2단원. 프로세스와 스레드 (최빈출 단원)

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

Q1. 프로세스와 프로그램의 차이는 무엇인가요? ★★★

답안 프로그램은 디스크에 저장된 정적인 실행 파일(코드와 데이터)이고, 프로세스는 그 프로그램이 메모리에 적재되어 실행 중인 동적인 인스턴스입니다. 프로세스는 코드뿐 아니라 PC(프로그램 카운터), 레지스터, 스택, 힙, 열린 파일 목록 등 실행 상태를 가지며, 하나의 프로그램에서 여러 프로세스가 동시에 생성될 수 있습니다.

관련 개념 실행 파일(ELF), 프로세스 이미지, 주소 공간, PCB

Q2. 프로세스의 메모리 구조(주소 공간)를 설명하세요. ★★★

답안 낮은 주소부터 **코드(텍스트) 영역**(기계어 명령, 읽기 전용), **데이터 영역**(전역·정적 변수, 초기화된 data와 초기화되지 않은 BSS), **힙**(malloc 등 동적 할당, 위로 성장), **스택**(지역 변수·함수 호출 프레임·복귀 주소, 아래로 성장)으로 구성됩니다. 스택과 힙이 서로를 향해 자라다 충돌하면 스택 오버플로우/메모리 부족이 발생합니다. 각 프로세스는 독립된 가상 주소 공간을 가져 서로의 메모리를 직접 침범할 수 없습니다.

관련 개념 Text/Data/BSS/Heap/Stack, 가상 주소 공간, 스택 오버플로우, ASLR(보안 연계)

Q3. 프로세스와 스레드의 차이를 설명하세요. ★★★

답안 프로세스는 자원 소유의 단위, 스레드는 CPU 스케줄링(실행)의 단위입니다. 프로세스는 독립된 주소 공간과 자원을 가지지만, 한 프로세스 안의 스레드들은 코드·데이터·힙·열린 파일을 **공유**하고 스택과 레지스터(PC 포함)만 각자 가집니다. 그래서 스레드는 생성·전환 비용이 싸고 통신이 쉽지만, 공유 메모리로 인한 동기화 문제가 생기고 한 스레드의 오류가 프로세스 전체를 죽일 수 있습니다. 프로세스는 격리성이 좋은 대신 생성·전환·IPC 비용이 큽니다.

관련 개념 자원 공유 범위, 스택/레지스터 독립, 동기화 필요성, 격리성 vs 효율성

Q4. PCB(Process Control Block)에는 어떤 정보가 들어있나요? ★★★

답안 PCB는 커널이 프로세스를 관리하기 위해 유지하는 자료구조로, 프로세스 상태(실행·준비·대기 등), PID, 프로그램 카운터와 레지스터 값, CPU 스케줄링 정보(우선순위 등), 메모리 관리 정보(페이지 테이블 포인터), 열린 파일 목록, 계정 및 I/O 상태 정보가 저장됩니다. 컨텍스트 스위칭 시 현재 프로세스의 실행 문맥을 PCB에 저장하고, 다음 프로세스의 PCB에서 문맥을 복원합니다. 리눅스에서는 task_struct가 이에 해당합니다.

관련 개념 task_struct, 컨텍스트 스위칭, 프로세스 테이블

Q5. 프로세스 상태 전이도를 설명하세요. ★★★

답안 **생성(new) → 준비(ready) → 실행(running) → 종료(terminated)**가 기본 흐름이고, 실행 중 I/O 등 이벤트를 기다리면 **대기(waiting/blocked)**로 갔다가 완료되면 준비로 돌아갑니다. 핵심 전이는 ① 준비→실행: 스케줄러의 디스패치, ② 실행→준비: 타임 슬라이스 만료나 선점, ③ 실행→대기: I/O 요청 등 블로킹, ④ 대기→준비: 이벤트 완료입니다. 대기 상태에서 곧바로 실행으로 갈 수 없다는 점(반드시 준비를 거침)이 자주 나오는 포인트입니다. 메모리 부족 시 프로세스를 디스크로 내리는 중단(suspended) 상태를 추가하기도 합니다.

관련 개념 디스패치, 선점, 블로킹, suspended ready/blocked, 중기 스케줄러

Q6. 컨텍스트 스위칭이란 무엇이고, 왜 비용이 발생하나요? ★★★

답안 CPU를 한 프로세스(스레드)에서 다른 프로세스로 넘길 때 현재 실행 문맥(레지스터, PC 등)을 PCB에 저장하고 다음 프로세스의 문맥을 복원하는 작업입니다. 비용은 ① 레지스터 저장/복원과 커널 모드 전환 같은 직접 비용, ② 캐시·TLB가 무효화되어 이후 메모리 접근이 느려지는 간접 비용으로 나뉘며, 실제로는 간접 비용이 더 큽니다. 같은 프로세스 내 스레드 간 전환은 주소 공간이 같아 TLB 플러시가 필요 없어 프로세스 간 전환보다 쌉니다.

관련 개념 PCB 저장/복원, TLB 플러시, 캐시 오염, 스레드 전환 vs 프로세스 전환

Q7. fork()와 exec()의 차이와, 자주 함께 쓰이는 이유를 설명하세요. ★★★

답안 fork()는 호출한 프로세스를 복제해 자식 프로세스를 만듭니다. 자식은 부모의 주소 공간 복사본을 가지며, 반환값(부모=자식 PID, 자식=0)으로 구분합니다. exec()는 현재 프로세스의 주소 공간을 새 프로그램으로 교체합니다(PID 유지). 쉘이 명령을 실행할 때 fork()로 자식을 만들고 자식에서 exec()로 명령 프로그램을 실행하는 fork-exec 패턴이 표준입니다. fork()의 복사 비용은 **Copy-on-Write**로 최적화되어, 실제로는 페이지 테이블만 복사하고 쓰기가 발생할 때만 해당 페이지를 복사합니다.

관련 개념 fork-exec 패턴, Copy-on-Write(COW), wait(), PID

Q8. 좀비 프로세스와 고아 프로세스의 차이는 무엇이며, 좀비는 어떻게 처리하나요? ★★★

답안 **좀비 프로세스**는 자식이 종료되었지만 부모가 wait()로 종료 상태를 회수하지 않아 프로세스 테이블 항목만 남은 상태입니다. 메모리는 거의 안 쓰지만 PID를 점유하므로 대량 발생 시 PID 고갈이 될 수 있습니다. 부모가 wait()/waitpid()를 호출하거나, SIGCHLD 핸들러에서 회수하도록 하고, 이미 쌓였다면 부모를 종료시켜 init(systemd)이 입양 후 회수하게 합니다. **고아 프로세스**는 부모가 먼저 죽은 자식으로, init(PID 1)이 입양하여 정상 관리되므로 좀비만큼 문제가 되지 않습니다.

관련 개념 wait()/waitpid(), SIGCHLD, init 입양, defunct(ps에서 Z 상태)

Q9. IPC(프로세스 간 통신) 방식들을 비교 설명하세요. ★★★

답안 크게 **공유 메모리**와 **메시지 전달**로 나뉩니다. 공유 메모리는 두 프로세스가 같은 메모리 영역을 매핑해 통신하며, 커널 개입 없이 읽고 써서 가장 빠르지만 동기화를 직접 해야 합니다. 메시지 전달은 커널을 경유해 안전하지만 시스템 콜 오버헤드가 있습니다. 구체적으로는 파이프(부모-자식 단방향), 네임드 파이프(FIFO, 무관한 프로세스 간), 메시지 큐, 시그널(간단한 이벤트 통지), 소켓(네트워크 포함, 다른 호스트 간 가능), 공유 메모리+세마포어 조합 등이 있습니다. 같은 호스트 대용량 데이터는 공유 메모리, 원격 통신은 소켓이 정석입니다.

관련 개념 파이프/FIFO, 메시지 큐, 공유 메모리(shm), 시그널, 유닉스 도메인 소켓, TCP 소켓

Q10. 사용자 수준 스레드와 커널 수준 스레드의 차이를 설명하세요. ★★

답안 사용자 수준 스레드는 커널이 모르는 채 사용자 공간 라이브러리가 스케줄링합니다. 생성·전환이 매우 싸지만, 한 스레드가 블로킹 시스템 콜을 하면 프로세스 전체가 블록되고 멀티코어 병렬 실행이 안 됩니다. **커널 수준 스레드**는 커널이 직접 관리해 개별 블로킹과 멀티코어 활용이 가능하지만 생성·전환에 시스템 콜이 필요해 상대적으로 무겁습니다. 매핑 모델로 다대일, 일대일(리눅스 NPTL, 현대 표준), 다대다가 있으며, Go의 고루틴은 다대다(M:N)를 런타임에서 구현한 사례입니다.

관련 개념 다대일/일대일/다대다 모델, NPTL, 고루틴, 그린 스레드

Q11. 멀티스레드 대신 멀티프로세스를 쓰는 게 나은 경우는 언제인가요? ★★

답안 격리성과 안정성이 중요할 때입니다. 멀티프로세스는 한 프로세스가 죽어도 다른 프로세스에 영향이 없어, 크롬 브라우저(탭별 프로세스), nginx(워커 프로세스), 오래된 아파치 prefork가 이 방식을 씁니다. 보안상 권한 분리가 필요하거나(권한 낮춘 워커), 파이썬처럼 GIL 때문에 스레드로 CPU 병렬화가 안 되는 경우도 멀티프로세스가 답입니다. 반대로 데이터 공유가 잦고 생성·전환 비용이 민감하면 멀티스레드가 유리합니다.

관련 개념 장애 격리, 권한 분리, GIL, nginx 워커 모델, 크롬 프로세스 모델

Q12. 스레드 풀은 왜 사용하나요? ★★

답안 요청마다 스레드를 생성/파괴하면 비용이 크고, 요청 폭주 시 스레드가 무한정 늘어나 메모리 고갈과 과도한 컨텍스트 스위칭이 발생합니다. 스레드 풀은 미리 일정 수의 스레드를 만들어 두고 작업 큐에서 작업을 꺼내 처리하게 하여, ① 생성 비용 제거 ② 동시 스레드 수 상한으로 시스템 보호 ③ 응답 시간 안정화를 달성합니다. 풀 크기는 CPU 바운드 작업이면 코어 수 내외, I/O 바운드 작업이면 그보다 크게 잡는 것이 일반적입니다.

관련 개념 작업 큐, 스레드 생성 비용, CPU 바운드 vs I/O 바운드, 백프레셔

Q13. CPU 바운드 프로세스와 I/O 바운드 프로세스의 차이는? ★★

답안 CPU 바운드는 계산 위주로 CPU 버스트가 길고(인코딩, 과학 계산), I/O 바운드는 I/O 대기가 대부분이며 CPU 버스트가 짧습니다(웹 서버, DB 클라이언트). 스케줄러는 응답성을 위해 I/O 바운드에 우선권을 주는 경향이 있습니다. I/O 바운드가 CPU를 짧게 쓰고 빨리 I/O를 걸어야 장치 이용률도 올라가기 때문입니다. 서버 튜닝 관점에서 CPU 바운드는 코어 수 증설·병렬화, I/O 바운드는 비동기 I/O·캐싱이 처방입니다.

관련 개념 CPU 버스트/I/O 버스트, 다단계 피드백 큐의 우선순위 부여, 비동기 I/O

Q14. 데몬 프로세스란 무엇인가요? ★

답안 터미널과 분리되어 백그라운드에서 상주하며 서비스를 제공하는 프로세스입니다(sshd, nginx, crond 등). 전통적으로 fork 후 부모 종료, setsid()로 새 세션 생성(제어 터미널 분리), 작업 디렉터리 변경, 표준 입출력 리다이렉트 과정을 거쳐 만들어집니다. 현대 리눅스에서는 systemd가 서비스 유닛으로 데몬의 기동·재시작·로깅을 관리합니다.

관련 개념 setsid(), 세션과 프로세스 그룹, systemd 서비스 유닛, nohup

Q15. 시그널(Signal)이란 무엇이고 어떻게 처리되나요? ★★

답안 시그널은 프로세스에게 비동기적으로 이벤트를 알리는 소프트웨어 인터럽트입니다. SIGTERM(정상 종료 요청), SIGKILL(강제 종료, 무시·처리 불가), SIGINT(Ctrl+C), SIGSEGV(잘못된 메모리 접근), SIGCHLD(자식 종료) 등이 있습니다. 프로세스는 시그널을 기본 동작으로 처리하거나, 핸들러를 등록해 직접 처리하거나, 무시할 수 있습니다(SIGKILL/SIGSTOP 제외). 운영 관점에서 kill -15로 정상 종료를 시도한 뒤 안 되면 kill -9를 쓰는 것이 안전한 순서입니다.

관련 개념 kill 명령, 시그널 핸들러, SIGKILL vs SIGTERM, graceful shutdown

---

Q16. 프로세스 우선순위(nice 값)는 무엇인가요? (리눅스 운영 연계) ★

답안 리눅스에서 nice 값은 -20(높은 우선순위)부터 19(낮은 우선순위)까지이며, 기본값은 0입니다. CFS 스케줄러는 nice 값에 따라 CPU 시간 배분 가중치를 달리합니다. nice/renice 명령으로 조정하며, 우선순위를 높이는 것(-값)은 루트 권한이 필요합니다. 배치성 작업을 nice 19로 돌려 서비스 프로세스에 영향을 줄이는 식으로 운영에 활용합니다. 이와 별개로 실시간 정책(SCHED_FIFO/RR)은 일반 프로세스보다 항상 우선합니다.

관련 개념 nice/renice, CFS 가중치, 실시간 스케줄링 정책, ionice
