Syllabus

Operating System: Multiprogramming, multiprocessing, and multitasking, Virtual memory, paging, fragmentation, Mutual exclusion. Critical regions, lock and unlock, CPU scheduling, I/O scheduling, Resource scheduling, Deadlock and scheduling algorithms. Unix System, Windows System.

Detailed syllabus

✅ Operating System – Study Index for SIU PhD Entrance

1. System Concepts

- Definitions: Operating System, Kernel, Shell
- Types of OS: Batch, Time-Sharing, Real-Time, Distributed, Networked, Mobile OS

---

2. Multiprogramming, Multiprocessing, and Multitasking

- Definitions and differences
- Process states and transitions
- Context switching
- Examples: How modern OS implement these

---

3. Memory Management

- **Virtual Memory**
  - Concept, demand paging, page faults
- **Paging**
  - Paging mechanism, page table, TLB
- **Fragmentation**
  - Internal vs External, memory allocation strategies

4. Process Synchronization

- **Mutual Exclusion**
  - Race conditions, need for synchronization
- **Critical Region**
  - Conditions and structure
- **Lock and Unlock Mechanisms**
  - Semaphores, mutex, monitors

---

5. Scheduling

- **CPU Scheduling**
  - Algorithms: FCFS, SJF, Round Robin, Priority
  - Gantt chart, waiting time, turnaround time
- **I/O Scheduling**
  - Disk scheduling: FCFS, SSTF, SCAN, LOOK
- **Resource Scheduling**
  - Resource Allocation Graphs, Banker's Algorithm

---

6. Deadlock

- Conditions for deadlock
- Deadlock Prevention, Avoidance, Detection
- Recovery from deadlock

---

7. File and I/O Management (Implied in I/O scheduling)

---

8. OS Architectures

- **Unix System**
  - Basic structure, commands, file system, process handling
- **Windows System**
  - Architecture overview, memory and process management, file system

Study material

1. System Concepts – Basics of Operating Systems

✅ What is an Operating System (OS)?

An **Operating System** is **system software** that manages all the hardware and software resources on a computer. It acts as a **bridge** between the **user** and the **computer hardware**.

➤ *Key Functions of an OS:*

- **Resource management** (CPU, memory, I/O)
- **Process management** (running programs)
- **File system management** (creating, reading, writing files)
- **User interface** (CLI or GUI)

 Technical term: OS is a *system software* that provides an interface between *hardware and user-level applications*.

✅ What is the Kernel?

The **kernel** is the **core part of the operating system**. It directly interacts with the **hardware** and manages resources like CPU, memory, and devices.

➤ *Types of Kernels:*

- **Monolithic Kernel**: All OS services run in one layer (e.g., Linux)
- **Microkernel**: Minimal services in the kernel; others run in user space (e.g., Minix)
- **Hybrid Kernel**: Mix of both (e.g., Windows NT)

 Technical term: Kernel operates in **privileged (kernel) mode** and controls low-level tasks.

✅ What is a Shell?

The **Shell** is a **program that takes user commands** (typed or clicked) and passes them to the **kernel**.

➤ *Types of Shells:*

- **Command-Line Interface (CLI)**: e.g., Bash, PowerShell
- **Graphical User Interface (GUI)**: e.g., Windows desktop, GNOME

 Technical term: Shell is a **user interface** for access to the OS services.

Summary:

Term	Role	Interacts With
OS	Manages system resources	Hardware & user applications
Kernel	Core of the OS	Directly with hardware
Shell	Interface for the user	Sends commands to kernel

☑ Types of Operating Systems

---

1. Batch Operating System

- Jobs are **collected into batches** and processed together.
- **No user interaction** during execution.
- Example: Early IBM systems.

📌 Used when computers were large and expensive; users submitted jobs on punch cards.

---

2. Time-Sharing Operating System

- Allows **multiple users** to share system resources simultaneously.
- CPU time is **divided into time slots** and given to each user in turn.
- Example: UNIX, Multics

 Enables **multi-user and multitasking** systems.

---

3. Real-Time Operating System (RTOS)

- Used for systems that need to **respond immediately** to input.
- Predictable and **deterministic response time**.
- Example: Air traffic control, industrial robots.

 Hard RTOS (strict timing) vs Soft RTOS (some flexibility).

---

4. Distributed Operating System

- **Multiple computers** (nodes) work together, but look like a **single system** to the user.
- Resources and files are distributed.

 Manages **distributed computing environments**.

---

5. Network Operating System (NOS)

- OS designed to **support connected computers** over a network.
- Provides file sharing, printer access, communication tools.
- Example: Novell NetWare, Windows Server

 Focus on **connectivity and shared access**.

---

6. Mobile Operating System

- OS specifically designed for **smartphones and tablets**.
- Optimized for touch, power-saving, wireless, and mobile apps.
- Examples: Android, iOS

 Lightweight, battery-conscious, and app-focused.

✅ Recap Table

Type	Key Feature	Examples
Batch	No interaction, jobs in batches	IBM early systems
Time-Sharing	Multi-user, time slots for each process	UNIX, Multics
Real-Time	Fast, deterministic response	VxWorks, RTLinux
Distributed	Multiple systems work as one	LOCUS, Amoeba
Network OS	Manages devices on a network	Windows Server, NetWare
Mobile OS	For smartphones and tablets	Android, iOS

✅ 2. Multiprogramming, Multiprocessing, and Multitasking

🔹 1. Multiprogramming

➤ *Definition:*

Multiprogramming means **more than one program is kept in memory at the same time**, and the **CPU switches between them** to keep working while some programs are waiting (e.g., for I/O like reading from a disk).

 Technical purpose: Improve **CPU utilization**.

➤ *How it works:*

- Suppose Program A is waiting for a file to load.
- Instead of letting the CPU sit idle, the OS runs Program B.

➤ *Important Concepts:*

- **Single CPU**
- **Multiple programs in RAM**
- **CPU scheduling** needed to decide which program runs next

📄 *Example: Old Unix systems, IBM mainframes*

🔹 2. Multiprocessing

➤ *Definition:*

Multiprocessing means a system has **two or more CPUs** (or cores), and **they can work simultaneously** on different tasks or on the same task in parallel.

 Technical purpose: Increase **computing power and speed**.

➤ *Types:*

- **Symmetric multiprocessing (SMP)**: All CPUs are equal (e.g., modern PCs)
- **Asymmetric multiprocessing (AMP)**: One main CPU, others do assigned tasks

Example: Dual-core, quad-core computers, servers

---

3. Multitasking

Definition:

Multitasking allows the user to **run multiple tasks (programs or processes) at the same time**, with the CPU switching rapidly between them.

 Technically, only **one task runs at a time** on a single-core CPU, but the **switching is so fast** that it feels simultaneous.

Types:

- **Preemptive multitasking**: OS decides when to switch (e.g., Windows, Linux)
- **Cooperative multitasking**: Program itself decides to give up control (e.g., old Mac OS)

Example: Editing a document, listening to music, and downloading files together

---

Differences Summary

Feature	Multiprogramming	Multiprocessing	Multitasking
CPU count	One	Multiple	One or more
Program execution	Several in memory, one runs	Parallel execution	Rapid switching
Goal	Better CPU usage	Speed, performance	Responsiveness
Example	IBM mainframe OS	Quad-core server	Windows or Android

Process States and Transitions

A **process** is a program in execution. It goes through several **states** during its lifetime:

States:

1. **New** – The process is being created.
2. **Ready** – Process is waiting to be assigned to a CPU.
3. **Running** – The process is currently being executed.

4. **Waiting** (or Blocked) – Waiting for some I/O (like disk or input).
5. **Terminated** – The process has finished execution.

🔁 State Transitions:

- **New → Ready**: Loaded into memory.
- **Ready → Running**: Assigned to CPU.
- **Running → Waiting**: Needs I/O, so it waits.
- **Waiting → Ready**: I/O is done.
- **Running → Terminated**: Finishes execution.
- **Running → Ready**: Preempted by OS to give time to another process.

□ Technical terms:

- **PCB (Process Control Block)**: A data structure used by the OS to store info about a process.
- **Scheduler**: The part of OS that selects the next process to run.

🔁 Context Switching

🔷 Definition:

Context switching is the process of **saving the state of a running process** and **loading the state of another**.

□ This allows the CPU to switch from one process to another.

🔷 What gets saved?

- Program counter
- CPU registers
- Memory maps
- Process ID

🔷 Why it's important:

- Enables **multitasking**
- Happens **every time OS switches tasks**

📱 Modern OS Examples

OS	Multiprogramming	Multiprocessing	Multitasking
Windows	✔	✔ (multi-core)	✔
Linux	✔	✔	✔
Android	✔	✔	✔
Mac OS	✔	✔	✔

✅ Memory Management in Operating Systems

🔷 1. Virtual Memory

➤ *Definition:*

Virtual memory is a memory management technique that **gives an illusion of a very large memory** to processes, even if the physical RAM is smaller.

 It allows execution of programs **that may not be completely in memory**.

➤ *Why Virtual Memory?*

- Helps run **large programs**.
- Allows **many processes** to run at once.
- Makes memory management **more efficient and secure**.

➤ *How it works:*

- Only **parts** of a program are loaded into RAM.
- The rest is kept on **disk** (usually in a swap area or paging file).
- When the CPU needs a part not in RAM, it **fetches it from disk**.

🔷 2. Demand Paging

➤ *Definition:*

Demand paging is a type of virtual memory where **pages are loaded into memory only when needed**, not all at once.

 Pages are brought into RAM **only when a page fault occurs**.

➤ *How it works:*

- Process starts running.
- Only a few pages are loaded.
- If a page not in memory is accessed → **Page Fault** occurs.

🔷 3. Page Fault

➤ *Definition:*

A **page fault** occurs when a process tries to access a memory page **not currently in RAM**.

➤ *Types of page faults:*

- **Minor (soft)** page fault: The page is in memory but not mapped correctly.
- **Major (hard)** page fault: The page must be loaded from disk → slow.

➤ *Steps on Page Fault:*

1. OS pauses the program.
2. Finds the page on disk.
3. Loads it into RAM.
4. Updates page table.
5. Resumes the program.

🔷 4. Paging

➤ *Definition:*

Paging is a memory management scheme that **divides memory into fixed-size blocks**:

- **Frames**: Fixed-size blocks in physical memory (RAM)
- **Pages**: Fixed-size blocks in virtual memory (process address space)

 Paging removes the need for **contiguous memory allocation**, which prevents fragmentation.

➤ *How it works:*

- OS keeps a **Page Table** to map virtual pages to physical frames.
- Each time the CPU accesses a memory address, it goes through this mapping.

🔷 5. Page Table

➤ *Definition:*

A **page table** is a data structure used by the OS to **map virtual addresses to physical addresses**.

Each entry includes:

- **Page number** (virtual)
- **Frame number** (physical)
- **Valid/invalid bit**
- **Access permissions** (read/write)

◈ 6. TLB (Translation Lookaside Buffer)

➤ *Definition:*

The **TLB** is a **fast cache** that stores recently used **page table entries**.

 Accessing the full page table in RAM is slow; TLB improves performance by storing frequent translations.

➤ *TLB Operation:*

- CPU checks TLB for a page number.
- If found (**TLB hit**) → fast access.
- If not found (**TLB miss**) → page table in RAM is checked.

◈ 7. Fragmentation

➤ Definition:

Fragmentation occurs when memory is **wasted** due to how it is allocated and freed.

There are **two types**:

◆ a. Internal Fragmentation

- Happens when memory is allocated in **fixed-size blocks**, and the block is **bigger than needed**.
- The **unused space inside a block** is wasted.

 Example: Process needs 18KB, block size is 20KB → 2KB wasted.

◆ b. External Fragmentation

- Happens when there is enough total free memory, but it's **scattered in small non-contiguous chunks**.
- No single block is big enough to allocate to a process.

 Example: Free memory: 5KB + 10KB + 15KB, but the process needs 25KB → allocation fails.

◈ 8. Memory Allocation Strategies

➤ *First Fit:*

- Allocates the **first block** that is big enough.
- Fast but may leave small unused spaces.

➤ *Best Fit:*

- Allocates the **smallest block** that fits the process.
- Reduces waste, but may increase time to find the block.

➤ *Worst Fit:*

- Allocates the **largest block**.
- Leaves large chunks of unused memory, often not efficient.

Quick Summary Table:

Concept	Key Point
Virtual Memory	Illusion of more memory using disk
Demand Paging	Load pages only when needed

Concept	Key Point
Page Fault	Access to a page not in RAM
Paging	Divide memory into pages (virtual) and frames (physical)
Page Table	Maps pages to frames
TLB	Cache for recent page table entries
Internal Fragment.	Wasted space inside fixed-size block
External Fragment.	Scattered free memory not useful as one block
First Fit	First available block
Best Fit	Smallest suitable block
Worst Fit	Largest block