8단원. 파일 시스템과 I/O

> 대상: 전공자 · 정보처리기사 취득자 | 목적: 인프라 직무 기술 면접 대비 중요도: ★★★ 최빈출 · ★★ 자주 출제 · ★ 기본 개념

Q1. 파일 시스템이란 무엇이고 어떤 일을 하나요? ★★

답안 디스크 같은 블록 장치 위에 파일과 디렉터리라는 추상화를 제공하는 OS 구성 요소입니다. 핵심 기능은 ① 파일 이름 → 디스크 블록 매핑(할당 관리), ② 디렉터리 계층 구조 관리, ③ 빈 공간 관리, ④ 권한·소유권 등 메타데이터 관리, ⑤ 크래시 일관성 보장(저널링)입니다. 리눅스는 ext4, XFS, Btrfs, 윈도우는 NTFS가 대표적이며, VFS(가상 파일 시스템) 계층이 서로 다른 파일시스템에 동일한 인터페이스(open/read/write)를 제공합니다.

관련 개념 블록 장치, 메타데이터, VFS, ext4/XFS/NTFS, 마운트

Q2. 아이노드(inode)란 무엇인가요? ★★★

답안 유닉스 계열 파일시스템에서 **파일 하나의 메타데이터를 담는 자료구조**입니다. 파일 크기, 소유자, 권한, 타임스탬프, 링크 수, 그리고 **데이터 블록들의 위치(직접/간접 블록 포인터)**를 담습니다. 중요한 점은 **파일 이름은 아이노드에 없다**는 것입니다. 이름은 디렉터리(이름→아이노드 번호 매핑 테이블)가 관리합니다. 운영 관점에서 아이노드 개수는 파일시스템 생성 시 고정되므로, 작은 파일이 수백만 개 쌓이면 디스크 용량이 남아도 아이노드 고갈로 파일 생성이 실패할 수 있습니다(df -i로 확인).

관련 개념 직접/간접 블록 포인터, 디렉터리 엔트리, df -i, 아이노드 고갈 장애

Q3. 하드 링크와 심볼릭 링크의 차이를 설명하세요. ★★★

답안 하드 링크는 같은 아이노드를 가리키는 또 다른 이름입니다. 원본과 완전히 동등하며, 아이노드의 링크 수가 0이 될 때(모든 이름 삭제) 실제 데이터가 삭제됩니다. 같은 파일시스템 안에서만 가능하고 디렉터리에는 만들 수 없습니다. **심볼릭 링크**는 대상의 **경로 문자열**을 담은 별도 파일(자체 아이노드 보유)로, 다른 파일시스템·디렉터리도 가리킬 수 있지만 원본이 삭제되면 깨진 링크(dangling)가 됩니다. 참고로 '파일 삭제(rm)'는 실제로는 unlink, 즉 이름-아이노드 연결 제거이며, 삭제된 파일을 프로세스가 열고 있으면 링크 수가 0이어도 닫을 때까지 공간이 반환되지 않습니다. 이는 "지웠는데 df 용량이 안 줄어요" 장애의 원인입니다.

관련 개념 링크 카운트, unlink, 열린 파일과 공간 미반환(lsof +L1), ln vs ln -s

Q4. 저널링 파일 시스템은 왜 필요한가요? ★★

답안 파일 작업 하나(예: 파일 추가)는 아이노드, 디렉터리, 빈 공간 비트맵 등 여러 블록 수정으로 이뤄지는데, 도중에 전원이 나가면 일부만 반영되어 파일시스템이 **불일치** 상태가 됩니다. 과거에는 부팅 시 fsck로 전체 디스크를 검사해 복구했지만 대용량 디스크에서는 수 시간이 걸립니다. **저널링**은 변경 내용을 먼저 저널(로그) 영역에 기록하고 커밋한 뒤 실

제 위치에 반영합니다. 크래시가 나면 저널만 재생(redo)하거나 버려서 일관성을 빠르게 복구합니다. DB의 WAL(Write-Ahead Logging)과 같은 원리입니다. ext4는 기본으로 메타데이터만 저널링(ordered 모드)합니다.

관련 개념 크래시 일관성, fsck, 저널 재생, WAL, ext4 저널 모드(journal/ordered/writeback)

Q5. 파일 할당 방식(연속, 연결, 인덱스)을 비교하세요. ★★

답안 연속 할당은 파일을 연속 블록에 저장합니다. 순차·직접 접근 모두 빠르지만 외부 단편화와 파일 확장 곤란이 문제입니다(CD-ROM 등에 적합). **연결 할당**은 각 블록이 다음 블록 포인터를 가집니다. 단편화가 없고 확장이 자유롭지만 직접 접근이 O(n)이고 포인터 하나만 깨져도 이후를 잃습니다. FAT는 포인터를 별도 테이블로 모아 이를 개선한 변형입니다. **인덱스 할당**은 블록 위치들을 인덱스 블록에 모아 저장해 직접 접근이 빠르며, 유닉스 아이노드의 직접/간접 포인터가 이 방식입니다(대용량은 다중 간접 블록으로 확장). 현대 파일시스템(ext4, XFS)은 연속된 블록 구간을 (시작, 길이)로 표현하는 **익스텐트**로 인덱스 방식의 메타데이터 부담을 줄입니다.

관련 개념 외부 단편화, FAT, 아이노드 간접 블록, 익스텐트

Q6. 버퍼드 I/O와 다이렉트 I/O, 동기와 비동기 I/O를 구분해 설명하세요. ★★

답안 버퍼드 I/O(기본값)는 페이지 캐시를 경유해 읽기 캐싱과 쓰기 지연(writeback)의 이득을 얻습니다. **다이렉트 I/O**(O_DIRECT)는 페이지 캐시를 우회해 사용자 버퍼와 디스크가 직접 데이터를 주고받으며, 자체 캐시를 가진 DB(오라클, MySQL InnoDB)가 이중 캐싱을 피하려 사용합니다. 별개 축으로 **동기 I/O**는 완료까지 호출 스레드가 대기하고(블로킹), **비동기 I/O**는 요청만 걸어두고 완료를 나중에 통지받습니다(리눅스 AIO, 최신 io_uring). 고성능 서버는 논블로킹 소켓+epoll(이벤트 기반) 또는 io_uring으로 적은 스레드로 많은 I/O를 처리합니다.

관련 개념 페이지 캐시, O_DIRECT, fsync(내구성 보장), epoll, io_uring

Q7. RAID 레벨(0, 1, 5, 6, 10)을 비교 설명하세요. ★★★

답안 RAID 0(스트라이핑): 데이터를 여러 디스크에 분산해 성능·용량 최대, 중복성 없음(디스크 1개 고장=전체 손실). **RAID 1(미러링)**: 동일 데이터를 복제해 안정성 높지만 용량 효율 50%. **RAID 5**: 블록 스트라이핑+분산 패리티로 디스크 1개 고장 허용, 용량 효율 (n−1)/n, 쓰기 시 패리티 계산 오버헤드와 재빌드 중 추가 고장 위험이 단점. **RAID 6**: 패리티 2중으로 2개 고장 허용, 대용량 디스크 시대에 재빌드 위험 때문에 5보다 선호됨. **RAID 10(1+0)**: 미러 쌍을 스트라이핑. 성능과 안정성이 좋아 DB 서버 정석이지만 용량 효율 50%. 면접에서는 "DB 서버에 어떤 RAID?"(→10 또는 6) 같은 선택 질문이 흔합니다. RAID는 백업이 아니라는 점(삭제·랜섬웨어 못 막음)도 언급하면 좋습니다.

관련 개념 스트라이핑/미러링/패리티, 재빌드 위험, 핫 스페어, RAID는 백업이 아님

Q8. 디스크 스케줄링 알고리즘(FCFS, SSTF, SCAN, C-SCAN)을 설명하세요. ★★

답안 HDD는 헤드 이동(seek)이 지배적 비용이므로 요청 순서를 재배열해 이동을 줄입니다. **FCFS**는 도착 순서대로 처리(공정하지만 비효율). **SSTF**는 현재 헤드에서 가장 가까운 요청부터 처리(효율적이나 먼 요청 기아 가능). **SCAN(엘리베

이터)은 헤드가 한쪽 끝까지 이동하며 경로상 요청을 처리하고 방향을 반전합니다. **C-SCAN**은 한 방향으로만 처리하고 끝에 도달하면 처음으로 복귀해 대기 시간이 더 균등합니다. 계산 문제(총 이동 거리)로 자주 출제됩니다. 참고로 SSD는 seek이 없어 이런 스케줄링이 무의미하며, 리눅스는 SSD에 noop/none 스케줄러를 씁니다.

관련 개념 seek time, 엘리베이터 알고리즘, 이동 거리 계산, SSD와 스케줄러(none/mq-deadline)

Q9. SSD와 HDD의 차이가 OS 설계에 주는 영향은? ★★

답안 SSD는 기계적 이동이 없어 임의 접근이 HDD보다 수백 배 빠르고 순차/임의 성능 격차가 작습니다. 다만 **덮어쓰기가 불가능**해 블록을 지우고 다시 써야 하며(erase-before-write), 쓰기 수명이 유한합니다. 이에 따라 ① 디스크 스케줄링 단순화(none), ② **TRIM** 명령으로 삭제된 블록을 SSD에 알려 가비지 컬렉션 효율화, ③ 컨트롤러의 웨어 레벨링으로 쓰기 분산, ④ 임의 I/O 회피를 전제로 한 알고리즘들(B-tree 최적화 등)의 재검토, ⑤ NVMe·io_uring 같은 고속 인터페이스 대응이 이뤄졌습니다. 인프라 관점에서는 IOPS/지연시간 프로파일이 달라 DB 스토리지 설계 기준이 달라집니다.

관련 개념 erase-before-write, TRIM/discard, 웨어 레벨링, NVMe, IOPS

Q10. 리눅스에서 "모든 것이 파일이다"라는 말의 의미는? ★★

답안 리눅스는 일반 파일뿐 아니라 디렉터리, 장치(/dev/sda), 파이프, 소켓, 커널 정보(/proc, /sys)까지 **파일 디스크립터와 read/write 인터페이스로 통일**해서 다룬다는 설계 철학입니다. 덕분에 같은 도구(cat, redirect, epoll)로 다양한 자원을 조작할 수 있습니다. 예로 /proc/cpuinfo로 CPU 정보를 읽고, /sys로 커널 파라미터를 조정하며, 소켓도 fd로 select/epoll에 넣습니다. 프로세스당 열 수 있는 fd 수에는 한계(ulimit -n)가 있어, 대량 커넥션 서버에서 "Too many open files" 장애의 원인이 됩니다. 이 역시 인프라 면접 단골입니다.

관련 개념 파일 디스크립터, /proc, /sys, ulimit -n, Too many open files

Q11. fsync는 왜 필요하며, write만 하면 데이터가 안전하지 않은 이유는? ★★

답안 write()는 데이터를 커널의 페이지 캐시에 복사하면 성공을 반환할 뿐, 그 시점에 디스크에는 아직 없습니다(dirty page). 커널이 나중에 writeback하기 전 전원이 나가면 데이터가 유실됩니다. **fsync()**는 해당 파일의 더티 페이지와 메타데이터를 실제 저장 장치까지 내려쓰고 완료를 기다려 **내구성(durability)**을 보장합니다. DB가 트랜잭션 커밋 시 WAL에 fsync하는 이유이며, fsync 빈도는 내구성과 성능의 직접적 트레이드오프입니다(PostgreSQL synchronous_commit, MySQL innodb_flush_log_at_trx_commit). 디스크 자체 캐시까지 고려하면 배리어/FUA 플러시도 관련됩니다.

관련 개념 페이지 캐시와 dirty page, 내구성, WAL+fsync, flush 설정 튜닝
