Operating Systems - Chapter 5 Summary

Process API (UNIX)

• Key system calls for process management: fork(), exec(), wait().
• These enable creation, execution, and control of processes.

The fork() System Call

• Creates a new process (child) as a near-identical copy of the parent.
• Both processes continue execution after fork().
• Return value differentiates: parent gets child PID, child gets 0.
• Execution order is non-deterministic, depends on scheduler.

The wait() System Call

• Parent process waits until the child finishes.
• Ensures deterministic output by synchronizing execution.

The exec() System Call

• Replaces the current process image with a new program.
• Does not create a new process, but transforms the current one.
• Variants exist: execl(), execv(), execvp(), etc.

Why fork() + exec()?

• This separation allows shells to modify environment before running a program.
• Enables features like redirection and pipes.

Process Control and Signals

• kill() sends signals (pause, stop, terminate, etc.).
• Common signals: SIGINT (Ctrl-C), SIGTSTP (Ctrl-Z).
• Processes can catch and handle signals with signal().
• Users can only control their own processes; root (superuser) can control all.

Useful Tools

• ps – list processes.
• top – live view of processes and resource usage.
• kill/killall – send signals to processes.

Summary

The UNIX process API centers on fork(), wait(), and exec(). Together, they provide flexible and powerful process management. Additional tools and signals extend control, enabling process synchronization, I/O redirection, and system-level management. This API design is simple yet extremely powerful, forming the foundation of UNIX shells and process handling.