Operating Systems - Chapter 2 Summary

The Abstraction: The Process

• A process is a running program (program + execution context).
• Programs on disk are lifeless; the OS loads them into memory and makes them run.
• Illusion of many CPUs: OS time-shares the CPU among processes.

Mechanism vs. Policy

• Mechanisms: Low-level methods (e.g., context switching).
• Policies: High-level decisions (e.g., scheduling which process runs next).
• Separating the two improves modularity and flexibility.

Process API

• Create – start new processes.
• Destroy – terminate processes.
• Wait – wait for process completion.
• Miscellaneous control – suspend, resume.
• Status – get info about running processes.

Process Creation

• Loading program code + static data from disk into memory.
• Allocate memory for stack (function calls, local variables).
• Allocate heap for dynamic memory (malloc/free).
• Initialize I/O (stdin, stdout, stderr).
• Transfer control to entry point (main()).

Process States

• Running – process is executing instructions.
• Ready – waiting to be scheduled.
• Blocked – waiting for an event (e.g., I/O).
• State transitions: scheduled, descheduled, blocked, unblocked.

OS Data Structures

• OS tracks processes using structures like the Process Control Block (PCB).
• PCB includes: PID, state, memory info, registers, parent, open files, etc.
• Process list/task list: OS maintains info about all processes.

Summary

A process is the fundamental OS abstraction of a running program. The OS manages process creation, execution, and termination, tracks their states, and provides APIs for control. With mechanisms (e.g., context switch) and policies (e.g., scheduling), the OS virtualizes the CPU and enables multitasking.