[Chapter 14]

소프트웨어융합학과/2020105742/한지훈

1. File System Implementation
   A. In-memory structure: 디스크 data를 커널이 읽어서 프로세스에 전달 시 커널에 어떻게 올려야 효과적일지
   B. On-disk structure: 실제 디스크 file system 구조를 어떻게 해야 효과적일지

2. In-memory structure
   A. per-process open-file table
      I. 개요: 프로세스는 커널에 파일 open 등 요청 시 파일의 최소 정보를 알아야 요청 가능
         - 각 프로세스는 PCB에 각각 테이블을 통해 각 프로세스에서 열려있는 파일 목록 확인
   B. system-wide open-file table
      I. 개요: 커널에서는 모든 프로세스가 열어놓은 파일 목록 정보를 중복 없이 통으로 관리
   C. in-memory partition table: file system을 구성하는 파티션들의 meta data를 저장
   D. Cache
      I. directory cache: 자주 접근하는 경로들을 캐싱
      II. buffer cache: 자주 접근하는 파일들을 캐싱
      III. 목적: 디스크는 느리기 때문에 병목을 줄이기 위해 캐싱을 활용함
   E. Virtual File System
      I. 개요: file system 별로 규격이 다 다른데, OS가 이를 다 알기 쉽지 않음
         - 공통 부분들은 공유하고(VFS), 특별한 부분만 따로 구현하여 커널이 알아야 하는 정보 줄이기
   F. Layered File System
      I. 개요: File system의 동작은 매체 종류/ 회사 특성 등에 따라 다름
         - 여러 레이어들을 각각 개발하고, 서로 조립하는 방식으로 File system 구현

3. On-Disk Structure
   A. 가정: Partition 1에 OS가 설치되어있고, file system이 구축 완료된 상태
   B. Partition 1 구성
      I. boot block: 0번지로, booting 관련 정보가 저장되어있음
      II. super block: File system의 metadata 저장(type, 블록 수 등)
      III. bitmaps: 디스크 용량 4GB로 가정하면 블록 크기는 4096kb이기 때문에 약 1M개의 블록 관리 필요
         - 모든 블록의 사용 여부를 저장하며, bitmap 크기로 disk 총량 확인 가능
            - 400GB 디스크 > 블록 100M개 > bitmap 크기 = 100M bit = 25Mbyte
            - 1: 사용 X / 0: 사용 중
      IV. i-nodes: File Control Block(FCB)에 각 파일에 대한 정보를 저장
      V. root dir: 시작 주소
      VI. files & directories: 실제 data들이 블록단위로 저장되는 공간

4. Directory Implementation
   A. Directory FCB에 모든 file 정보 저장
      I. 문제: 무거워짐
         - D FCB에는 file 이름 정보만, 이름 통해 포인터로 매핑
         - 또는 하이브리드(직접 접근 정보는 D FCB에, 나머지는 개별 FCB에)

5. Allocation Methods

A. Contiguous Allocation: D에는 시작/길이 정보, 쭉 쓰기

I. 장점: spacial locality

II. 단점: external fragmentation

III. 적용: CD, DVD

B. Linked allocation: D에는 시작/끝 정보, 각 블록은 다음 블록 링크 존재

I. 장점: external fragmentation X

II. 단점:

- spacial locality X
- 신뢰성 낮음(링크 1개 깨지면 다 날라감)

➢ 개선: FAT(File allocation table) – but 포인터만 저장하는 블록 있어도 신뢰성 문제 있음

C. Indexed allocation: 실제 data 블록 인덱스 모두 저장

I. 장점: 신뢰성 높음(인덱스 깨지면 그 파일만 날라감)

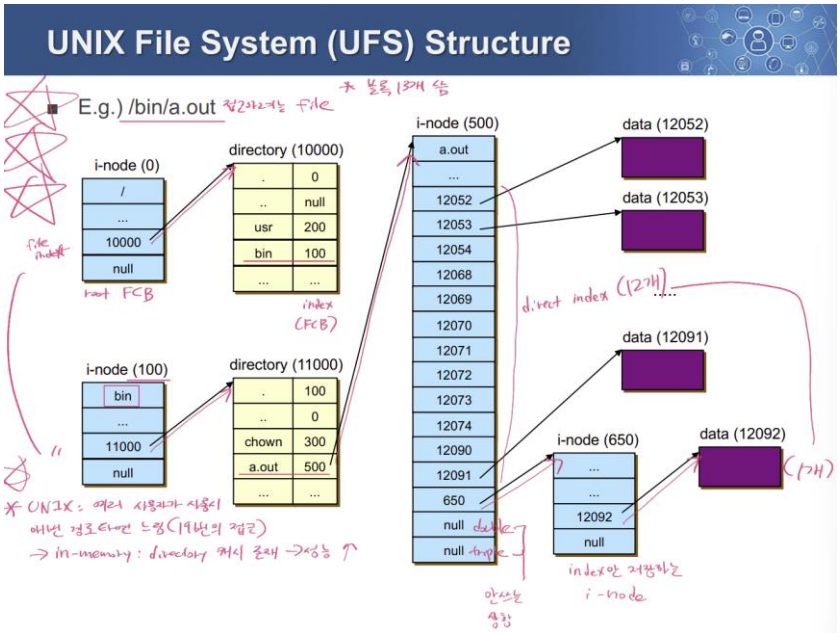
II. 단점:

- spacial locality X
- 인덱스까지 저장해야 해서 용량 커짐

III. 특징: UNIX 사용(i-node)

IV. 용량 보완: UFS

- FCB 크기: 4kb
- Direct block: 최대 12개 블록
- Single indirect: single 포인터로 인덱스 1000개(4mb file) 저장
- Double indirect: double 포인터로 인덱스 1M개(4GB file) 저장
- Triple indirect: triple 포인터로 인덱스 1G개(1TB file) 저장



6. 블록 크기

A. 4kb

I. Data rate: 블록 크기 클수록 높아짐(data 한번에 많이)

II. Utilization: 블록 크기 클수록 낮아짐(internal fragmentation)

III. 절충, utilization 최소 이상 유지: 4kb

7. Buffer Cache

A. 개요: unix 여러 사용자가 사용시 매번 경로 타면 느림

- In-memory: directory 캐시를 둬서 성능 높임

B. 문제: write-back시 버퍼, 디스크 사이 캐시는 느려서 일관성 유지 어려움(unreliability)

- OS 딴에서 체크(fsck): 근본 해결 X
- Journaling file systems = Log Structured File Systems:
  - 개요: unreliability 사전에 제거 – JFS
  - 방식: log 정보 저장, inconsistancy 유지시 log 유지, consistant하게 되면 log 지우기
    ➔ 다운 시 log 확인하여 log 있으면 log 통해 복구