**CHP10. 파일 시스템 인터페이스**

**10.1 파일 개념**

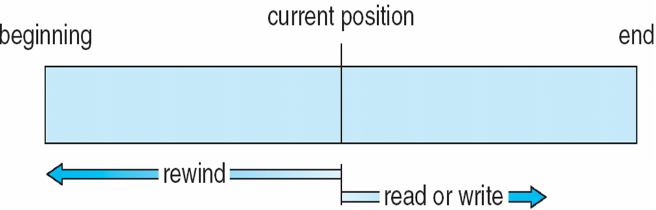
* 파일의 개념
  + 논리적 저장 단위
  + 비휘발적 성질
  + 보조 저장장치에 기록된 관련 정보의 집합
  + 운영체제에 의해 물리 장치로 사상됨
  + 사용자 관점 : 논리적 보조 저장 장치의 가장 작은 할당 요소
  + Type  
    ⬝ Data (자료) : 숫자, 문자, 이진수  
    ⬝ Program
  + 파일 내의 정보는 작성자에 의해 정의  
    ⬝ text file  
     : 라인들로 구성되는 연속된 문자 들  
    ⬝ source file  
     : 선언문 다음에 실행 문장들로 구성  
     된 서브루틴이나 함수들의 집합체  
    ⬝ 목적 파일  
     : 시스템의 링커가 인식 가능한  
     블록으로 구성된 연속된 바이트  
     들의 집합체  
    ⬝ 실행 가능 파일  
     : 적재기가 메모리로 가져와 실행  
     시킬 수 있는 연속된 코드 부분들
* 파일 속성
  + 이름 : 기호형 파일 이름은 사람이 읽을 수 있는 형태로 유지된 유일한 정보
  + 식별자(Identifier) : 통상 하나의 숫자로 파일 시스템 내에서 파일을 호가인 / 사람이 읽을 수 없음
  + 유형 : 여러 유형을 제공하는 시스템을 위해 필요
  + 위치 : 파일이 존재하는 장치와 그 장치 내의 위치에 대한 포인터
  + 크기 : 파일의 현재 크기와 최대 허용   
      
      
      
    가능한 크기
  + 보호 : 접근 제어 정보는 누가 읽기, 쓰기, 실행 등을 할 수 있는가 제어
  + 시간, 날짜, 사용자 식별 : 생성, 최근 변경, 최근 사용 등을 유지 / 보호, 보안 및 사용자 감시를 위해 사용
  + 파일에 대한 정보는 보조 젖아 장치에 상주하는 디렉토리 구조에 의해 유지
* 파일 연산 (Operations)
  + 파일 : 추상적인 데이터 유형
  + 운영체제는 파일 생성, 쓰기, 읽기, 등을 하기 위한 System call 제공
  + Create  
    ➀ 파일 시스템 내에서 공간 찾기  
    ➁ 새로 생성된 파일에 대한 항목이   
     디렉토리에 만들어져야 함
  + Write  
    ⬝ 파일 이름, 파일에 기록될 정보  
     명시하는 시스템 호출 실행  
    ⬝ 파일 내의 다음 쓰기 일어날 위치를   
     가리키는 write pointer 유지
  + Read  
    ⬝ 파일과 파일이 읽혀져야 할 블록의  
     위치를 기술하는 시스템 호출 실행  
    ⬝ 파일 내의 다음 읽기 일어날 위치를  
     가리키는 read pointer 유지  
    -> 현재 파일 위치 포인터  
     : 읽기, 쓰기 연산 모두 사용
  + Reposition within file (위치 재설정)  
    = seek  
    ⬝ 디렉토리에서 적합한 항목을 탐색  
     하고, 현재 파일 위치를 주어진  
     값으로 설정
  + Delete  
    ➀ 지명된 파일 디렉토리에서 탐색  
    ➁ 해당 파일이 차지한 공간 방출  
    ➂ 디렉토리 항목 삭제
  + Truncate (절단)  
    ⬝ 파일 내용은 지우고,  
     속성은 그대로 남기기를 원할 때  
    ⬝ 파일의 길이를 제외한 모든 속성  
     그대로 유지  
    ⬝ 파일 길이 0으로 재설정,  
     파일이 가지고 있던 공간 해제
  + Open(Fi)  
    ➀ 파일 이름(Fi)을 취하여  
     그 파일 이름으로 디렉토리 검색  
    ➁ 디렉토리 항목을 오픈 파일의  
     테이블로 복사  
    ⬝ 오픈 파일 테이블의 항목에 대한  
     포인터를 리턴  
    ⬝ 포인터를 입/출력 연산에 사용  
     -> 시스템 호출 인터페이스 단순화
  + Close(Fi)  
    ⬝ 메모리의 디렉토리 항목을 디스크의   
     디렉토리 구조로 이동
* Open Files
  + Open-file table  
    ⬝ 모든 열린 파일에 대한 정보를 가짐  
    ⬝ 임의의 파일 연산이 요구되면,  
     테이블에 대한 index로 파일 지정  
     -> 어떠한 탐색도 필요 x
  + File pointer  
    ⬝ 가장 최근의 read/write 위치 추적  
    ⬝ 각 프로세스에 대해 유일하게  
     하나씩 만들어짐  
     -> 파일 속성들과는 분리되어 유지
  + File-open count  
    ⬝ 파일의 총 열기와 닫기 수 추적  
    ⬝ 값이 0에 도달하면,  
     시스템은 항목 제거 가능  
     ->시스템은 Open-file table 항목  
     제거 전에 마지막 파일이 닫히기를   
     기다려야 함
  + Disk location of the file  
    ⬝ 메모리에 저장됨  
     ->매 연산마다 디스크에서 읽는  
     것을 피하기 위함
  + 접근 권한   
    ⬝ 각 프로세스는 한 파일을  
     하나의 접근 모드로 엶  
    ⬝ 프로세스 별 테이블에 저장 됨
* Open File Locking
  + 몇몇 운영체제는 Open file을  
     잠금 할 수 있는 기능 제공
  + 여러 프로세스에 의해  
    공유되는 파일들에 유용  
    ex) 시스템 로그 파일
  + Shared lock (공유 잠금)  
    ⬝ reader lock과 유사  
    ⬝ 여러 프로세스가 동시에  
     잠금 획득 가능
  + Exclusive lock (배타적인 잠금)  
    ⬝ writer lock과 유사  
    ⬝ 한 번에 한 프로세스만 잠금 획득
  + Mandatory vs Advisory
    - Mandatory (강제적)  
      🢭 프로세스가 배타적 잠금을 획득  
       하면, 다른 프로세스가 잠겨진   
       파일에 접근하는 것을 막음  
      🢭 잠금 무결성 보장  
      🢭 Windows
    - Advisory (권고적)  
      🢭 프로세스는 잠금 상태를 찾아서,  
       수행할 작업 결정 가능  
      🢭 다른 프로세스가 잠겨진 파일에   
       접근하는 것을 막지 않음   
      🢭 파일에 접근하기 전에,  
       잠금을 획득 할 수 있어야함  
      🢭 잠금이 적절히 획득되고, 해제  
       되는 것은 개발자의 몫  
      🢭 Unix
* 파일 유형

|  |  |  |  |
| --- | --- | --- | --- |
| 파일 유형 | 일반적 확장자 | 기능 |  |
| 실행 가능 | exe, com, bin, 없는 경우 | 실행 가능 기계어 프로그램 |  |
| 목적 | obj, o | 연결되지 않는 상태의 컴파일된 기계 언어 |  |
| 소스 코드 | c, cc, java, pas, asm, a | 다양한 언어로 된 소스 코드 |  |
| 배치 | bat, sh | 명령어 해석기에 대한 명령 |  |
| 문서 | txt, doc | 텍스트 자료, 서류 |  |
| 워드프로세서 | wp, tex, rtf, doc | 다양한 워드프로세서 형식 |  |
| 라이브러리 | lib, a, so, dill | 프로그래머들을 위한 라이브러리 루틴 |  |
| 인쇄나 view | ps, pdf, jpg | 인쇄 출력용 파일로서 ASCII 혹은, 이진 파일 형식 |  |
| 백업, 보관 | arc, zip, tar | 1.관련된 파일들을 하나로 묶어서 보관 2.저장을 위해 압축 가능 |  |
| 멀티미디어 | mpeg, mov, rm, mp3, avi | 오디오나 비디오 정보를  포함하는 이진파일 |  |

* + 운영체제 설계 시 중요한 사항 중 하나는, 운영체제가 파일 유형들을 인식하고 지원할 것인지를 결정하는 것
* File Structure (파일 구조)
  + 각 파일들은 그 파일을 다루는 프로그램에 의해 인식 가능한 내부 구조를 일정한 형태로 가짐
  + None  
    ⬝ 단어의 순서, bytes
  + Simple record structure (단순기록구조)  
    ⬝ Lines  
    ⬝ Fixed length (고정 길이)  
    ⬝ Variable length (가변 길이)
  + Complex Structures (복잡한 구조)  
    ⬝ 형식화된 문서  
    ⬝ Relocated load file  
     (재할당 가능한 load 파일)
  + 적절한 제어 문자를 사용하여,  
    None 방법으로 나머지 두 개 방법  
    시뮬레이션 가능
  + Who decides  
    ⬝ 운영체제 / 프로그램

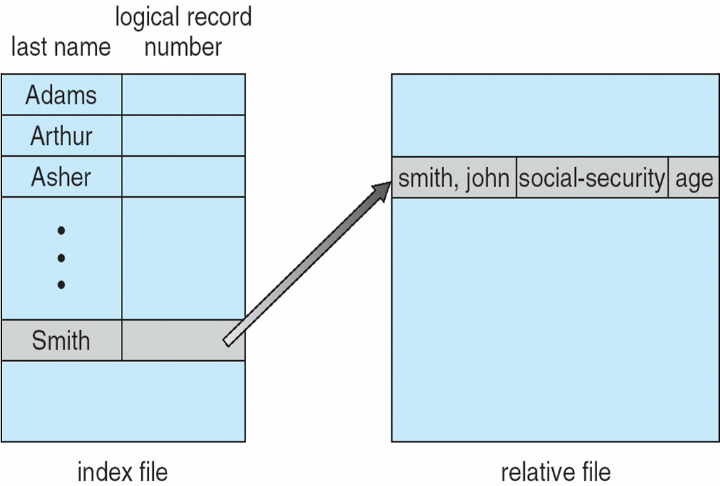
**10.2 접근 방법**

* 순차 접근 (Sequential Access)



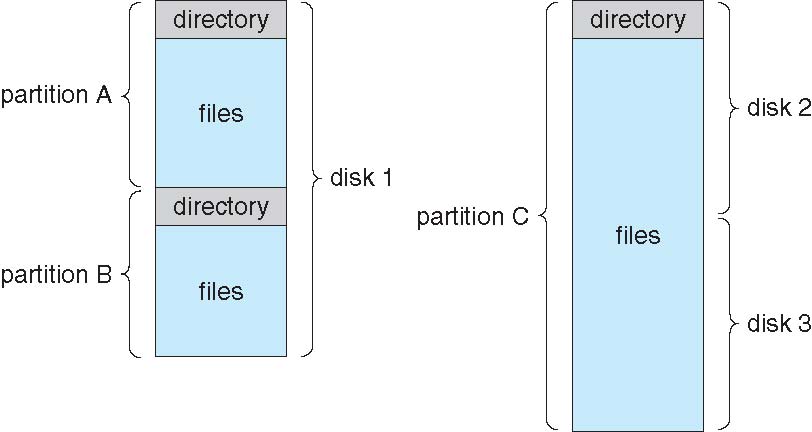
* + 저장되어 있는 레코드 순서로
  + Read : 파일 포인터(offset) 자동 증가
  + Write : 새로운 파일의 끝으로  
     파일 포인터 이동
* 직접 접근 ( Direct Access)
  + 파일 : 고정 길이의 논리 레코드 집합  
     : 번호를 갖는 일련의 블록
  + 어떤 블록이라도 직접 접근 가능
  + 읽기/쓰기의 순서에 제약 x
  + 대규모의 정보를 다루는데 유용
* Access Methods

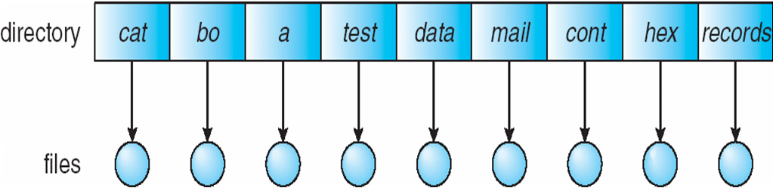
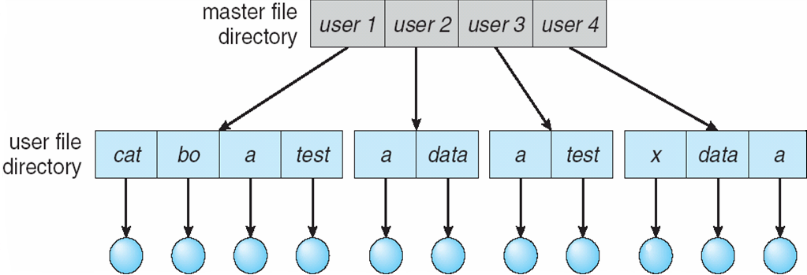
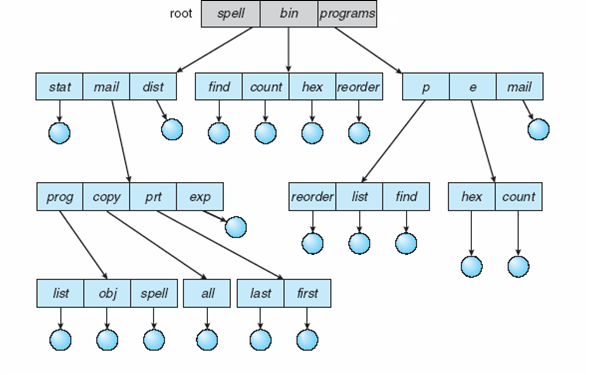
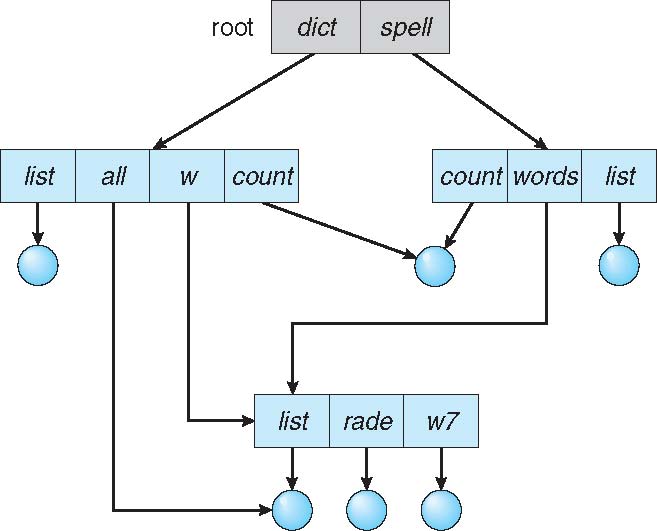
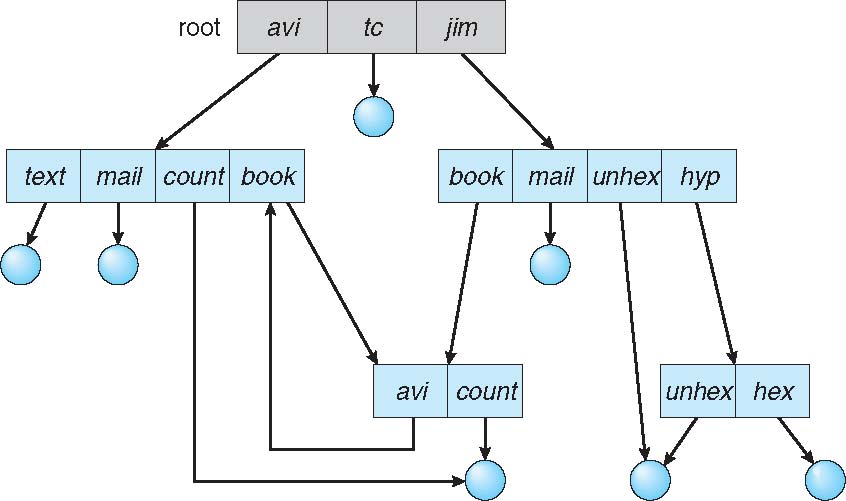
|  |  |
| --- | --- |
| 순차 접근 | 직접 접근에서 순차 접근 구현 |
| Reset | Cp = 0; (cp : current point) |
| Read next | Read cp;  Cp = cp +1 |
| Write next | Write cp;  Cp = cp+1 |

* Direct Access   
  ⬝ read n / write n  
  ⬝ position to n read next  
   write next  
  ⬝ rewrite n   
  ⬝ n : 상대 블록 번호  
   -> 파일의 시작을 0으로 보고  
   계산한 레코드의 위치
* 기타 접근 방법
  + 직접 접근 파일이 있으면, 그것을 기반으로 다른 접근 방법 제공
  + 파일에 대한 index 사용
  + 접근할 레코드의 index를 찾아 그에 대응하는 포인터를 얻음  
    -> 포인터를 사용하여 파일을 직접 접근하고, 원하는 레코드 찾음
  + 파일이 너무 크면, index가 너무 커서 메모리에 다 들어가지 못함  
    -> index자체를 파일로 만들어야 함
  + Index파일이 너무 커지면,  
    그것에 대해 또 index를 만듦
  + Primary index file  
    : 이차 색인 파일의 포인터를 가짐
  + IBM의 ISAM  
    (Indexed Sequential Access Method)  
    ⬝ 크기가 작은 master index를 메모리  
     에 유지, master index가 이차 색인   
     디스크 블록을 가리키도록  
    ⬝ 이차색인이 실제 파일블록을 가리킴  
    ⬝ 파일은 키 갑에 따라 순서대로 유지  
    ⬝ 특정 항목 찾기  
     : master index에서 이진 탐색을 통해   
     이차 색인 블록 번호 알아내기  
     -> 이 블록을 읽은 후, 이진 탐색을  
     통해 원하는 레코드 찾기  
     -> 이 블록을 순차적으로 탐색  
    ⬝ 모두 OS에서 이루어짐
  + VMS index 와 상대 파일  
    (VMS : Virtual Memory System)

**10.3 디렉토리 구조**

* Disk Structure
  + 디스크는 파티션으로 세분화 가능  
    ⬝ 파티션=슬라이스=미니 디스크
  + RAID를 오류로부터 보호 가능  
    ⬝ RAID – 복수 배열 독립 디스크  
    (Redundant Array of Independent   
     Disks)  
     : 여러 개의 하드 디스크에 일부   
     중복된 데이터를 나눠서 저장
  + 파일 시스템으로 일부분을 사용하고 나머지 부분은 스왑 공간이나 포맷되지 않는 디스크 공간 같은 것들로 사용
  + Volume  
    ⬝ 파일 시스템을 지닌 일련의  
     저장소  
    ⬝ 논리적인 가상 디스크  
    ⬝ 한 개 이상의 시스템이 부팅  
     되고 실행될 수 있도록 여러 개  
     의 운영체제 저장 가능  
    ⬝ 시스템에 존재하는 파일에 대한   
     정보를 가지고 있어야 함  
    ⬝ 정보는 device directory 또는  
     컨텐츠 볼륨 테이블에 저장  
    ⬝ device directory  
     : 그 볼륨에 있는 모든 파일에   
     대한 이름, 위치, 크기, 유형  
     등의 정보 기록



* Directory Overview
  + Directory  
    ⬝ 파일 이름을 그 위치로 바꾸어 주는   
     심볼 테이블
* 디렉토리에 대한 연산
  + Search for a file (파일 찾기)
  + Create a file (파일 생성)
  + Delete a file (파일 삭제)
  + List a directory (디렉토리 나열)  
    ⬝ 디렉토리에 존재하는 파일 나열  
    ⬝ 파일에 대한 디렉토리 항목의  
     내용을 보여줌
  + Rename a file (파일의 재명명)
  + Traverse the file system  
    (파일 시스템의 순회)  
    ⬝ 파일 시스템의 여러 디렉토리를 순회하면서 파일들을 볼 수 있음  
    ⬝ 신뢰성 측면을 고려하면,  
     파일 시스템의 구조와 관련된  
     정보를 주기적으로 저장 (백업)  
    -> 시스템 고장에 대비  
    -> 백업한 디스크 공간은  
     다른 목적으로 전용 가능
* 디렉토리 조직화
  + 디렉토리는 논리적으로 구성되어 있음
  + 효율성 : 빠른 파일 탐색 가능
  + 이름 지정 : 사용자에게 편리함  
    ⬝ 두 명의 사용자는 다른 파일에 대해   
     동일한 이름을 가질 수 있음  
    ⬝ 같은 파일은 여러 다른 이름이 있을   
     수 있음
  + 그룹화 : 속성별로 논리적인 그룹화  
    ex) 모든 java programs, 모든 게임, ..
* 1단계 디렉토리 (Single-Level Directory)
  + 
  + 모든 파일이 한 개의 디렉토리 밑에 있는 구조
  + Naming problem  
    ⬝ 같은 디렉토리에 모든 파일이  
     존재하므로 각 파일들은 유일한   
     이름을 가져야 함
  + Grouping problem
* 2단계 디렉토리 (Two-Level Directory)  
    
    
    
    
    
    
    
  + 각 사용자는 자신만의 UFD (User File  
    Directory)를 가짐
  + UFD 각 디렉토리에는 한 사람의 파일만을 저장
  + 시스템은 MFD(Master File Directory)를 먼저 탐색
  + MFD는 사용자 이름이나 계정 번호로 index, 각 항 복은 그 사용자의 UFD를 가리킴
  + 다른 user라면, 같은 파일 이름 가능
  + 효율적인 검색
  + 그룹화 기능 X
  + 높이 2의 트리로 간주  
    ⬝ root : MFD  
    ⬝ root의 child : UFD  
    ⬝ leaf node : 파일
  + Path name : 사용자 이름+파일 이름
* 트리구조 디렉토리(Tree-Structed Directory)  
    
    
    
    
    
    
    
    
    
    
  + 일반 사용자들에게  
    자신의 서브 디렉토리 생성을 허용
  + 시스템 내의 모든 파일은 고유한 경로이름을 가짐
  + 디렉토리  
    ⬝ 서브 디렉토리, 파일 가질 수 있음  
    ⬝ 모든 디렉토리는 내부적으로  
     똑 같은 구조
  + 한 비트를 통해 디렉토리 여부 확인  
    ⬝ 일반 파일(0) / 디렉토리(1)
  + 효율적인 검색
  + 그룹화 가능
  + Current directory (working directory)
  + 절대 경로 이름 (Absolute path name)  
    : 루트에서부터 지정된 파일까지의  
     경로가 명시된 것
  + 상대 경로 이름 (Relative path name)  
    : 현재 디렉토리를 기준으로 목적하는   
     파일까지의 경로
  + 새 파일을 생성하는 것은  
    현재 디렉토리에서 수행
  + 파일 삭제 : rm <file-name>
  + 현재 디렉토리에서 서브디렉토리 생성  
    : mkdir <dir-name>
* 비순환 그래프 디렉토리  
  (Acyclic-Graph Directory)  
    
    
    
    
    
    
    
    
    
    
    
    
    
  + 디렉토리들이 서브디렉토리들과 파일들을 공유할 수 있도록 허용
  + 똑 같은 파일이나 서브디렉토리가 서로 다른 서브디렉토리에 있을 수 있음
  + 공유를 위해 복사 x
  + 다른 파일 이름이 같은 파일을 가리키는 경우 -> aliasing
  + 공유 파일을 누구든지 삭제할 경우, 그 파일을 제거  
    -> dangling pointer (존재하지 않는   
     파일을 가리키는 포인터) 존재  
    -> solutions  
     ⬝ Backpointers,  
     모든 포인터들을 삭제  
     ⬝ Backpointers using a daisy chain   
     organization  
     ⬝ Entry-hold-count solution
  + 새 디렉토리 항목  
    ⬝ Link : 다른 파일이나 서브 디렉토리  
     를 가리키는 포인터  
    ⬝ Resolve the link ( 링크 해석)  
     : 디렉토리 항목이 링크로 표시되어   
     있다면, 시제 파일 이름은 링크  
     정보에 포함됨. 실제 파일에 대한   
     경로를 사용하는 것이 해석
* 일반 그래프 디렉토리  
  (General Graph Directory)  
    
    
    
    
    
    
    
    
  + 순환이 발생하지 않도록 어떻게 보장?
  + 링크를 파일에만 허용한다  
    (서브 디렉토리 x)
  + 가비지 컬렉션 (garbage collection)  
    ⬝ 마지막 참조가 제거되고, 디스크  
     공간이 재할당될 수 있을 때를 결정  
     하기 위해  
    ⬝ 전체 파일 시스템을 검색하고,  
     접근 가능한 모든 것을 표시  
     -> 표시되지 않은 것들을 수집하고,   
     사용 가능한 공간 리스트에 추가  
    ⬝ 디스크 기반의 파일 시스템에서  
     가비지 컬렉션이 존재하면 시간을   
     매우 많이 소요  
    ⬝ 새 링크가 추가될 때마다 순환 탐지   
     알고리즘을 사용하여 여부를 결정

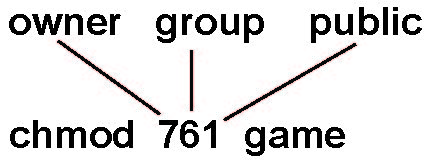
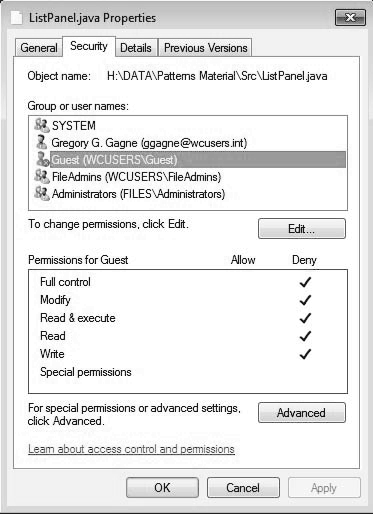
**10.4 파일 시스템 마운팅**

* File System Mounting  
  11_14.pdf
  + 파일 시스템은 프로세스들에 의해 사용되기 전에 mount 되어야 한다.
  + 운영체제에게 장치 이름과 파일 시스템을 부착할 수 있는 파일 구조 내의 위치(mount point)가 주어짐
  + Mount 되지 않는 파일 시스템은 mount point에 mount 되어야 함
  + Mount point  
    ⬝ 장착되는 파일 시스템이 부착될  
     비어 있는 디렉토리

**10.5 파일 공유**

* File Sharing
  + 다중 사용자 시스템에서 파일을 공유하는 것이 바람직함
  + 접근 제어, 파일 보호가 중요
  + 분산 시스템에서는 네트워크 전체에서 파일 공유 가능
  + NFS (Network File System)  
    : 공통 분산 파일 공유 방법
  + 소유자 : 파일 속성 변경, 파일 접근 허용, 파일과 디렉토리에 대해 가장 많은 제어 권한을 갖고 있는 user
  + User ID : 사용자를 식별하여, 사용자 당 권한 및 보호 허용
  + Group ID : 그룹 액세스 권한을 허용하는 그룹
* 원격 파일 시스템 (Remote File System)
  + 네트워크의 출현은 원격 컴퓨터 간의 통신을 가능하게 함  
    ⬝ FTP와 같은 프로그램을 통해  
     기계간에 파일을 직접 전송  
     🢭 FTP : 익명 or 인증형 접근  
    ⬝ 지역 기계에서 원격 디렉토리를  
     접근할 수 있는 DFS (Distributed   
     File Systems)  
    ⬝ WWW : 원격 파일에 접근하기 위해  
     브라우저가 필요하고, 파일 전송을   
     위해서는 별도의 연산이 사용  
     🢭 익명 접근
  + 익명 접근 : 원격 시스템에 계정이  
     없어도 사용자가 파일 전송 가능
* Client-Server Model
  + 원격 파일 시스템은 컴퓨터가 하나 이상의 원격 시스템으로부터 하나 이상의 파일 시스템 mount를 허용함
  + Server : 파일을 가지고 있는 컴퓨터
  + Client : 파일에 접근하기를 원하는  
     컴퓨터
  + 서버는 여러 클라이언트에게 서비스 제공 가능
  + 클라이언트는 여러 서버 사용 가능
  + 클라이언트는 네트워크 이름이나 IP 주소 같은 다른 식별자로 확인 가능  
    But, 해킹될 수 있음  
    -> Spoofing을 이용하여 허가된 것  
     처럼 위장하여 서버에 접속
  + NFS(Network File System)  
    : UNIX 표준 client-server 파일 공유   
     프로토콜
  + CIFS : Windows 표준 프로토콜
* 분산 정보 시스템  
  (Distributed Information System  
   = Distributed Naming Services)
  + 원격 컴퓨팅을 위해 필요한 정보에 단일화된 접근을 제공  
    ex) LDAP, DNS, NIS
* 오류 모드 (Failure Modes)
  + 모든 파일 시스템은 failure modes를 가짐  
    ex) 다른 디스크 관리 정보   
     (메타데이터)의 오류
  + 원격 파일 시스템은 네트워크 오류, 서버 오류 때문에 새로운 오류 모드가 추가됨
  + 오류 복구에는 원격 요청의 상태에 대한 static information이 포함
  + NFS v3와 같은 무 상태 프로토콜에는 각 요청의 모든 정보가 포함됨  
    -> 복구 쉬움 / 보안 낮음
* 일관성의 의미(Consistency Semantic)
  + 동시에 공유 파일에 접근하는 여러 사용자의 의미 명시
  + 여러 사용자가 공유 파일에 동시에 액세스하는 방법 지정
  + 6장 Synchronization 알고리즘과 연관  
    ⬝ 6장의 복잡한 알고리즘은  
     디스크와 네트워크 간의 큰 지연과  
     느린 전송률 때문에 파일 I/O의 경우  
     구현되지 않음
  + AFS (Andrew File System)은  
    복잡한 공유 의미 구현
* UFS (Unix File System)
  + 오픈 파일에 대한 한 사용자의 쓰기는 동일 파일을 연 다른 사용자에게 즉시 보일 수 있음
  + 현재 위치 포인터를 공유하는 공유 모드가 있어, 한 사용자에 대한 포인터의 전진은 공유하는 모든 사용자에게 영향을 미침
  + 파일 포인터를 공유하여 여러 사용자가 동시에 읽고 쓸 수 있도록 함
* 세션 의미(Session Semantic)
  + AFS은 다음의 일관성 의미 사용  
    ⬝ 오픈 파일에 대한 한 사용자의  
     쓰기는 동시에 동일 파일을 연 다른  
     사용자들에게 바로 보이지 않음  
    ⬝ 일단 파일이 닫히면, 파일에 대한   
     변경들이 그 후에 시작되는 Session  
     에서만 보임. (이미 열려진 파일의  
     인스턴스들은 변경 반영 X)

**10.6 보호**

* Protection
  + 파일 소유자/생성자는 다음 항목을 제어할 수 있어야 함  
    ⬝ What can be done (할 수 있는 일)  
    ⬝ by whom (누구에 의해서)
* 액세스 유형
  + 보호 기법은 가능한 파일 접근 유형을 제한함으로써 통제된 접근 제공
  + Read : 파일로부터 읽기
  + Write : 파일에 쓰기
  + Execute : 파일을 메모리에 읽어오고  
     실행하기
  + Append : 파일의 끝에  
     새로운 정보 첨부
  + Delete : 파일을 지우고  
     사용 공간을 반납하기
  + List : 파일의 속성, 이름 등을 출력
* 접근 제어
  + Mode of access  
    : Read / Write / Execute
  + Three classes of users on Unix/Linux  
    ⬝ owner access  
    ⬝ group access  
    ⬝ public access  
    ex) RWX RWX RWX  
     1 1 1 1 1 0 0 0 1  
     7 6 1  
    
  + 관리자에게 그룹 생성을 요청,  
    사용자를 그룹에 추가
  + 특정 파일이나 서브 디렉토리의 접근 제어 정의
  + Windows 7에서의 접근 제어  
      
    
  + UNIX Driectory Listing   
    