Lecture 08

OS Overview & Process Models

OS Objectives · Services · ISA/ABI/API · Evolution · Resource Management

SENG 21213 · Computer Architecture & Operating Systems

Learning Objectives — By the end of this lecture you should be able to:

- 1. State the three OS objectives and explain each with a concrete example
- 2. Describe the seven OS services and identify which applies to a given scenario
- 3. Distinguish the ISA, ABI, and API interface layers and explain what each enables
- 4. Trace the evolution of OS from serial processing to time-sharing with the problem each generation solved
- 5. Describe the four OS control table types and their interrelationships
- 6. Identify the four major OS achievements and describe Stage 0 of the OS assignment

Section 1 · OS Objectives and Services

📖 Stallings Reference

- Chapter 2: Operating System Overview
- Section 2.1 — Operating System Objectives and Functions
- Section 2.2 — The Evolution of Operating Systems

An Operating System (OS) is the software layer between the hardware and user applications. It manages hardware resources, provides abstractions, and enforces protection policies.

	Uniprogramming	Multiprogramming
Processor use	20%	40%
Memory use	33%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min

Figure: Layered view of a computer system: hardware → OS kernel → OS utilities → applications → user (Stallings Fig. 2.1)

1.1 Three Core Objectives

Objective	Description	Without It…
Convenience	Makes the computer easier to use by hiding hardware complexity. Provides high-level abstractions: files instead of disk sectors; sockets instead of network frames.	Programmers would write device drivers for every application — hardware-specific, brittle, non-portable.
Efficiency	Allows hardware resources to be used efficiently. Maximises CPU utilisation (multiprogramming), memory throughput (virtual memory), and I/O bandwidth (DMA, buffered I/O).	Resources sit idle. One process monopolises CPU while others wait. Memory is wasted.
Ability to Evolve	The OS must support new hardware, new services, and bug fixes without breaking existing applications. Stable ABIs allow software to run across kernel versions.	Every hardware change or bug fix requires rewriting all applications.

1.2 Seven OS Services

Service	Description
Program Development	Editors, compilers, debuggers, linkers — the toolchain is part of the OS environment.
Program Execution	OS loads program code and data into RAM, initialises stack, PCB, and I/O, then dispatches the process to the CPU.
Access to I/O Devices	Provides a uniform read/write interface — programmers do not need to know the 200-page register spec for each device.
Controlled Access to Files	Understands file system structure; enforces read/write/execute permissions for multi-user safety.
System Access Control	Mediates access to the entire system; resolves resource contention; enforces quotas and authentication.
Error Detection and Response	Handles hardware errors (parity, ECC), software errors (divide-by-zero, segfault), and recovers or terminates gracefully.
Accounting	Tracks per-user and per-process CPU time, memory usage, and I/O for billing, performance analysis, and optimisation.

1.3 ISA, ABI, and API Interface Layers

	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	75 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes

Figure: System interfaces: ISA between hardware and OS; ABI between OS and user programs; API between libraries and applications (Stallings Fig. 2.3)

Interface	Between	Purpose
ISA — Instruction Set Architecture	Hardware and Software (OS and applications)	Defines machine instructions, registers, addressing modes. System ISA: privileged instructions (IN, OUT, HLT) for OS only. User ISA: available to all code.
ABI — Application Binary Interface	OS kernel and user applications	Defines system call numbers, register conventions, binary format (ELF/PE). Enables a compiled binary to run on any OS version with the same ABI.
API — Application Programming Interface	Libraries and application source code	High-level function calls (read(), malloc(), pthread_create()) that translate to ABI calls. Same source code compiles on different platforms.

Section 2 · Evolution of Operating Systems

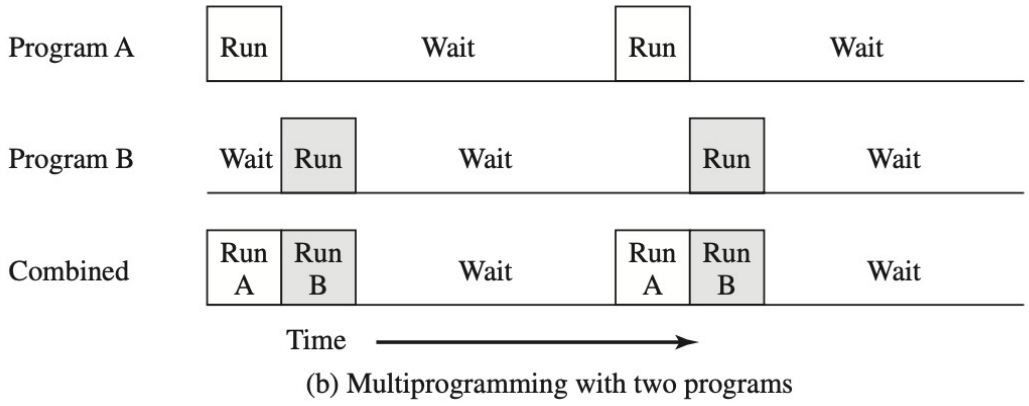


Figure: Timeline of OS evolution from serial processing through batch to multiprogrammed and time-sharing systems (Stallings Fig. 2.4)

Era	Problem Solved	New Problem Introduced	Key OS Feature
Serial Processing (1940s–50s)	Direct hardware programming	Scheduling waste, setup time	None — no OS

Era	Problem Solved	New Problem Introduced	Key OS Feature
Simple Batch (mid-1950s)	Automates job sequencing; Resident Monitor runs without human intervention	CPU idle during I/O — 30–40% utilisation typical	Resident Monitor, timer interrupt, memory protection
Multiprogrammed Batch (1960s)	Keeps CPU busy by overlapping CPU and I/O of multiple jobs	Complex memory management; race conditions between jobs	Job scheduler, memory partitioning, I/O interrupt handling
Time-Sharing (1967+)	Interactive response for multiple simultaneous users (CTSS at MIT)	Deadlocks, starvation, security between users	Preemptive scheduling, virtual memory, user accounts, file protection

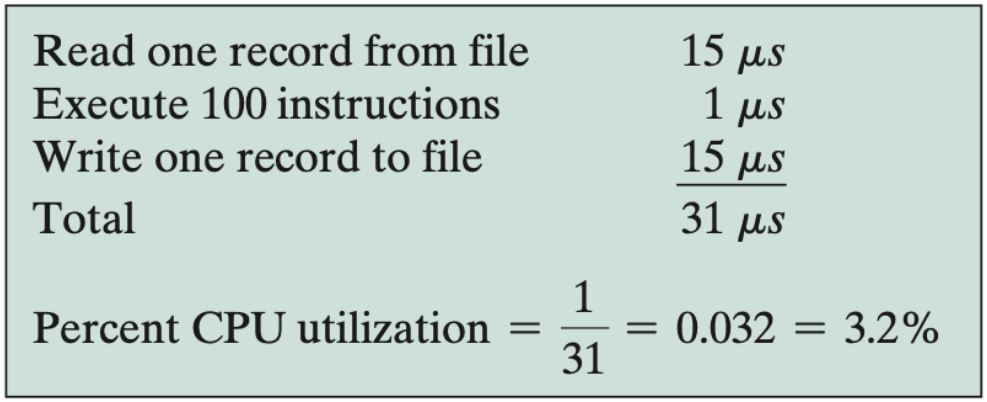


Figure: Multiprogramming: CPU switches between jobs while I/O proceeds, improving CPU utilisation dramatically (Stallings Fig. 2.9)

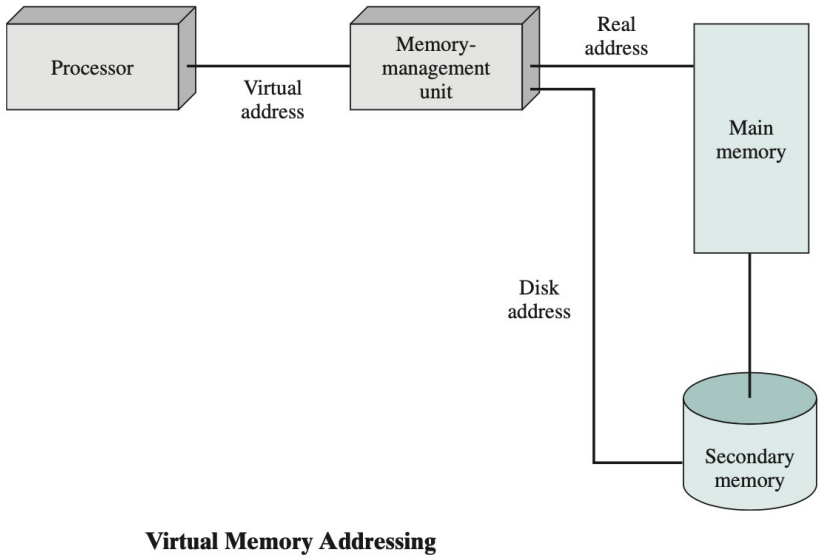


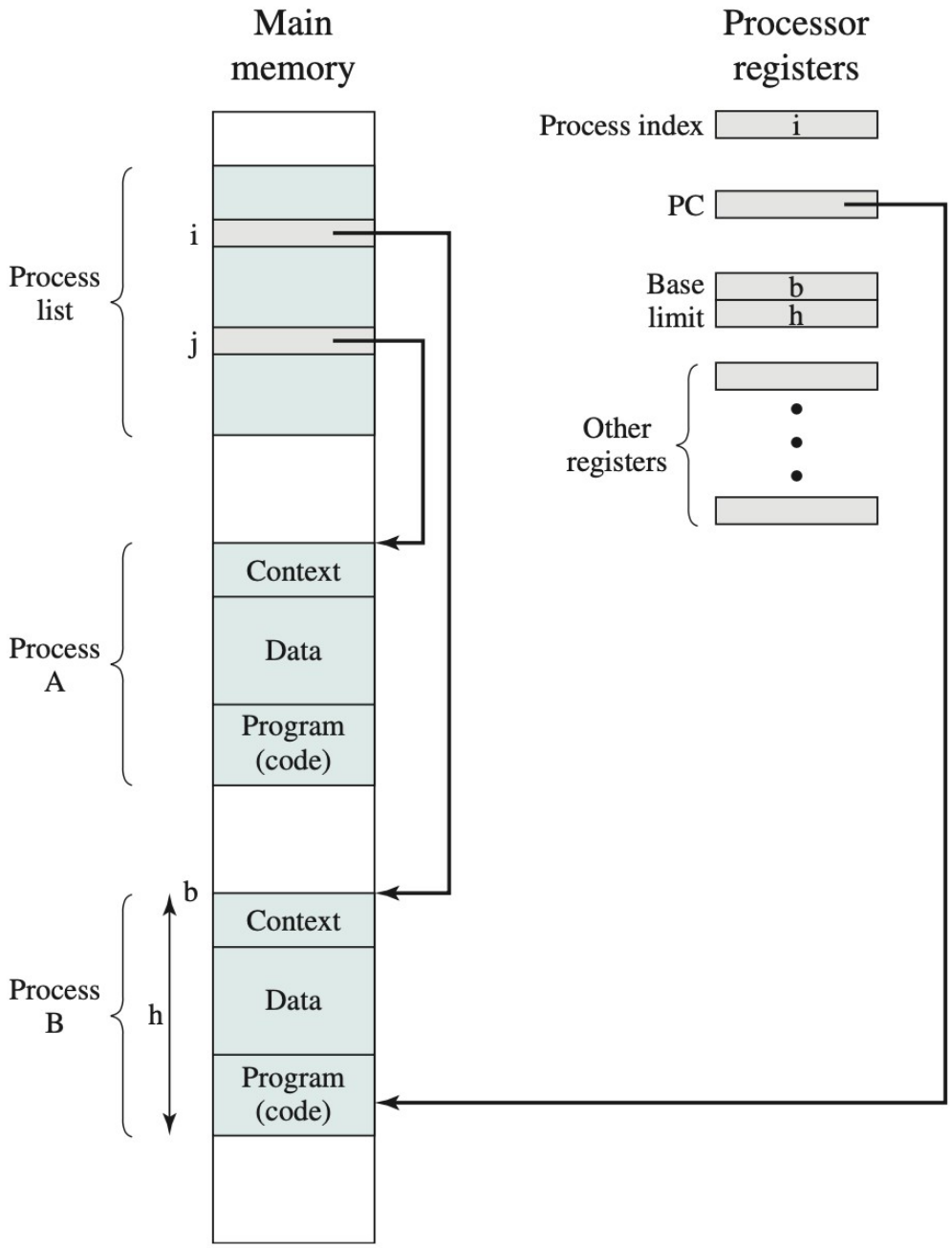
Figure: Time-sharing: multiple interactive users each receive a time quantum of CPU — appears simultaneous to each user (Stallings Fig. 2.12)

Section 3 · OS as Resource Manager — Control Structures

📖 Stallings Reference

- Chapter 3: Process Description and Control
- Section 3.1 — What Is a Process? Section 3.2 — Process States

3.1 The Four OS Control Table Types



Typical Process Implementation

Figure: OS control tables: process table, memory table, I/O table, and file table and their interrelationships (Stallings Fig. 3.4)

Table Type	Contents	Role
Process Table	One entry per process. Pointer to PCB; process state; PID.	Primary management structure — all other tables are cross-referenced from here.
Memory Tables	Virtual-to-physical address maps; page tables; memory segment	Tracks what memory is allocated to which process; enforces isolation.

Table Type	Contents	Role
	bounds; protection attributes.	
I/O Tables	Which device is allocated to which process; current I/O operation status; data buffer addresses.	Prevents two processes simultaneously using the same device; queues pending I/O.
File Tables	Open file list; file size, location, type; inode/FCB pointer; access mode (read/write/execute).	Tracks all open files system-wide; per-process open file descriptor table points here.

3.2 Scheduling Queues

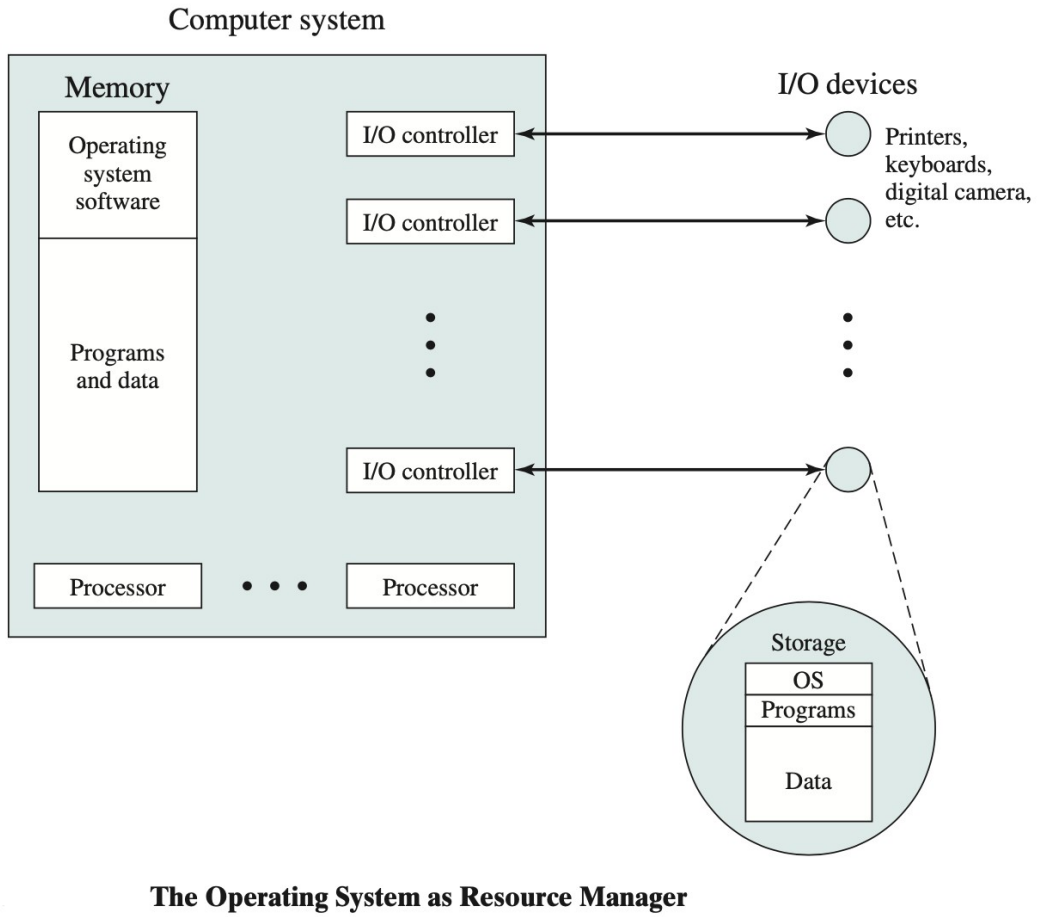


Figure: OS queuing model — processes flow from job queue to ready queue to CPU and back through I/O queues (Stallings Fig. 3.8)

Queue	Managed By	Contents	Decision
Long-term (Job) Queue	Long-term scheduler	New processes waiting to be admitted to memory	Admits processes when memory and CPU capacity available; controls degree of multiprogramming

Queue	Managed By	Contents	Decision
Ready Queue	Short-term scheduler (dispatcher)	Processes in memory, ready to run, waiting for CPU	Selects which process runs next; runs very frequently (every 10–100 ms)
I/O Wait Queue (per device)	Device driver / I/O manager	Processes blocked waiting for a specific device	FIFO or priority-ordered; process released when I/O completes

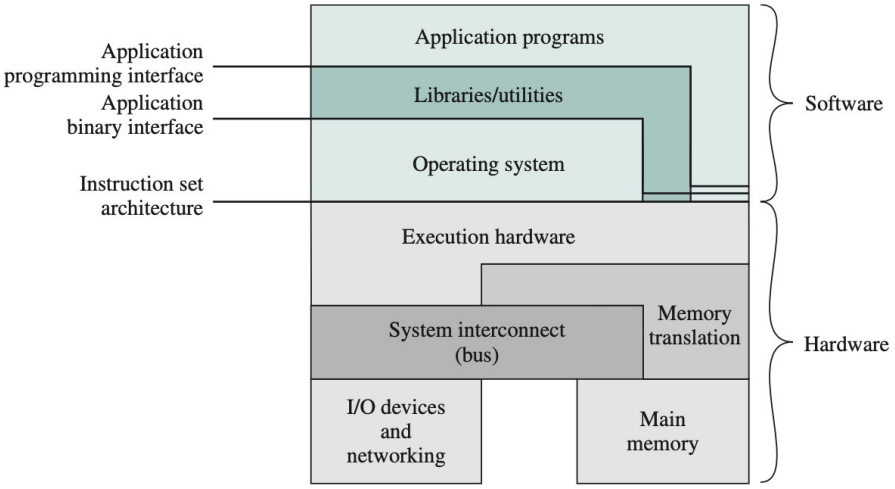
3.3 Four Major OS Achievements

Achievement	Description
Processes	A process is a program in execution — the fundamental OS abstraction. Solves the problems of isolation, synchronisation, mutual exclusion, nondeterminism, and deadlocks.
Memory Management	Five responsibilities: (1) process isolation, (2) automatic allocation/deallocation, (3) modular programming support, (4) protection and access control, (5) long-term storage (file system).
Information Protection & Security	Four goals: Availability (service remains accessible), Confidentiality (data is hidden from unauthorised users), Data Integrity (data is not modified without authorisation), Authenticity (identities and message sources are verified).
Scheduling and Resource Management	Three factors: Fairness (no indefinite starvation), Differential Responsiveness (interactive jobs respond quickly; batch jobs maximise throughput), Efficiency (maximise throughput and minimise idle time).

Section 4 · The Process Concept & Stage 0

◆ Definition — Process

- A process is a program in execution, consisting of three parts:
- 1. Executable Program — the machine code to be executed
- 2. Associated Data — the memory regions: code segment, data segment, heap, and stack
- 3. Execution Context (PCB) — the internal data the OS uses to manage and control the process, including PC, registers, state, priority, and resource tables



Computer Hardware and Software Structure

Figure: Process image in memory showing code, data, heap, stack segments and PCB location (Stallings Fig. 3.5)

★ Stage 0 OS Assignment — File Structure

- boot/boot.asm — 512-byte MBR. Loads 64 disk sectors (32 KB) using INT 0x13 extended read. Sets up 3-entry GDT. Sets CR0 bit 0. Far jump to Protected Mode.
- kernel/kernel_entry.asm — Protected Mode entry point. Sets DS/ES/SS to data segment selector 0x10. Sets ESP = 0x90000. Calls kernel_main().
- kernel/kernel.c — OS entry: vga_init(); kb_init(); display splash screen; run shell loop calling kb_readline() + parse_command().
- kernel/vga.c + vga.h — 80×25 VGA text driver. vga_write_char(row, col, c, colour). vga_printf(fmt, ...). Hardware cursor control via ports 0x3D4/0x3D5.
- kernel/keyboard.c + keyboard.h — PS/2 keyboard driver. Poll port 0x60. scancode_to_ascii[] table. kb_getchar() and kb_readline().
- Build: make all → links to kernel.bin → make run → boots in QEMU. SENG OS splash screen = Stage 0 complete.

Section 5 · Practice Exercises

✎ Exercise 8.1 [10 marks] — OS Objectives and Services

- (a) [3] For each scenario below, identify which OS service is primarily responsible and explain how it fulfils its role: (i) A web browser opens a TCP socket. (ii) A C program calls printf() which writes to the terminal. (iii) A process divides by zero.
- (b) [4] A computer runs without an OS (bare metal). A programmer writes an application that reads from a disk file. List the low-level operations the programmer must implement themselves (at least 5 steps) that the OS would otherwise provide. How does this demonstrate the 'convenience' objective?
- (c) [3] Explain the difference between the ISA and the ABI. A programmer compiles a C program on Linux for x86-64. It runs correctly. The same binary is copied to a Windows x86-64 machine

and fails. Which interface layer (ISA, ABI, or API) is the incompatibility at, and why?

✎ Exercise 8.2 [12 marks] — Evolution of OS

- (a) [4] Three jobs are submitted to a simple batch OS: Job A (5 min CPU, no I/O), Job B (2 min CPU + 8 min I/O), Job C (3 min CPU + 2 min I/O). (i) Uniprogramming: calculate total wall-clock time (sequential execution). (ii) Multiprogramming: construct a Gantt chart showing overlapped execution. What is the total time? (iii) Calculate CPU utilisation for both cases.
- (b) [4] The CTSS time-sharing system used 200 ms time quanta. (i) If 10 users are active and the context switch takes 1 ms, what is the worst-case response time for a user keypress? (ii) If the quantum is increased to 1000 ms, how does response time and throughput change? (iii) What is the ideal quantum size trade-off?
- (c) [4] Compare the protection mechanisms introduced at each OS generation: (i) What hardware feature was introduced in batch systems to prevent user programs from modifying the monitor? (ii) What mechanism prevents a job from running indefinitely in batch systems? (iii) What OS feature was needed in time-sharing but not in batch to protect user data from other users?