**운영체제 10장 파일시스템**

2015253039 권진우

10장

**파일시스템 : Disk를 추상화 한 개념**

- 파일 : OS가 정보 저장장치의 1차원적 특성을 추상화한 논리적 저장 단위

- **보조 저장장치(Disk)에 저장된 파일 정보들의 집합** - Ex. 이름이 있음

- **OS가 물리적 저장장치에 맵핑하여 저장 및 접근(Disk에도 분할되어 파일(프로그램)을 저장)**

파일 구조(정의) **3가지**

1. 없음 : 형식을 정의하지 않음 (Word, Byte의 연속)

2. 단순 레코드 구조 : 줄, 고정길이, 가변 길이 레코드의 연속

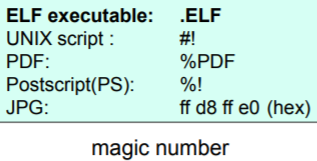
3. 복잡한 구조 : 형식이 정해진 Formatted문서

**OS에서는 파일 구조를 Byte의 연속으로 다루며 형식을 정의하지 않는다**

-> **다양한 형식의 파일(프로그램)을 실행시킬 수 있게 함**

-> BUT, 실행파일 형식은 알아야 실행할 수 있으므로 **실행을 위한 실행파일 형식만 정의** 함 (Ex. elf, exe)

Unix의 파일 유형 구분



파일 속성 : 파일 이름, 식별자, 유형, 파일의 위치, 크기 등등

**디렉토리 구조** : **파일을 체계적(계층적)으로 관리하고 찾기 위한 구조**

-> 디렉토리는 파일들에 대한 정보(이름, 식별자 등)를 갖고 있음

파일을 탐색하기 위한 파일 연산

파일의 시작 부분의 코드를 이용하여 파일 유형을 구분

1**. 기본연산** : 생성, 쓰기, 읽기, 삭제, 절단(길이가 0인 파일로 만듦)

2. **기타연산** : 추가, 복사, 이름 변경

3. **OPEN연산** : 메모리에 있는 OPEN file table에 대한 정보를 등록하고 해당 파일 인덱스 반환

**-> 파일이름을 통해 -> 디렉토리 검색 -> Open할 파일에 대한 디렉토리 항목을 Open file table로 복사해옴 -> 요청한 접근 모드(RD/RW/EO)와 파일의 접근 가능 모드 비교 -> 허용되면 Open file table에 대한 Index 값 반환**('3'부터 시작)

4. **Close연산** : Open file table의 항목을 제거(종료)

\*fopen = 파일의 포인터 반환 / \*open = 파일의 인덱스(정수) 반환

2단계 **Open file table** : **프로세스 단위 테이블 / 전체 시스템 테이블\*\*\***

\***프로세스 단위 테이블** : 프로세스의 종속된 정보 : 현재 file position pointer, access right 등등

(\*file position pointer는 현재 프로세스에서 진행 포인터 위치를 말함)

\***전체 시스템 테이블** : 프로세스의 독립(공통된) 정보 : file open count, disk location, file size 등

(\*file open count : 해당 파일이 몇개가 열려있는지 확인, 값이 '0'이 되면 제거된 것)

\*동일한 프로세스가 여러 개 열릴 수 있음(Ex. 워드 파일을 여러 개 열 때)

**\*프로세스 단위 테이블은** 같은 프로그램인데 각각 **열린 개개의 프로세스에 대한 정보**

**\*전체 시스템 테이블은** 여러 개 열려있는 해당 **프로세스(본체)에 대한 정보**

\*Ex. 워드파일이라고 치면 워드 프로세스는 전체 시스템 파일 / 각각 열려있는 Word1.docx, Word2.docx는 프로세스 단위 테이블

-OS마다 파일을 구분하는 기준이 다름(컴파일러가 구분하여 실행)

\*시스템은 많은 수의 파일을 Disk에 저장하기 때문에 체계적인 구성이 필요

- Disk와 파일 시스템

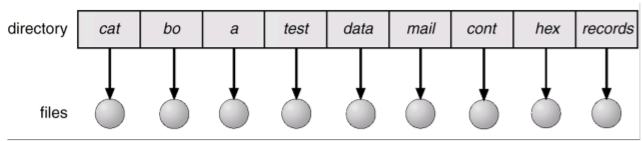
- **디스크를 여러 부분으로 분할하여 체계적 사용 -> Disk 파티션 분할**

- **파티션 별로 File System(파일 시스템)이 구성**됨 -> Directory 구조 사용하여 구성

**- Directory는 Directory 안에 있는 파일의 정보(속성)를 저장하고 있음**

디렉토리의 단계별 발전 과정 \*\*

**Single** Level Directory

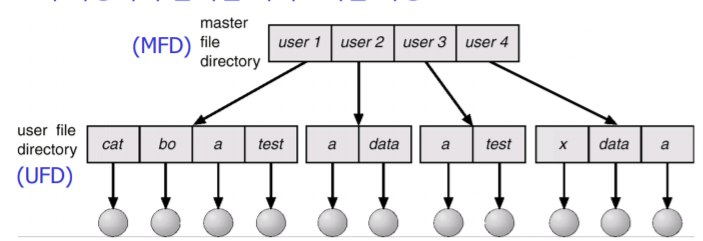


- 한 단계 깊이의 디렉토리 사용 -> 디렉토리 안에는 무조건 파일만 들어있음

- 단점 : Directory별로 같은 Directory 이름을 사용할 수 없음

종류별로 묶지(Grouping problem) 못함

**Two** Level Directory

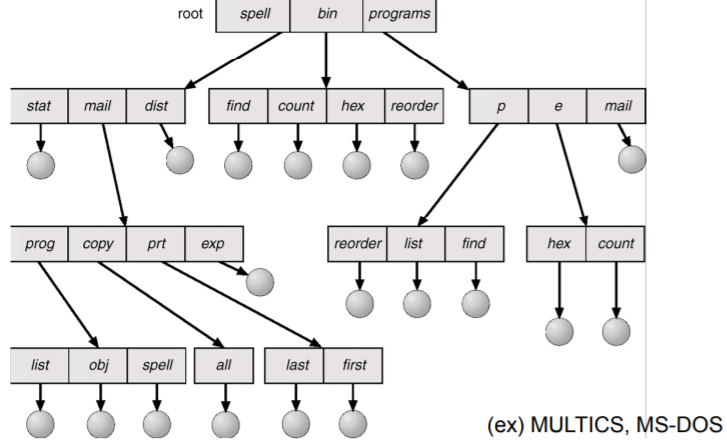


- 모두 분리된 디렉토리 사용

- 경로이름 예시 : /User/Test file

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

효율적인 파일 탐색(2단계만 거치면 모든 파일 탐색 가능)

- 단점 : **Grouping 단계가 2단계로 제한적이다**.

**Tree-Structure** Directory -->

- 트리구조의 디렉토리

- 효율적 파일 탐색, Grouping 무한

- 각 프로세스는 현재 디렉토리를 가짐

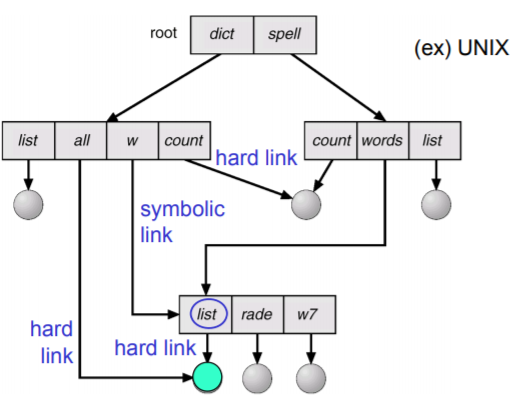
- 파일 이름만 사용하여 디렉토리 검색 가능

현재 많이 쓰는 Directory 구조

**Acyclic-Graph Directory <시험\*\*\*\*\*\*\*\*\*\*\*\*>**

- **단방향성 그래프 구조**의 디렉토리 ----->(단방향)

- **디렉토리 간 파일과 서브디렉토리 공유 허용**

-***Hard link 와 Symbolic link 구분\*\****

- Symbolic link는 바로가기와 같이 **경로를 가르키는 link, 특정 파일의 위치(모든 경로?)와 정보만 기록(원본의 변경 여부와는 관계없는 독립적)한 파일(링크)**

- Symbolic link의 **장점** : **상위 Directory단계에서 바로 찾아가서 수행이 가능, 디렉토리를 하나하나 찾아가는 번거로움X**

- Hard link : 프로그램(파일)을 직접 가르킴

**-Symbolic link : 다른 파일/디렉토리에 대한 포인터 저장 = Indirect link**

**-Hard link : 저장 장치의 파일 위치 직접 저장 = Direct link**

\* 따라서 같은 파일이 **여러개의 경로를 통해서 접근** 될 수 있음(여러개 경로를 가질 수 있음)

-> Aliasing

-> 전체 파일 순회 시, **중복탐색 문제 발생**(경로가 여러개여서)

*공유 파일에 대한* ***삭제***

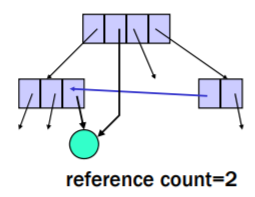
-**Hard link 경우** : **직접 링크를 삭제**, 실제 파일을 가르키고 있는 Reference Count = '-1'

**Count = 0**(아무도 자신<파일>을 가르키지 않음)이 되면 **실제 파일이 삭제** 됨

Reference Count를 끊고 -1을 하며 Count값이 0이 될 때 아무도 가리키지 않는 다는 뜻이므로 해당 파일을 삭제시킨다.

-**Symbolic link 경우** : **Symbolic link만 삭제** -> 링크된 원본 파일이 삭제 되면 Symbolic link는 존재하지 않는 파일을 계속 가르키고 있음

(삭제 작업이란? **디렉토리에서 실제파일을 가리키는 연결 링크(hard link?)를 끊음** -> 파일 정보는 남아있음)



\* 디렉토리 : 파일을 체계적으로 관리하기 위한 도구(구조)

Symbolic link

\* 파티션 별로 나눠서 서로 다른 파일 시스템(FAT32, NTFS)을 쓸 수 있다.

\* 파일 시스템을 포맷()하여서 사용

-> 파일 시스템 프로그램을 거쳐서 Disk 사용

-> 프로그래밍이 쉽지만 속도 저하

\* 파일 시스템을 포맷하지 않고(Raw) 사용

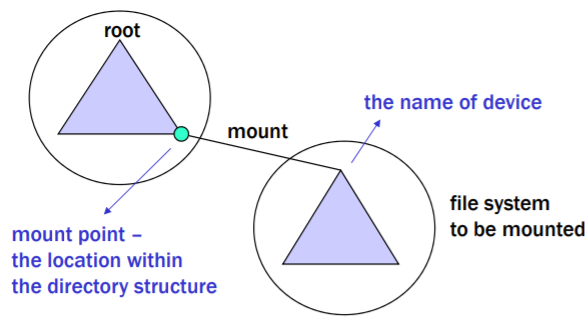
-> 파일 시스템을 거치지 않고 Disk사용 -> 프로그래밍이 어렵지만 Disk 빠른 접근 가능

**파일 시스템 마운트** : 새로운 장치를 파일시스템에 접속 시키는 것

- Disk + USB(마운팅) => 효과 : 파일 시스템을 확장시키는 효과를 가져온다.

- **마운팅된 디바이스(ex.USB)가 디바이스 이름과 Mount point(빈 디렉토리)를 제공**

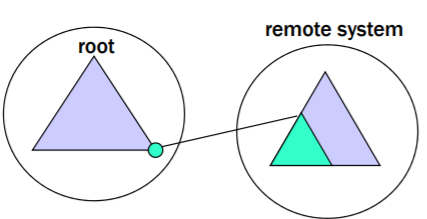
-> OS는 디바이스를 **검사** -> 지정된 **Mount point에 마운트**됨으로써 Mounting 완료



파일 마운트의 개념을 확장하여 File 공유/ 원격 지원이 가능해짐

- 보호 방식을 이용하여 공유가 이루어 질 수 있음

**원격 파일 시스템**



요청

**- 원격 마운팅을 통해 파일 시스템의 일부를 Mount Point부터 원격으로 사용토록 내어줌(상대방의 시스템의 허가를 받았을 시)**

- NFS, DFS등을 이용하여 Client - Server 모델을 사용하여 구현

- NFS : 표준 Unix 파일공유 프로토콜

- CIFS : 윈도우 파일공유 프로토콜

\*랜섬웨어가 몇 년 전 CIFS의 취약점(접근 허용에 대한 방어취약)을 이용하여 원격 시스템 조종문제

**-디스크의 구조**

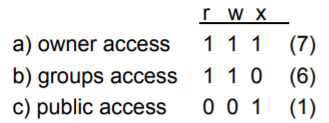
- 디스크는 여러개의 파티션으로 분할 사용 가능(Raw + Formatted 여러개도 가능)

- 디스크는 여러개를 묶어서 RAID와 같은 방식으로도 사용 가능

- 볼륨 : 파일 시스템을 포함한 개체

보호 : 파일 공유(원격)를 위해 **접근 제어를 통한 보호가 필요**

- 접근 제어 : 파일에 대한 접근 권한을 제어



111 : 7(2진수로 표현)

110 : 6

001 : 1

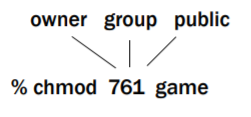
누가? 주인, 그룹내 사용자, 공용 사용자 등으로 예를 들어 나눌 수 있음

무엇을? r : read 권한 / w : write 권한 / x : execute 권한

(**'1' : 권한 O** / **'0' : 권한 X**)

\*윈도우 : 사용자별로 권한 지정 가능

\*리눅스 : 기본적으로는 오너, 그룹, Public사용자로 구분

\*권한 지정하는 법 in Linux