운영체제 13장
File-System Interface
파일 시스템 인터페이스

0. File Concept
1. Access Methods
2. Disk and Directory Structure
3. File-System Mounting
4. File Sharing
5. Protection

목표
-file systems의 기능
-ile systems을 proc 사용자가 어떻게 access 하는지
-file-system을 구현하는 방법의 장단점(시스템 환경, 애플리케이션 특성마다 잘 선택해서)
<구현 내용: access methods, file sharing, file locking, directory structures 등>
-file-system protection

0. File Concept
연속적인 공간(page)을 제공하는 것
(proc도 각각 메모리에 로딩될 때, 연속적인 메모리 공간을 사용하는 것처럼 느낌)
특성
-Contiguous(연속적) logical address space(사실, HDD의 블록으로 흩어져 존재하며, file table의 index에 매핑되어 연속적인 것처럼)
-Data(numeric, char, binary), Program 등 다양한 파일 타입 존재

(1) File Attributes
파일을 관리하기 위한 파일 테이블
-다양한 파일 정보가 담겨져 있음
Name – only information kept in human-readable form
Identifier – unique tag (number) identifies file within file system
Type – needed for systems that support different types
Location – pointer to file location on device
Size – current file size
Protection – controls who can do reading, writing, executing
Time, date, and user identification – data for protection, security, and usage monitoring
파일에 대한 정보는 directory structure(tree 형태) 내에 저장되며 이는 disk에서 유지됨

(2) File Operations
다양한 file operation 존재 >> OS에서 sys.call로 제공함

File is an **abstract data type**

Create

Write – at **write pointer** location

Read – at **read pointer** location

Reposition within file - **seek**

Delete

Truncate

***Open*(F_i)** – search the directory structure on disk for entry F_i, and move the content of entry to memory

Close (F_i) – move the content of entry F_i in memory to directory structure on disk

seek: read, write할 파일의 position(위치)을 지정하는 것

Truncate: 파일을 일정한 크기로 잘라냄

*Open(Fi)-생성된 파일에 read, write 전, 열어보는 것(file 빨리 access 위한 준비 작업)

Fi 엔트리에 대한 directory 구조를 찾고 그 엔트리를 메모리에 미리 이동시키는 것

*Close(Fi)-이제 access 안하므로, 앞서 open 때 준비했던 작업을 삭제하는 과정

(3) Open Files(생성된 파일에 read, write 전, 열어봐야 됨)

-Open-file table: 파일을 open할 때마다 생성됨

같은 파일 여러 번 open되도 독립적인 Open-file table 생성됨

-File pointer: 최근 read/write location를 가리키는 포인터,

-File-open count: 자신을 ref하는 것을 세는 counter

(open-file table도 file table을 ref함, open-file table, file table 모두 가지고 있음)

>> 마지막 proc들이 ref를 닫을 때, open-file table로부터 data 삭제 허가함

proc-> open-file table -> file table [ref 관계]

(P, C는 Open-file table에 대해서도 똑같이 copy하여 같은 파일을 가리킴)

(4) Open File Locking

Locking 매커니즘(몇 OS나 F.S에 의해 제공됨)

#reader-writer locks와 유사(Proc 동기화 때, C.S 내에서 발생하는 것처럼)

-Shared lock: reader lock과 유사(여러 reader가 공유해서 read 계속 가능)

-Exclusive lock: writer lock과 유사(독점권 부여, 다른 reader-writer접근 X)

#Mandatory or advisory:

Mandatory: lock이 열려있을 때, access 거부되고 요청하고 대기

(proc 동기화 때 이야기하던 것)

Advisory: proc가 lock의 상태만 확인하고 무엇을 할지 결정

File Locking Example - PPT 10,11p

(5) File Structure

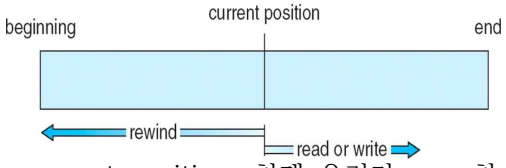
-None: sequence of words, bytes(아무 구조 없는 것)

-Simple record structure(레코드, 아이템 단위로 되어 있는 것)

아이템 종류 - Lines, Fixed length, Variable length

-Complex Structures(복잡한 구조)
Formatted document: 정해진 포맷 존재
Relocatable load file: 실행 파일 같이 내부적으로 어디가 text, data인지 나타내는 것
간단한 구조의 file형태 -> 복잡한 file 형태 구현 가능

1. Access Methods
파일을 access하는 방법(file 우선 open 후 read, write함)



*current position: 현재 우리가 open한 파일에 대해 access하는 위치이다.
current pos로부터 순차적으로 **read or write**함(seek 통해 특정 pos에 대해서 가능)

-Sequential Access : 순차적 access
-Direct Acess : 어떤 position 지정(lseek)해 다음 read, write 진행
(Random access, 이후 seq access도 진행 가능)

#Other Access Methods
-Index Access : index 구조를 통해 타겟 파일에 access
(자주 사용하여, memory에 cache하기도 함)
*index는 빨리 access 위해 파일에 포함되어 HDD에 존재하는데, index가 너무 큰 경우 index의 index를 가질 수 있음(ex. ISAM-index레벨 여러 개 존재)

-Index and Relative Files(VMS OS에서 제공)
index 파일이 따로 존재하고, relative file에 본 data를 저장하여 access하는 방식

2. Directory Structure & Disk Structure
(1) Directory Structure
디렉토리는 자신 아래에 존재하는 파일들의 정보를 저장하고 있음(index의 일종)
-Directory 저장 내용 : file name + inode(file에 대한 unique한 ID)
-각 file마다는 file table에서 더 자세한 정보 저장하고 있음

(2) Disk Structure
-paritions: HDD를 쪼개어 사용하는 한 조각
-RAID: 병렬 처리(striping에 의해 빨리 access할 수 있고 parit bit, ecc, erc bit를 추가함으로써, failure 탐지 및 복구 가능)
*Disk나 parition은 F.S없이 raw하게 사용 가능하고, F.S이랑 formatted하게 사용 가능
-volume: F.S과 parition 사이 중간에 logical한 parition을 둔 것
✓ volume은 partition 하나 이상을 가질 수 있음(일반적으로 1:1 매핑)
-volume table: volume과 disk parition을 관리

-special-purpose file system: 파일 이외의 proc, memory 등의 것들도 일반 F.S처럼 쉽게 access할 수 있는 환경 제공하는 것 (<=>general-purpose file system)

(3) Types of File Systems
-general-purpose file system
ufs, zfs 등은 파일을 보관하기 위한 용도

-special-purpose file system
나머지(procfs, tmpfs)는 proc에 대한 정보, 커널 메모리의 인터페이스 등 특별한 목적 존재
✓ F.S을 access하듯이 쉽게 OS의 다른 리소스도 access할 수 있도록 하는 것

#Operations Performed on Directory

Search for a file

Create a file

Delete a file

List a directory

Rename a file

Traverse the file system

(4) Directory Organization
디렉토리 구조 설계 목적
-효율성: 파일이라는 자원을 효율적으로 관리
-Naming: 사용자가 쉽게 이용할 수 있음
-Grouping: 파일을 특성, 가치에 따라 논리 집합으로 구분 가능

(5) File Structure
-Single-Level Directory
자원의 제약이 많은 곳에서 사용하는 File의 구조

-Two-Level Directory
Single-Level보다 풍부한 기능이 존재하지만, 역시 자원의 제약이 많은 특별한 환경에서 사용하는 File 구조

-Tree-Structured Directories
일반적으로 사용하는 Tree 형태의 File 구조이며, level을 자기가 원하는 만큼 만들어 파일을 Grouping하여 Tree형태 구성
장점: Efficient searching, Grouping Capability(파일 유형별로 그룹), Current directory (=working directory, 트리 구조에서 자신이 지금 access하고 있는 위치를 표시해줌)

#Tree-Structured Directories(Cont.)
Absoulte path: root부터 시작하여 파일 경로 표시
Relative path: current directory부터 시작하여 파일 경로 표시

-Acyclic-Graph Directories
일반적 Tree구조는 root로부터 시작해서 가지가 뻗어나가고, leaf 끼리는 붙어있을 수 없다.
✓ 그러나, Acyclic-Graph Directories는 leaf를 공유하고 directory 밑에 child로 이어가도 cycle이 형성되지 않음
*Tree를 공유할 수 있음
*Two different names (aliasing)
*dangling pointer: tree구조에서 어떤 subtree가 연결이 끊어진 상태
✓ 해결책: searching을 통해 연결이 끊어진 부분 삭제함
New directory entry type
*Link: 어떤 존재하는 파일에 대해, 양쪽에서(또 다른 ptr가) access하는 것
*Resolve the link: Link된 파일을 ptr따라 찾는 것

-General Graph Directory
Acyclic-Graph와 다른 점은 child에서 Parent로 가리키는 ptr 존재(loop형성 가능)

*어떻게 우리가 cycle이 없음을 보장하는가?
Garbage collection(root로부터 search해서 도달하지 못하는 것 삭제)

3. File System Mounting
Mount: F.S을 사용할 수 있도록 root나 root 하위 요소에 연결하는 과정
Mount Point: Mounting을 했을 때 연결되는 지점

4. File Sharing
여러 사용자가 파일을 공유할 수 있는 것

-멀티 유저 시스템에서는 protection 기능을 통해 나, 같은 그룹, 다른 사람들은 어떤 권한을 가지고 있는지 protection 기능을 설정할 수 있음(ex. rwxrw-r--)
✓ 어떤 file에 대해 누가 어떤 operation 가능 유무 표시(모든 OS는 file에 prot 기능 O)

-파일 공유는 인터넷, 분산 환경에서도 이루어질 수 있음 ex. NFS
-User ID, Group ID를 통해 누군지를 인증하고 인증받은 사람에 한해 file 공유 권한 부여

(1) Remote File Systems(File access 허용권 필수)
-3가지 단계(Manually, Automatically, Semi automatically) 존재
Manually: FTP를 이용해 어떤 호스트에 연결해 파일을 GET, PUT을 통해 가져오는 것
Automatically: 사용자는 local에 있는지 remote에 있는지 분간이 되지 않는 형태
✓ 분산FS, RFS의 경우, local에 마운트 시켜놓고 local에 있는 것처럼 사용

Semi automatically: WWW을 통해 File을 access할 수 있는 형태

-Client-server model
NFS(UNIX 표준), CIFS(Windows 표준)들은 서버에서 F.S을 제공하고 Client들은 이 F.S을 마운트하여 마치 local에 있는 것처럼 사용

-File Sharing의 Failure Modes
파일 공유에 두 가지 상태 존재: State, Stateless(파일 access하는 client의 정보 저장 유무)
stateful: open -> r/w -> close(connection oriented)
stateless: r/w(connectionless) <- recovery에 쉽지만 security 약함

-File Sharing의 Consistency(연속성) Semantics
어떤 File에 대해 여러 proc가 access하거나 여러 번 copy가 진행될 때, 서로 간 R/W하는 순서에 대해 합의를 한다는 것

5. Protection
누가 어떤 File에 대해 어떤 operation이 가능한가?

Types of access
- **Read**
- **Write**
- **Execute**
- **Append**
- **Delete**
- **List**

-Access Lists and Groups
Unix나 Linux는 protection 기능이 매우 simple

			RWX
a) **owner access**	7	⇒	1 1 1
b) **group access**	6	⇒	RWX 1 1 0
c) **public access**	1	⇒	RWX 0 0 1