ABHYASMITRA.IN

COMPREHENSIVE UNIVERSITY STUDY NOTES

OPERATING SYSTEM

Unit I — Introduction to Operating System and Process

Topics Covered in this Unit

Introduction to Operating Systems: Roles (Resource Manager & Extended Machine) and Dual-Mode Operation

Taxonomy of Operating Systems: Batch, Multiprogrammed, Time-Sharing, Distributed & Real-Time OS (Hard vs. Soft RTOS)

The Linux Operating System: Architecture, Monolithic vs. Microkernel, System Call Interface & GNU Userland

Process Abstraction & Memory Layout: Code (Text), Data, BSS, Heap, and Stack Segments

Process Life Cycle & States: 5-State Model (New, Ready, Running, Blocked, Terminated) & 7-State Suspended Model

Process Control Block (PCB / Linux task_struct): State, PID, PC, Registers, Memory Limits & File Descriptors

Process Execution Mechanisms: Limited Direct Execution (LDE), Hardware Traps, System Calls & Context Switching

Process API in Unix/Linux: fork(), exec() Family, wait()/waitpid(), exit(), Zombie & Orphan Processes

Process Control, Users & Permissions: User/Group IDs, Privilege Levels & Inter-Process Signals (SIGKILL, SIGTERM)

High-Yield University Exam Question Bank with Detailed Model Answers & Unit Summary

Compiled in a structured, easy-to-understand, textbook style format
Specially curated for B.Tech / B.E. / BCA / MCA / B.Sc Computer Science & Information Technology Students

COURSE SYLLABUS & MAPPING

[UNIT I SYLLABUS — PRESCRIBED UNIVERSITY CURRICULUM]

This section contains the official syllabus for Unit I. Verify with your university curriculum mapping.

Unit I Syllabus Breakdown:

Unit I - Introduction to Operating System and Process: Introduction to operating systems, Types of OS, real time OS, the Linux Operating Systems. Process: process abstraction, system calls for process management, process creation: process states, data structures, process execution mechanisms process api, process control and users

• Course Code: _____ • Semester / Year: _____

TABLE OF CONTENTS

1.1 Introduction to Operating Systems & Fundamental Roles

Definition: Operating System (OS)

An Operating System (OS) is system software that acts as an intermediary between computer hardware and user applications. It fulfills two primary roles:

1. Extended Machine / Virtual Machine (Illusionist): Transforms raw, complex, low-level hardware into clean, user-friendly high-level abstractions (Processes, Virtual Address Spaces, Files).

2. Resource Manager (Government): Multiplexes physical hardware resources (CPU, Physical RAM, I/O devices, Disk space) efficiently, fairly, and securely among multiple competing programs.

Dual-Mode Operation & Hardware Protection

Modern CPUs provide hardware-enforced dual-mode operation governed by a Mode Bit in the processor status register:

Mode Dimension	User Mode (Ring 3 / Mode Bit = 1)	Kernel Mode (Ring 0 / Mode Bit = 0 / Supervisor)
Privilege Level	Restricted, unprivileged execution level. Applications run here.	Unrestricted, fully privileged access to all CPU instructions and physical memory.
Hardware Access	Cannot directly execute I/O instructions, disable interrupts, or modify page tables.	Full direct control over CPU control registers, memory management units (MMU), and I/O devices.
Transitions	Transitions to Kernel Mode via hardware Traps (System Calls, Page Faults) or Interrupts.	Transitions back to User Mode via privileged 'return-from-trap' instruction (e.g., IRET/SYSRET).

1.2 Taxonomy of Operating Systems

OS Architecture Type	Core Operational Mechanism	Key Characteristics & Tradeoffs	Typical Use Cases
Batch Processing OS	Jobs with similar requirements are grouped into batches and processed sequentially by an operator without user interaction.	High CPU utilization; poor turnaround time; zero interactive debugging.	Early mainframe payroll processing, scientific batch jobs.

OS Architecture Type	Core Operational Mechanism	Key Characteristics & Tradeoffs	Typical Use Cases
Multiprogrammed OS	Keeps multiple jobs in main memory simultaneously; when current job performs I/O, CPU switches to another ready job.	Eliminates CPU idle time during I/O; requires memory protection hardware.	Enterprise database servers, multi-user mainframe servers.
Time-Sharing / Multitasking OS	CPU time is multiplexed across multiple active user processes using rapid round-robin time slicing (e.g., 10-100ms).	Provides interactive responsiveness; gives each user the illusion of dedicated machine.	Unix, Linux, Windows, macOS desktops and cloud servers.
Distributed OS	Manages a collection of independent networked computers, presenting them to users as a single unified system.	High fault tolerance, horizontal scalability, complex consensus & network synchronization.	Amoeba, Plan 9, Google Borg / Kubernetes cluster orchestrators.
Real-Time OS (RTOS)	Guarantees deterministic execution and strict task completion within defined physical deadlines.	Strict predictability prioritized over average throughput. Deterministic context switch times.	Automotive engine control (ECU), pacemakers, aerospace fly-by-wire.

Real-Time Operating Systems: Hard vs. Soft Real-Time

- • **Hard Real-Time Systems:** Strict deadlines where missing a single deadline results in total, catastrophic system failure (e.g., aircraft flight control, anti-lock braking systems ABS, missile guidance).
- • **Soft Real-Time Systems:** Deadlines are important, but missing a deadline causes degraded service quality without catastrophic harm (e.g., live video streaming, VoIP audio calls, AAA multiplayer gaming).

1.3 The Linux Operating System & Kernel Architecture

- • **Monolithic Kernel Architecture:** Linux implements a monolithic kernel design where all core operating system services (Process scheduler, Virtual Memory Manager, Virtual File System [VFS], Network Protocol Stack, and Hardware Device Drivers) execute within a single shared kernel address space in Ring 0 for maximum execution speed.
- • **Loadable Kernel Modules (LKMs):** Provides modularity without performance penalty; device drivers and file systems can be dynamically compiled, loaded (`insmod`), and unloaded (`rmmod`) into the running kernel on the fly.

- • **Monolithic vs. Microkernel:** Microkernels (e.g., Mach, QNX, Minix) run only minimal scheduling, IPC, and low-level memory in Ring 0, moving file systems and drivers to User Space servers (Ring 3) via message passing (higher modularity and crash resilience, but message passing overhead).

1.4 Process Abstraction & Memory Layout

Definition: Process

A Process is an active instance of a running computer program in execution. While a program is a passive collection of compiled binary instructions sitting dormant on disk, a process is an active entity with a program counter, CPU registers, stack, heap, and dedicated virtual address space.

VIRTUAL ADDRESS SPACE LAYOUT OF A USER PROCESS

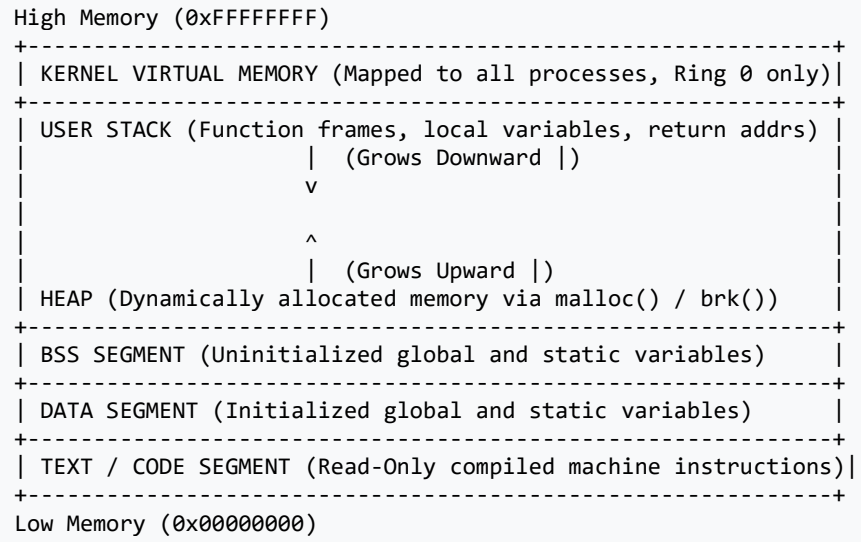

Figure: Canonical memory segmentation of a running 32-bit / 64-bit Linux user process

1.5 Process States & The Process Control Block (PCB)

The Classic 5-State Process Transition Model

- **1. New:** The process is being created and its address space is being set up.
- **2. Ready:** The process is loaded in main memory and waiting in the ready queue to be allocated CPU time by the scheduler.
- **3. Running:** The process's instructions are currently executing on the physical CPU core.

- **4. Blocked / Waiting:** The process cannot execute until an external event occurs (e.g., disk I/O completion, waiting for network packet, sleep timer).
- **5. Terminated / Zombie:** The process has finished execution and released its memory, waiting for its parent to read its exit status code.

The Process Control Block (PCB / Linux task_struct)

Definition: Process Control Block (PCB)
The PCB is the central data structure maintained by the OS kernel for every active process, storing all state information required to freeze, context-switch, and resume the process seamlessly.

- **• Key Fields in PCB / task_struct:**
  - **• Process Identifier (PID):** Unique integer identifying the process.
  - **• Process State:** Current execution state (TASK_RUNNING, TASK_INTERRUPTIBLE, TASK_UNINTERRUPTIBLE, TASK_ZOMBIE).
  - **• Program Counter (PC):** Memory address of the next machine instruction to be executed.
  - **• CPU Registers:** Saved values of general-purpose registers, stack pointer (SP), base pointer (BP), and status flags during a context switch.
  - **• CPU Scheduling Information:** Scheduling priority, dynamic priority, static priority (`nice` value), virtual runtime (`vruntime`).
  - **• Memory Management Info:** Pointers to page directory/tables (`struct mm_struct`), base/limit registers.
  - **• Open File Descriptors:** Array of pointers to open file table (`struct files_struct`), standard streams (stdin=0, stdout=1, stderr=2).

1.6 Process Execution: Limited Direct Execution (LDE)

The Limited Direct Execution (LDE) Protocol
To make programs run fast, the OS allows the program to run DIRECTLY on the physical CPU hardware. However, to maintain control and safety, the OS imposes two limits:

1. Restricted Operations: Sensitive operations (I/O, memory allocation) are trapped into the kernel via System Calls.

2. Regaining Control: The OS uses periodic Timer Interrupts to preempt running processes and schedule others.

The Context Switch Mechanism

A Context Switch is the low-level assembly procedure of switching the CPU from one running process P1 to another ready process P2:

- **Step 1:** Timer interrupt fires or P1 issues a blocking system call. CPU hardware switches to Kernel Mode (Ring 0) and pushes P1's registers onto P1's kernel stack.
- **Step 2:** Kernel saves remaining register state (including Stack Pointer) into P1's PCB.
- **Step 3:** The scheduler algorithm selects process P2 from the ready queue.
- **Step 4:** Kernel restores P2's register state from P2's PCB, switches CPU page table register (CR3) to P2's address space, and executes 'return-from-trap'.
- **Step 5:** CPU restores user registers and switches to User Mode (Ring 3), resuming execution of P2 at its saved Program Counter.

1.7 Process API: fork(), exec(), wait() & Process Life Cycle

System Call API	Syntactic Signature	Functional Behavior & Return Value
fork()	pid_t fork(void);	Creates an exact duplicate child process sharing open file table. Returns 0 to child process; returns Child's PID to parent; returns -1 on error. Uses Copy-On-Write (COW) optimization.
execvp() / execve()	int execvp(const char *file, char *const argv[]);	Replaces current process image, text, data, heap, and stack entirely with a new executable binary. Does NOT create a new PID. Only returns if error occurs.
wait() / waitpid()	pid_t wait(int *status);	Suspends calling parent process until one of its child processes terminates. Reaps child's exit status and allows kernel to fully delete child PCB.

Zombies vs. Orphan Processes

- • **Zombie Process:** A child process that has completed execution (`exit(0)`), but its parent has not yet called `wait()`. The child's memory is freed, but its entry in the process table (PCB) remains to hold its exit status code.
- • **Orphan Process:** A running child process whose parent process terminated without waiting. In Linux, orphans are immediately adopted by the root `init` / `systemd` process (PID 1), which periodically reaps terminating children.

1.8 High-Yield University Exam Question Bank & Model Answers

Part A: Short Answer Questions (2–3 Marks)

- **Q1. What is the difference between a Program and a Process?**

Answer: A Program is a passive entity (a static compiled binary stored on disk). A Process is an active entity in execution, with its own Program Counter, CPU registers, stack, heap, and virtual address space.

- **Q2. What is a Context Switch? What causes its computational overhead?**

Answer: A context switch is the mechanism of saving the state of the currently executing process into its PCB and restoring the state of another ready process. Overhead arises from saving/restoring CPU registers, switching memory address space (flushing TLB), and cache misses caused by cold cache lines.

- **Q3. Differentiate between Zombie and Orphan processes in Linux.**

Answer: A Zombie is a terminated process whose exit status has not yet been read by its parent via `wait()`, retaining an entry in the process table. An Orphan is a running process whose parent has died; it is automatically adopted and reaped by PID 1 (`systemd` / `init`).

- **Q4. Explain the return values of the fork() system call.**

Answer: `fork()` returns twice: (1) It returns `0` inside the newly created child process, (2) It returns the child's positive `PID` inside the parent process, and (3) It returns `-1` on failure (e.g., process limit reached).

Part B: Long Answer Questions (8–10 Marks)

- **Q5. Explain the Process State Transition Model in detail with a state transition diagram. Describe the contents and purpose of the Process Control Block (PCB).**

Model Answer Outline:

- 1. Definition of a process and state abstractions.
- 2. Comprehensive 5-state model: New → Ready → Running → Blocked/Waiting → Terminated; complete state transition diagram showing scheduler dispatch, timer interrupts, I/O requests, and I/O completion.
- 3. Suspended state extensions: Ready-Suspended and Blocked-Suspended in swapping scenarios.
- 4. Detailed breakdown of PCB fields: PID, State, PC, CPU Registers, Memory Limits (`mm_struct`), Open Files table, Scheduling priority.
- 5. Role of PCB during context switching and kernel process management.

- **Q6. Describe the Limited Direct Execution (LDE) protocol. Walk through a complete hardware trap and system call flow between User Mode and Kernel Mode.**

Model Answer Outline:

- 1. Motivation for LDE: High performance without sacrificing OS control.

- 2. Hardware support: Dual-mode execution (User Ring 3 vs Kernel Ring 0), Mode bit, and Trap Table initialized at boot time.
- 3. System call execution sequence: User program sets arguments → executes Trap instruction (`syscall`/`int 0x80`) → CPU switches to Kernel Mode, saves registers to kernel stack → Kernel executes system call handler → executes Return-From-Trap → CPU restores user registers, switches to User Mode, resumes user program.
- 4. Regaining CPU control: Cooperative approaches (yielding) vs Non-cooperative Timer Interrupts.
- 5. Complete timeline diagram showing user/kernel transitions.

1.9 Unit I Summary & Quick Revision Sheet

OS & Process Domain	Key Theoretical & System Concepts for Exams
OS Roles & Dual-Mode	Extended Machine (Abstractions) + Resource Manager (Multiplexing). Ring 3 (User) vs Ring 0 (Kernel). Traps switch modes.
OS Types & RTOS	Batch, Multiprogrammed, Time-Sharing. RTOS: Hard (catastrophic failure on missed deadline) vs Soft (degraded QoS).
Process Memory	Text (Code), Data (Initialized globals), BSS (Uninitialized globals), Heap (grows up), Stack (grows down).
Process States & PCB	5 States: New, Ready, Running, Blocked, Terminated. PCB (PID, PC, Registers, Memory, Open Files).
Process API & Execution	LDE protocol (Direct run + Timer preemption). `fork()` (returns 0 child, PID parent), `exec()` (replaces image), `wait()` (reaps zombies).