↑ 목록으로 ← 이전 단원 다음 단원 →

2단원. 사용자 권한과 프로세스 관리

> 대상: 전공자 · 정보처리기사 취득자 | 목적: 인프라 직무(네트워크·클라우드·시스템·보안) 기술 면접 대비 중요도: ★★★ 최빈출 · ★★ 자주 출제 · ★ 기본 개념

> **Q1.** 리눅스의 사용자와 그룹 개념, **/etc/passwd**와 **/etc/shadow**의 차이를 설명하세요. ★★★

답안 리눅스는 다중 사용자 운영체제로, 모든 파일과 프로세스는 특정 사용자(**UID**)와 그룹(**GID**)에 소속됩니다. `/etc/passwd`에는 사용자 계정명, UID, GID, 홈 디렉토리, 기본 셸 등 계정 정보가 저장되며 시스템의 모든 사용자가 읽을 수 있습니다. `/etc/shadow`에는 실제 암호화된 비밀번호와 비밀번호 만료 정책이 저장되며 root만 읽을 수 있도록 권한이 제한되어 있는데, 이는 과거 `/etc/passwd`에 평문에 가까운 해시가 그대로 노출되어 있던 보안 문제를 해결하기 위해 분리된 것입니다. 그룹 정보는 `/etc/group`에 저장되며, 한 사용자는 하나의 기본(**primary**) 그룹과 여러 개의 보조(**secondary**) 그룹에 속할 수 있습니다. 이 구조는 파일 권한 체계(소유자/그룹/기타) 및 `sudo` 권한 부여 방식과 직접 연결됩니다.

관련 개념 UID/GID, /etc/shadow 권한, primary/secondary 그룹, /etc/group, 쉐도우 패스워드 분리

> **Q2.** 리눅스 파일 권한(**rwx**)과 **chmod**의 숫자/기호 표기법을 설명하세요. ★★★

답안 리눅스 파일 권한은 소유자(**owner**), 그룹(**group**), 기타(**other**) 세 주체에 대해 각각 읽기(**r=4**), 쓰기(**w=2**), 실행(**x=1**) 권한을 부여하는 구조입니다. 예를 들어 `chmod 754 file`은 소유자에게 rwx(7), 그룹에게 r-x(5), 기타에게 r--(4) 권한을 부여합니다. 기호 표기법은 `chmod u+x,g-w,o=r file`처럼 `u`(user)/`g`(group)/`o`(other)/`a`(all)와 `+`/`-`/`=`를 조합해 특정 권한만 변경할 수 있습니다. 디렉토리에서 실행(x) 권한은 파일 목록이 아니라 해당 디렉토리 내부로 진입(**cd**)하거나 파일에 접근할 수 있는 권한을 의미한다는 점이 자주 헷갈리는 포인트로 출제됩니다. 소유자와 그룹 변경은 `chown`, `chgrp`으로 수행합니다.

관련 개념 chown/chgrp, SUID/SGID/Sticky bit, 디렉토리 실행권한의 의미, 8진수 권한 표기

> **Q3. umask**란 무엇이며 파일과 디렉토리 생성 시 어떻게 적용되는지 설명하세요. ★★

답안 **umask**는 새로 생성되는 파일과 디렉토리의 기본 권한에서 "빼줄" 권한을 지정하는 마스크 값입니다. 리눅스에서 파일의 기본 최대 권한은 666(rw-rw-rw-), 디렉토리는 777(rwxrwxrwx)이며, 여기에 umask 값을 비트 단위로 빼서 실제 생성 권한이 결정됩니다. 예를 들어 umask가 022이면 파일은 666-022=644(rw-r--r--), 디렉토리는 777-022=755(rwxr-xr-x)로 생성됩니다. 파일에는 기본적으로 실행 권한이 부여되지 않기 때문에 umask 계산 시 실행 비트가 이미 0인 자리는 그대로 유지된다는 점에 유의해야 합니다. umask는 `~/.bashrc`나 `/etc/profile`에 설정해 사용자별/시스템 전역 기본 권한 정책을 관리하는 데 사용됩니다.

관련 개념 기본 권한 666/777, umask 022, .bashrc 설정, 보안 정책 기본값

> **Q4.** 리눅스 프로세스의 상태(**R, S, D, Z, T**)를 설명하세요. ★★★

답안 **R(Running/Runnable)**은 실제로 CPU를 사용 중이거나 실행 대기열에 있는 상태이고, **S(Sleeping, interruptible)**는 특정 이벤트(입력, 타이머 등)를 기다리며 대기 중이지만 시그널에 의해 깨어날 수 있는 상태입니다.
D(Uninterruptible Sleep)는 주로 디스크 I/O 완료를 기다리는 상태로, 시그널로도 깨울 수 없어 이 상태가 오래 지속되면 하드웨어나 I/O 병목 문제를 의심해야 합니다. **Z(Zombie)**는 프로세스가 종료되었지만 부모 프로세스가 `wait()`로 종료 상태를 회수하지 않아 프로세스 테이블에 정보만 남아 있는 상태이고, **T(Stopped)**는 `SIGSTOP` 등으로 일시 정지된 상태입니다. `ps aux`나 `top`의 STAT 컬럼에서 이 상태들을 확인할 수 있으며, 특히 D 상태와 Z 상태는 장애 진단에서 자주 다뤄집니다.

관련 개념 ps aux STAT, D 상태와 I/O 병목, 좀비 프로세스, top, SIGSTOP/SIGCONT

> **Q5.** 주요 시그널(**SIGTERM, SIGKILL, SIGHUP, SIGINT**)의 차이를 설명하세요. ★★

답안 시그널은 프로세스에게 특정 이벤트가 발생했음을 비동기적으로 알리는 소프트웨어 인터럽트입니다. **SIGTERM(15)**은 프로세스에게 "정상적으로 종료해 달라"고 요청하는 신호로, 프로세스가 이를 받아 자원 정리 후 스스로 종료할 기회를 가지며 프로세스가 직접 핸들러를 등록해 가로챌 수 있습니다. **SIGKILL(9)**은 커널이 프로세스를 즉시 강제 종료시키는 신호로, 프로세스가 이를 가로채거나 무시할 수 없어 자원 정리 없이 바로 종료되므로 최후의 수단으로만 사용해야 합니다.
SIGHUP(1)은 원래 터미널 연결이 끊겼음을 알리는 신호였으나 현재는 많은 데몬에서 "설정 파일을 재로드하라"는 용도로도 관행적으로 사용됩니다. **SIGINT(2)**는 사용자가 `Ctrl+C`를 눌렀을 때 발생하는 신호로 SIGTERM과 마찬가지로 가로채거나 무시할 수 있습니다. `kill -9`를 남용하면 데이터 정합성 문제가 생길 수 있다는 점이 실무 면접에서 자주 강조됩니다.

관련 개념 kill -9 vs kill -15, 시그널 핸들러, nohup, trap, graceful shutdown

Q6. 데몬(**daemon**) 프로세스란 무엇인지 설명하세요. ★

답안 데몬은 사용자와의 직접적인 상호작용(터미널 입출력) 없이 백그라운드에서 지속적으로 실행되며 특정 서비스를 제공하는 프로세스로, 이름 끝에 흔히 `d`가 붙습니다(예: `sshd`, `httpd`, `crond`). 데몬은 일반적으로 부모 프로세스와의 연결을 끊고(제어 터미널 분리) 독립적인 세션을 생성하는 더블 포크(**double fork**) 방식으로 생성되어 부모 프로세스가 종료되어도 영향을 받지 않습니다. 표준입출력을 `/dev/null`이나 로그 파일로 재지정해 터미널 종료와 무관하게 계속 실행되도록 하는 것이 일반적입니다. 현대 리눅스에서는 systemd가 데몬의 생성, 모니터링, 재시작을 관리하는 표준적인 방식으로 자리 잡았습니다.

관련 개념 더블 포크, 제어 터미널 분리, systemd unit, nohup/setsid

Q7. init과 **systemd**의 차이를 설명하세요. ★★

답안 전통적인 **SysVinit**은 `/etc/rc.d`의 런레벨(runlevel) 스크립트를 순차적으로 실행하는 방식으로, 서비스 간 의존 관계 처리가 단순하고 병렬 처리가 어려워 부팅 속도가 느린 편이었습니다. **systemd**는 PID 1로 동작하며 서비스 간 의존성을 명시적으로 선언하고 병렬로 기동할 수 있어 부팅 속도를 크게 개선했고, 유닛(unit) 파일 단위로 서비스, 소켓, 타이머, 마운트 등을 통합 관리합니다. `systemctl start/stop/restart/status`로 서비스를 제어하고, `systemctl enable`로 부팅 시 자동 시작 여부를 설정하며, `journalctl`로 로그를 통합 조회할 수 있는 점이 실무에서 자주 활용됩니다. 런레벨 개념은 systemd의 **target**(예: `multi-user.target`, `graphical.target`)으로 대체되었습니다.

관련 개념 유닛 파일, systemctl enable/status, journalctl, target vs runlevel

---

> **Q8. cron**을 이용한 작업 스케줄링 방법과 **crontab** 표기법을 설명하세요. ★★

답안 **cron**은 지정된 시각이나 주기에 맞춰 명령어나 스크립트를 자동으로 실행해 주는 데몬(`crond`)이며, 사용자는 `crontab -e` 로 자신의 스케줄을 등록합니다. crontab 표기법은 `분 시 일 월 요일 명령어` 순서의 5개 필드로 구성되며, `*` 는 모든 값, `*/5` 는 5단위 간격, `1-5` 는 범위, `1,3,5` 는 특정 값 나열을 의미합니다. 예를 들어 `0 2 * * * /home/user/backup.sh` 는 매일 새벽 2시에 백업 스크립트를 실행하라는 의미입니다. 시스템 전체 작업은 `/etc/crontab` 이나 `/etc/cron.d/` 에 사용자명을 포함해 등록하며, cron 환경에서는 사용자의 로그인 셸 환경변수(PATH 등)가 로드되지 않는 경우가 많아 스크립트 내에서 절대경로를 사용하거나 환경변수를 명시적으로 설정해야 하는 점이 실무에서 자주 지적되는 함정입니다.

관련 개념 crontab -e, */5 * * * *, /etc/cron.d, PATH 환경변수 문제, anacron

---

> **Q9.** 좀비 프로세스와 고아 프로세스의 차이를 설명하세요. ★★

답안 좀비 프로세스는 자식 프로세스가 실행을 마치고 종료(exit)했지만, 부모 프로세스가 `wait()` 시스템 콜을 호출해 종료 상태 코드를 회수하지 않아 프로세스 테이블에 최소한의 정보만 남아 있는 상태입니다. 좀비 프로세스는 실제 자원을 거의 사용하지 않지만 프로세스 테이블 슬롯을 차지하므로, 대량으로 쌓이면 새로운 프로세스를 생성할 수 없는 문제가 발생할 수 있습니다. 고아 프로세스는 반대로 부모 프로세스가 자식보다 먼저 종료된 경우로, 이때 커널은 해당 자식 프로세스를 **init(또는 systemd, PID 1)**의 자식으로 재입양시켜 정상적으로 회수(reap)되도록 처리합니다. 즉 좀비는 부모가 살아있는데 회수를 안 하는 문제, 고아는 부모가 먼저 죽는 상황이라는 차이가 핵심입니다. 좀비 프로세스를 없애려면 부모 프로세스를 종료시키거나(재입양되어 init이 회수) 부모 코드에서 `wait()` 를 호출하도록 수정해야 합니다.

관련 개념 wait()/waitpid(), PID 1 재입양, 프로세스 테이블, fork-exec 모델

> **Q10. su와 sudo의 차이를 설명하세요.** ★

답안 `su`는 다른 사용자(기본값은 root)로 완전히 전환하는 명령으로, 대상 계정의 비밀번호를 알아야 하며 전환 이후에는 해당 사용자 권한으로 셸이 계속 유지됩니다. `sudo`는 자신의 계정 비밀번호로 인증한 뒤 `/etc/sudoers`에 정의된 정책에 따라 특정 명령어만 관리자 권한으로 일시적으로 실행하는 방식으로, 실행되는 모든 명령이 로그에 기록되어 감사(audit) 추적이 가능하다는 보안상의 장점이 있습니다. `sudo`는 사용자별로 실행 가능한 명령어를 세밀하게 제한할 수 있어 root 비밀번호를 여러 사람과 공유할 필요가 없다는 점에서 다중 관리자 환경에 더 적합합니다. `visudo` 명령으로 `/etc/sudoers`를 안전하게 편집하는 것이 권장됩니다.

관련 개념 /etc/sudoers, visudo, 감사 로그, 최소 권한 원칙
