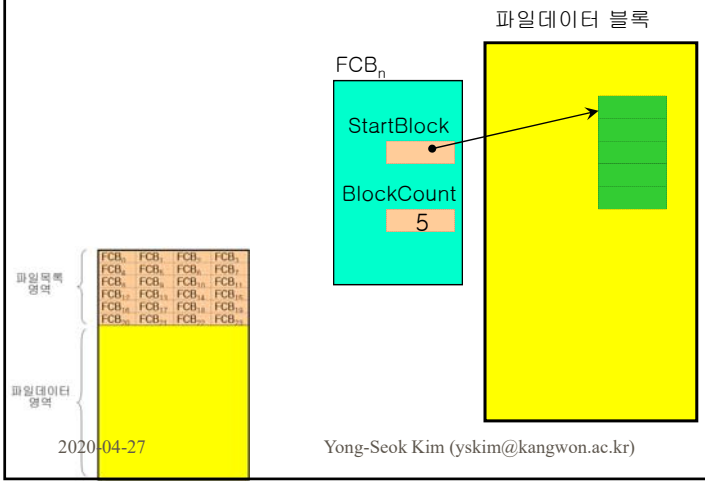
인덱스 블록할당

- ✓ **Indexed allocation**
- ✓ **FCB에 데이터 블록들의 목록을 배열로 기록**
- ✓ **데이터 블록할당은 임의 위치의 블록들을 필요한 개수만큼 할당**



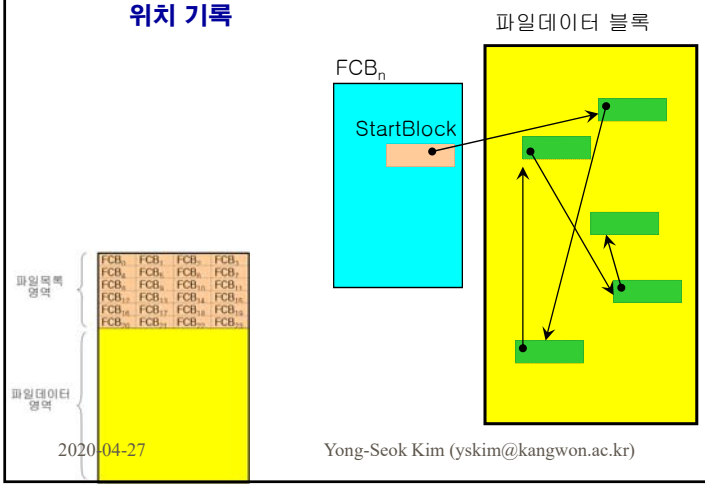
연속된 블록 할당

- ✓ Contiguous allocation
- ✓ 하나의 파일에 속한 데이터 블록들은 연속된 것이어야 함
- ✓ FCB에는 시작 블록의 번호와 블록 개수만 기록하면 됨



연결된 블록 할당

- ✓ Linked allocation 또는 chained allocation
- ✓ 데이터 블록할당은 임의 위치의 블록들을 필요한 개수만큼 할당
- ✓ FCB에는 첫 블록의 위치만 기록, 각 데이터 블록에는 다음 데이터 블록의 위치 기록



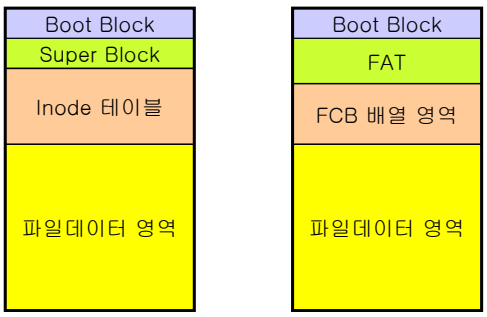
블록할당 방식들의 비교

평가 기준	인덱스 블록 할당	연속된 블록 할당	연결된 블록 할당
파일 크기의 한계	나쁨	우수	우수
임의 위치 접근 속도	우수	우수	나쁨
순차적 파일접근 성능	보통	우수	보통
디스크 공간의 활용도	우수	나쁨	우수
파일 크기의 가변성	우수	나쁨	우수
파일의 안정성	우수	우수	나쁨

㈜ 사용목적에 따라 적절한 방식 적용

파티션의 구성

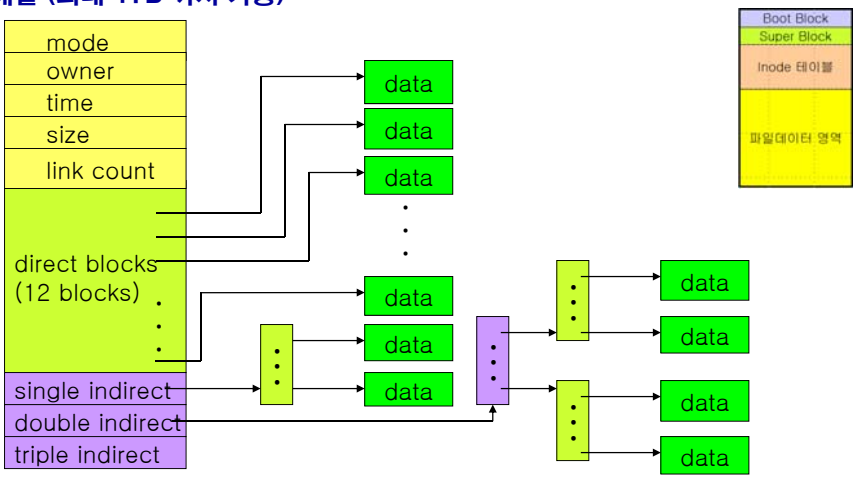
- ✓ **부트블록 (boot block)**
  - ✓ 부팅을 위한 프로그램 저장
  - ✓ 모든 파티션의 첫 블록
- ✓ **슈퍼블록 (super block)**
  - ✓ 파일시스템에 관련된 주요 내용들 (파일시스템 종류, 파티션 크기, 파일 개수, 빈 블록 목록 등



유닉스 파티션 구조 FAT 파티션 구조

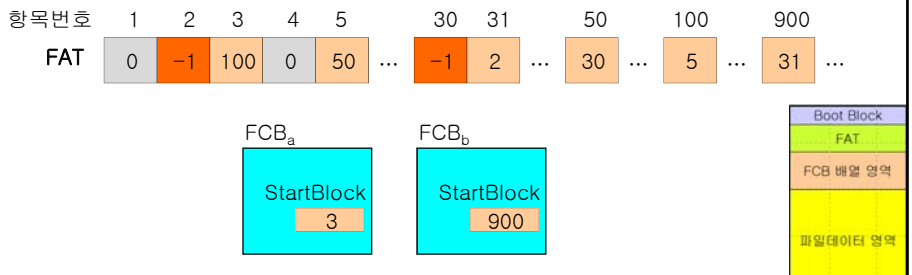
유닉스의 다단계 인덱스

- ✓ 인덱스 블록할당의 장점을 취하면서 파일크기의 제한은 다단계로 처리하여 해결 (최대 4TB 까지 가능)



윈도즈의 FAT16/32

- ✓ 연결된 블록할당의 장점을 취하면서 임의접근 속도문제를 FAT로 해결
- ✓ FAT (file allocation table)
  - ✓ 블록들 간의 연결 정보만 별도로 관리하는 부분
  - ✓ FAT를 메모리에 적재해두고 사용 ➔ 연결 정보를 빠르게 검색
- ✓ FAT32: OS 간에 호환성 높음 (Linux, OS X, Camera 등)
- ✓ FAT 블록할당 정보의 예



FAT32 의 FCB 구조

- ✓ **FCB당 32바이트**

```
struct FCB_FAT32 {
  char Name[8],Ext[3];
  char Att[3];
  short CreateTime, CreateDate;
  short Att2;
  short FirstBlockUpper;
  short ModifyTime, ModifyDate;
  short FirstBlockLower;
  long Size;
}  // 32 bytes per FCB
```


빈 블록들의 목록 관리

- ✓ **연결된 블록할당 방식 활용**
  - ✓ 모든 빈 블록들의 목록 관리
  - ✓ 초기 유닉스에서 이를 응용하여 그룹단위로 사용
- ✓ **비트맵 (bitmap 또는 bit vector) 할당**
  - ✓ 파티션의 일부분에 블록별 사용 여부를 비트맵으로 관리
  - ✓ 한 블록당 1비트에 할당 여부 표시
  - ✓ 리눅스의 Ext2나 Mac OS에서 응용하여 사용

파일시스템 마운트

- ✓ **파일시스템 마운트 (mount)**
  - ✓ 특정 파티션을 사용자가 접근할 수 있도록 등록하는 절차
- ✓ **유닉스**
  - ✓ 부팅과정에서 루트 (root) 파일시스템 지정
  - ✓ 명령어 (또는 시스템 콜 함수)로 각 파티션을 현재 트리 구조의 임의의 하부 트리로 등록
  - ✓ 사용 명령어: mount
  - ✓ 커널 내부의 mount table에 기록
- ✓ **윈도즈**
  - ✓ 장착된 파티션들을 자동으로 검사해서 등록
  - ✓ '내PC' 폴더에 고정된 'C:', 'D:' 등의 이름으로 등록

파일시스템 생성과 검사

- ✓ **파일시스템 초기화**
  - ✓ 지정된 파티션을 특정 파일시스템 종류에 맞게 초기화
  - ✓ 파일목록 영역에는 모든 항목이 빈 상태로 기록
  - ✓ 모든 파일데이터 블록들은 빈 블록으로 등록
  - ✓ 루트 디렉터리 생성
  - ✓ 유닉스의 'mkfs' (make file system), 윈도즈의 'format'
- ✓ **파일시스템 검사**
  - ✓ 파티션 내의 정보들이 파일시스템 종류에 맞게 서로 일관성이 있는지 검사하고 복구
  - ✓ 데이터 일관성 오류: 이중 등록 데이터 블록, 등록되지 않은 데이트 블록, 경로가 없는 파일 등을 검사
  - ✓ 디스크 블록 오류: 특정 블록에 쓰기/읽기가 제대로 되는지 검사
  - ✓ 유닉스의 'fsck' (file system check), 윈도즈의 'chkdsk'

파일시스템의 구현

- ✓ **picoKernel은 아니지만 구현 방법에 대한 이해를 위한 소스코드 제시**
- ✓ **파일제어블록 및 주요 변수 (프로그램 7.1)**
- ✓ **파일 생성 (프로그램 7.2)**
  - ✓ 빈 FCB를 할당하고 초기화
- ✓ **파일 열기와 접근권한 검사 (프로그램 7.3, 7.4, 7.5)**
  - ✓ 대상 FCB를 찾고, 프로세스와 FCB 정보를 바탕으로 권한 검사
- ✓ **파일 읽기와 쓰기 (프로그램 7.6)**
  - ✓ 대상 FCB의 DataBlock 정보에서 데이터 블록을 결정하고 디스크 제어기에 명령 전달
- ✓ **바이트 단위의 파일 입출력 (프로그램 7.7)**
- ✓ **파일 닫기와 위치 이동 (프로그램 7.8)**
- ✓ **장치 독립적인 입출력 처리 (프로그램 7.9, 7.10)**

바이트 단위의 입출력

- ✓ **프로세스와 디스크 제어기의 입출력 단위가 다름**
  - ✓ 프로세스: 바이트 단위로 읽기/쓰기 요청
  - ✓ 디스크 컨트롤러: 블록 단위로 읽기/쓰기 처리
  - ✓ 운영체제에서 바이트 단위의 입출력 요청을 블록단위의 입출력으로 변환
- ✓ **읽기 작업 처리 절차**
  1. 프로세스가 요청한 범위를 포함하는 블록을 커널의 버퍼로 읽어들인 후
  2. 요청한 부분만 프로세스의 읽기 버퍼 영역으로 복사
- ✓ **쓰기 작업 처리 절차**
  1. 프로세스가 요청한 범위를 포함하는 블록을 커널의 버퍼로 읽어들여서
  2. 요청한 부분만 프로세스의 쓰기 버퍼 내용으로 수정 후
  3. 해당 블록을 다시 디스크에 기록

Open 상태의 파일들에 대한 정보