Structure of an Operating System

The **structure of an operating system (OS)** refers to how different components of the OS are **organized and interact** with each other to provide services to users and hardware.

Basic Components of an OS Structure

1. **User**
   - End user or application programs
   - Examples: Browser, editor, compiler
2. **System Calls**
   - Interface between user programs and OS
   - Allow programs to request OS services
3. **Kernel**
   - Core of the operating system
   - Controls hardware and system resources
4. **Hardware**
   - CPU, memory, disk, I/O devices

---

2. Types of Operating System Structures

1. Simple Structure (MS-DOS)

- Very small and simple OS
- No clear separation between components
- Applications can access hardware directly

Advantages

- Fast execution
- Easy to design

Disadvantages

- Poor security
- Difficult to maintain

Example: MS-DOS

---

2. Monolithic Structure

- Entire OS runs as a single large program
- All services run in **kernel mode**

Features

- File system, memory, device drivers all in kernel
- High performance

Advantages

- Fast system calls
- Efficient execution

Disadvantages

- Large kernel
- Hard to debug and maintain

Examples: UNIX, Linux (mostly monolithic)

---

3. Layered Structure

- OS is divided into **layers**
- Each layer uses services of the layer below it

Structure Example

User Programs

↑

System Call Interface

↑

Process Management

↑

Memory Management

↑

Hardware

Advantages

- Easy debugging
- Better security

Disadvantages

- Slower performance due to layers
- Difficult layer design

Example: THE OS, early UNIX versions

---

4. Microkernel Structure

- Only essential services in kernel
- Other services run in **user space**

Kernel Handles

- Process scheduling
- Memory management
- Inter-process communication (IPC)

Advantages

- High reliability
- Easy to extend and maintain

Disadvantages

- Slower due to IPC overhead

Examples: Mach, QNX, Minix

---

5. Modular Structure (Loadable Kernel Modules)

- Kernel divided into **modules**
- Modules can be loaded or removed at runtime

Advantages

- Flexible
- Efficient like monolithic kernel

Disadvantages

- Complex design

Examples: Linux, Solaris

---

6. Hybrid Structure

- Combination of monolithic and microkernel
- Performance of monolithic + safety of microkernel

Advantages

- Fast and reliable
- Flexible design

Disadvantages

- Complex architecture

Examples: Windows NT, macOS

Types of Operating Systems

1. Batch Operating System

- Jobs are collected and executed in **batches**
- No user interaction during execution

Advantages

- High CPU utilization

Disadvantages

- Long waiting time
- No priority handling

Example: Early IBM mainframe systems

---

2. Multiprogramming Operating System

- Multiple programs reside in memory at the same time
- CPU switches between jobs to increase utilization

Advantages

- Better CPU usage
- Reduced idle time

Example: UNIX

---

3. Time-Sharing Operating System

- CPU time is divided into small **time slices**
- Multiple users interact with the system simultaneously

Advantages

- Fast response time
- Supports multiple users

Example: Linux, UNIX

---

4. Multiprocessing Operating System

- Uses **more than one CPU**
- Processes execute in parallel

Advantages

- High performance
- Reliability

---

5. Distributed Operating System

- Multiple computers connected by a network
- Appears as a single system to users

Advantages

- Resource sharing
- Load balancing

---

6. Network Operating System

- Provides network services like file sharing and communication
- Each system has its own OS

Advantages

- Centralized management

---

7. Real-Time Operating System (RTOS)

- Produces output within a **fixed time limit**
- Used in critical systems

Types of RTOS:

- **Hard RTOS:** Deadline must be met (miss = failure)
- **Soft RTOS:** Deadline is important but not strict

---

9. Mobile Operating System

- Designed for smartphones and tablets

Examples: Android, iOS

---

1. What is a Process?

A **process** is a **program in execution**.
When a program runs, it becomes a process and goes through different **states** during its execution.

---

2. Process Life Cycle

The **process life cycle** describes the **different states** a process goes through from creation to termination.

Main States of a Process

1. **New**
2. **Ready**
3. **Running**
4. **Waiting / Blocked**
5. **Terminated**

---

3. Explanation of Each Process State

1. New

- Process is being **created**
- Memory is allocated
- Process Control Block (PCB) is initialized

---

2. Ready

- Process is **loaded into main memory**
- Waiting for **CPU allocation**
- Ready to run but CPU is busy

---

3. Running

- Process is currently **executing on the CPU**
- Instructions are being processed

---

4. Waiting / Blocked

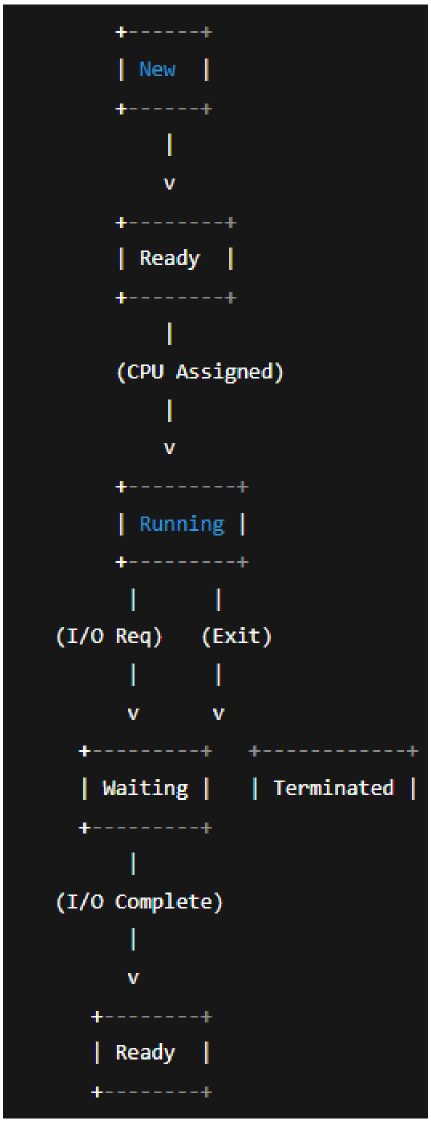
- Process is waiting for **I/O operation** or some event
- CPU is not needed at this time

5. Terminated

- Process has finished execution
- Resources are released

4. Process State Diagram

Basic Process State Diagram



```
     +------+
     | New  |
     +------+
         |
         v
     +--------+
     | Ready  |
     +--------+
         |
     (CPU Assigned)
         |
         v
     +---------+
     | Running |
     +---------+
      |      |
(I/O Req)   (Exit)
      |      |
      v      v
  +---------+   +------------+
  | Waiting |   | Terminated |
  +---------+
      |
(I/O Complete)
      |
      v
   +--------+
   | Ready  |
   +--------+
```

5. State Transitions Explained

Transition	Reason
New → Ready	Process admitted to ready queue

Transition	Reason
Ready → Running	CPU scheduler selects process
Running → Waiting	Process requests I/O
Waiting → Ready	I/O completed
Running → Terminated	Process execution finished
Running → Ready	CPU preemption (time slice expired)

6. Process Control Block (PCB)

Each process is represented by a **PCB**, which contains:

- Process ID (PID)
- Process state
- Program counter
- CPU registers
- Memory limits
- Scheduling information
- I/O status

7. Extended Process States (Optional – for Exams)

Some operating systems include additional states:

- **Suspended Ready**
- **Suspended Waiting**

Used when process is swapped out of main memory.

8. Difference: Program vs Process

Program	Process
Static	Dynamic
Stored on disk	Stored in memory
No execution	Under execution