6단원. 메모리 관리

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

Q1. 논리 주소와 물리 주소의 차이, 그리고 MMU의 역할을 설명하세요. ★★★

답안 논리(가상) 주소는 CPU(프로세스)가 바라보는 주소이고, **물리 주소**는 실제 메모리 하드웨어의 주소입니다. 프로세스는 자신만의 논리 주소 공간(0부터 시작)을 가지며, 실행 시점에 **MMU(Memory Management Unit)**가 논리 주소를 물리 주소로 변환합니다. 이 분리 덕분에 프로세스를 물리 메모리 어디에나 배치할 수 있고, 프로세스 간 메모리 보호와 가상 메모리 구현이 가능해집니다. 페이징에서는 페이지 테이블이 변환 정보를 담고 MMU가 이를 참조합니다.

관련 개념 MMU, 주소 바인딩(컴파일/적재/실행 시간), 재배치 레지스터, 페이지 테이블

Q2. 연속 메모리 할당에서 내부 단편화와 외부 단편화의 차이는? ★★★

답안 내부 단편화는 할당된 블록이 요청보다 커서 블록 **내부**에 남는 낭비 공간입니다(고정 분할, 페이징에서 마지막 페이지). **외부 단편화**는 할당/해제가 반복되며 생기는 흩어진 작은 빈 공간들로, 총합은 충분한데 연속 공간이 없어 할당에 실패하는 문제입니다(가변 분할). 외부 단편화는 **압축(compaction)**으로 해소할 수 있으나 비용이 크고, 근본적으로는 **페이징**이 연속 할당 요구를 없애 외부 단편화를 제거합니다. 대신 페이징은 페이지 단위 반올림으로 인한 내부 단편화를 감수합니다.

관련 개념 고정/가변 분할, 압축, 페이징의 트레이드오프, 50% 규칙

Q3. 가변 분할에서 최초 적합, 최적 적합, 최악 적합을 비교하세요. ★★

답안 최초 적합(first fit)은 처음 발견한 충분한 공간에 할당하며 탐색이 빨라 일반적으로 성능과 공간 효율의 균형이 좋습니다. **최적 적합(best fit)**은 가장 작은 충분한 공간을 골라 낭비를 줄이려 하지만, 전체 탐색이 필요하고 아주 작은 자투리 공간을 대량 생산해 외부 단편화가 오히려 심해질 수 있습니다. **최악 적합(worst fit)**은 가장 큰 공간에 할당해 남는 조각을 크게 유지하려는 발상이지만 실험적으로 성능이 가장 나쁩니다. 시뮬레이션 결과 최초 적합과 최적 적합이 최악 적합보다 우수하며, 속도는 최초 적합이 가장 빠릅니다.

관련 개념 free list 탐색, 자투리 조각, 정보처리기사 계산 문제 유형

Q4. 페이징(Paging)의 원리와 장단점을 설명하세요. ★★★

답안 물리 메모리를 고정 크기 **프레임**으로, 논리 메모리를 같은 크기 **페이지**로 나누고, 페이지 테이블로 페이지→프레임 매핑을 관리합니다. 논리 주소는 (페이지 번호 p, 오프셋 d)로 나뉘고, p로 페이지 테이블을 조회해 프레임 번호를 얻은 뒤 오프셋을 붙여 물리 주소를 만듭니다. 장점은 외부 단편화 제거, 연속 배치 불필요, 가상 메모리·공유·보호의 기반 제공

입니다. 단점은 페이지 테이블 유지 메모리 오버헤드, 주소 변환으로 인한 메모리 접근 2회(TLB로 완화), 마지막 페이지의 내부 단편화입니다. 일반적 페이지 크기는 4KB입니다.

관련 개념 페이지/프레임, 페이지 테이블, (p, d) 주소 분해, TLB, 4KB 페이지

Q5. TLB(Translation Lookaside Buffer)란 무엇이고 왜 필요한가요? ★★★

답안 페이지 테이블은 메모리에 있으므로 주소 변환마다 메모리 접근이 한 번 더 필요해, 모든 메모리 접근이 2배로 느려집니다. TLB는 최근 사용한 페이지→프레임 변환을 담는 MMU 내부의 고속 연관 캐시로, TLB 히트 시 메모리 접근 없이 즉시 변환합니다. 지역성 덕분에 히트율이 99% 수준이라 실효 접근 시간이 거의 1회 접근에 수렴합니다. 컨텍스트 스위칭 시 주소 공간이 바뀌므로 TLB를 플러시해야 하는데(ASID로 완화), 이것이 프로세스 전환이 비싼 주요 이유 중 하나입니다. 대용량 메모리 서버에서는 TLB 미스를 줄이기 위해 huge page(2MB/1GB)를 씁니다.

관련 개념 실효 접근 시간(EAT) 계산, ASID, TLB 플러시, huge page(DB 서버 튜닝)

Q6. 다단계 페이지 테이블은 왜 사용하나요? ★★

답안 64비트(실질 48비트) 주소 공간을 단일 페이지 테이블로 관리하면 프로세스마다 수백 GB급 테이블이 필요해 불가능합니다. 다단계 페이지 테이블은 페이지 테이블 자체를 페이지 단위로 쪼개 트리 구조로 만들고, **실제로 사용하는 주소 영역의 테이블만 생성**합니다. 대부분의 프로세스는 주소 공간을 드문드문 쓰므로 메모리가 크게 절약됩니다. 대가는 변환 시 단계 수만큼 메모리 접근이 늘어나는 것인데(x86-64는 4~5단계), TLB 히트가 대부분이므로 실용적입니다. 대안으로 해시 페이지 테이블, 역페이지 테이블이 있습니다.

관련 개념 x86-64 4단계 워킹, 희소 주소 공간, 역페이지 테이블, 페이지 워크 비용

Q7. 세그멘테이션과 페이징의 차이를 설명하세요. ★★

답안 세그멘테이션은 프로그램을 코드, 데이터, 스택 등 의미 있는 논리 단위(세그먼트)로 나누며, 세그먼트마다 크기가 다릅니다. 논리적 구조가 보존되어 세그먼트 단위 보호·공유가 자연스럽지만, 가변 크기라 외부 단편화가 발생합니다. **페이징**은 의미와 무관한 고정 크기로 잘라 외부 단편화가 없지만 논리적 단위와 무관합니다. 과거에는 둘을 결합(세그먼트를 페이징)하기도 했으나, 현대 OS(리눅스, x86-64)는 사실상 페이징만 사용하고 세그멘테이션은 형식적으로만 남아 있습니다. 다만 '논리적 영역' 개념은 가상 메모리 영역(VMA) 관리로 이어집니다.

관련 개념 세그먼트 테이블(base/limit), 외부 단편화, 페이지드 세그멘테이션, VMA

Q8. 스와핑(Swapping)이란 무엇이고, 현대 OS에서는 어떻게 달라졌나요? ★★

답안 전통적 스와핑은 메모리가 부족할 때 프로세스 **전체**를 디스크(스왑 영역)로 내보내고 필요할 때 다시 들여오는 방식입니다. 프로세스 단위 이동은 비용이 너무 커서, 현대 OS는 **페이지 단위 스와핑(페이지 아웃/인)**을 사용합니다. 즉 잘 안 쓰는 페이지만 골라 내보냅니다. 리눅스 운영 관점에서 스왑 사용량 자체보다 **스왑 인/아웃이 지속 발생(si/so)**하는지가

성능 문제의 신호이며, swappiness 파라미터로 페이지 캐시 회수와 익명 페이지 스왑의 비율을 조절합니다. 스왑이 없으면 메모리 부족 시 OOM Killer가 바로 동작합니다.

관련 개념 스왑 파티션/파일, vmstat si/so, swappiness, OOM Killer, zram

Q9. 메모리 보호는 어떻게 구현되나요? ★★

답안 페이징 환경에서는 페이지 테이블 엔트리에 **보호 비트**(읽기/쓰기/실행 권한, 유효 비트, 사용자/커널 구분)를 두고, MMU가 모든 접근을 검사해 위반 시 트랩(폴트)을 발생시킵니다. 프로세스는 자기 페이지 테이블에 매핑된 페이지만 접근할 수 있으므로 다른 프로세스의 메모리는 원천적으로 보이지 않습니다. 보안 기법으로 확장하면, **NX 비트**는 데이터 페이지에서 코드 실행을 막아 버퍼 오버플로우 공격을 어렵게 하고, **ASLR**은 스택·힙·라이브러리 배치를 무작위화합니다. 잘못된 접근은 SIGSEGV(세그멘테이션 폴트)로 나타납니다.

관련 개념 보호 비트, 유효/무효 비트, NX(DEP), ASLR, SIGSEGV

Q10. Copy-on-Write(COW)의 동작 원리를 설명하세요. ★★★

답안 fork() 시 부모의 주소 공간 전체를 즉시 복사하는 대신, 부모와 자식이 **같은 물리 페이지를 공유**하고 해당 페이지들을 읽기 전용으로 표시합니다. 어느 쪽이든 쓰기를 시도하면 보호 폴트가 발생하고, 그때서야 커널이 그 페이지만 복사해 각자에게 사설 사본을 줍니다. 대부분의 fork가 곧바로 exec로 이어져 복사가 무의미해지는 현실에서, COW는 fork를 사실상 페이지 테이블 복사 비용만으로 줄여줍니다. 같은 원리가 파일시스템 스냅샷(Btrfs, ZFS), 가상 머신 디스크 이미지 등 인프라 전반에 쓰입니다.

관련 개념 fork 최적화, 보호 폴트, 스냅샷(ZFS/Btrfs), Redis BGSAVE의 COW 활용

Q11. 캐시 메모리와 메모리 계층 구조를 설명하세요. ★★

답안 레지스터 → L1/L2/L3 캐시 → 주기억장치(DRAM) → SSD/디스크 순으로, 위로 갈수록 빠르고 작으며 비쌉니다. 계층 구조가 동작하는 근거는 **지역성**입니다. 시간 지역성(방금 쓴 데이터를 곧 다시 씀)과 공간 지역성(인접 데이터를 곧 씀) 덕분에 작은 캐시로도 대부분의 접근을 상위 계층에서 처리할 수 있습니다. 대략적 접근 시간은 L1 ~1ns, DRAM ~100ns, NVMe SSD ~수십 µs, HDD ~ms로 계층 간 수십~수천 배 차이가 나므로, 상위 계층 히트율이 시스템 성능을 좌우합니다. 캐시 일관성(MESI)은 멀티코어에서 코어 간 캐시 동기화 프로토콜입니다.

관련 개념 시간/공간 지역성, 캐시 라인(64B), 히트율과 평균 접근 시간, MESI
