MODULE 2

Operating System Structures

Comprehensive Study Material

Course Code: DCSE250042
B.Sc. Computer Science | Semester 3

Section 2.1: Operating System Services

2.1.1 Overview

The Operating System provides services to both users (via applications) and the system itself. These are broadly classified into User-Oriented and System-Oriented services.

2.1.2 User-Oriented Services

Service 1: User Interface (UI)

UI Type	Description	Example
Command-Line Interface (CLI)	Text-based commands	bash, cmd, PowerShell
Graphical User Interface (GUI)	Windows, icons, mouse interaction	Windows, macOS, GNOME
Touch Interface	Gesture-based input	Android, iOS
Batch Interface	Script files executed sequentially	Batch files, shell scripts

Service 2: Program Execution

When a user runs a program, the OS performs the following steps:

- Load program from disk into memory
- Allocate memory for the program
- Create a process and its Process Control Block (PCB)
- Set up initial stack and heap
- Load dynamic libraries (DLLs / shared objects)
- Transfer control to the program's entry point
- Handle program termination and cleanup

Service 3: I/O Operations

The OS abstracts all hardware details. When a program calls printf(), the OS:

- Validates the file descriptor
- Checks permissions
- Buffers the output
- Calls the appropriate device driver
- Displays output on screen

The user program never needs to know which graphics card is installed, the screen resolution, or the display protocol.

Service 4: File System Manipulation

Operation	Description	System Call
Create file	Make a new file	creat()
Delete file	Remove a file	unlink()
Open file	Prepare for access	open()

Operation	Description	System Call
Read file	Get data from file	read()
Write file	Put data to file	write()
Close file	Release file	close()
Rename	Change filename	rename()

Service 5: Communication

Two main approaches exist for inter-process communication:

- Shared Memory: Processes access a common memory region — same computer, very fast
- Message Passing: Messages sent via shared memory or network sockets — works across different computers

Service 6: Error Detection

Hardware Errors:

- Memory errors (parity, ECC)
- Disk read/write failure
- Power failure or device not responding

Software Errors:

- Division by zero
- Segmentation fault (invalid memory access)
- Infinite loop, stack overflow, resource deadlock

OS Actions: Log error, terminate offending process, attempt recovery, notify user/admin.

2.1.3 System-Oriented Services

Service	Description	Example
Resource Allocation	Assign CPU, memory, devices to processes	CPU scheduling, memory paging
Accounting	Track resource usage per user	Billing, quotas, statistics
Protection & Security	Control access to resources	User authentication, file permissions

Section 2.2: System Calls

2.2.1 Definition

A system call is the programming interface through which a user program requests services from the operating system's kernel. It acts as a controlled gateway from user mode to kernel mode.

2.2.2 System Call Mechanism

The flow of a system call (e.g., printf → write):

- User program calls a library function (e.g., printf)
- Library prepares system call number and loads arguments into registers
- Executes TRAP (software interrupt) to switch to kernel mode
- Kernel verifies parameters, executes kernel code, returns result
- Control returns to user space

2.2.3 Categories of System Calls

Category 1: Process Control

System Call	Description	Example Use
fork()	Create new process	Web server spawning child
exec()	Replace process image	Running a new program
exit()	Terminate process	Program completion
wait()	Wait for child process	Parent waiting for child
getpid()	Get process ID	Identify current process
kill()	Send signal to process	Terminate another process

Category 2: File Management

System Call	Description	Example
open()	Open a file	fd = open("file.txt", O_RDONLY)
read()	Read from file	read(fd, buffer, 100)
write()	Write to file	write(fd, "Hello", 5)
close()	Close a file	close(fd)
lseek()	Move file pointer	lseek(fd, 0, SEEK_SET)
unlink()	Delete a file	unlink("file.txt")

Category 3: Device Management

System Call	Description	Example
ioctl()	Device control	Set terminal attributes
read()	Read from device	Read from keyboard

System Call	Description	Example
write()	Write to device	Write to display

Category 4: Information Maintenance

System Call	Description	Example
getpid()	Get process ID	Identify current process
gettimeofday()	Get system time	Measure performance
alarm()	Set a timer	Schedule signal delivery
sleep()	Suspend process	Delay execution

Category 5: Communication

System Call	Description	Example
pipe()	Create a pipe	Interprocess communication
shmget()	Get shared memory	Shared memory segment
msgget()	Get message queue	Message passing
socket()	Create network socket	Network communication

2.2.4 System Call Parameter Passing

Method	Description	Notes
Registers	Parameters placed in CPU registers	Fast but limited count
Block (Linux/UNIX)	Parameters stored in memory block; register points to it	Can pass many parameters
Stack	Parameters pushed onto program stack	Common in some architectures

2.2.5 System Call vs Library Call vs Shell Command

Type	Example	Description
System Call	read()	Direct kernel request – fast, low-level
Library Call	fread(), printf()	Wrapper around system calls – buffered
Shell Command	ls, cat	User-level program that makes system calls

Section 2.3: System Programs

2.3.1 Definition

System programs (also called system utilities) provide a convenient environment for program development and execution. They are NOT part of the kernel but use system calls to perform their functions.

2.3.2 Categories of System Programs

File Management

Program	Function	Example
ls	List directory contents	ls -la
cp	Copy files	cp source dest
mv	Move/rename files	mv old new
rm	Delete files	rm file.txt
chmod	Change permissions	chmod 755 script.sh

Status Information

Program	Function	Example
ps	List processes	ps aux
top	Dynamic process view	top
df	Disk free space	df -h
du	Disk usage	du -sh *
date	Show date/time	date
who	List logged-in users	who

Programming Support

Program	Function	Example
gcc	C compiler	gcc program.c -o program
gdb	Debugger	gdb program
make	Build automation	make
ld	Linker	ld object.o -o program

2.3.3 System Programs vs System Calls

Aspect	System Programs	System Calls
Privilege level	User mode	Switches to kernel mode
Purpose	Perform specific tasks	Request OS services

Aspect	System Programs	System Calls
Who uses	End users	Programs (including system programs)
Examples	ls, cp, gcc	open(), read(), fork()
Location	Separate programs on disk	Part of kernel interface

Section 2.4: Operating System Architectures

2.4.1 Monolithic Architecture

Structure: The entire OS runs as a single program in kernel mode. All OS components (process manager, memory manager, file system, I/O, network, device drivers) are compiled into one large kernel binary.

Monolithic Advantages

Advantage	Explanation
Performance	Direct function calls between components; no IPC overhead
Simplicity	Everything is in one place
Efficiency	No message passing between kernel components

Monolithic Disadvantages

Disadvantage	Explanation
Reliability	Any bug can crash the entire system
Maintainability	Hard to modify due to large, tightly coupled codebase
Portability	Tight coupling with specific hardware
Extensibility	Adding features requires recompiling the entire kernel
Security	All code runs in privileged mode

Monolithic Examples

OS	Type
Linux	Hybrid monolithic (with loadable modules)
UNIX (original)	Monolithic
MS-DOS	Monolithic
BSD	Monolithic
Windows NT	Mostly monolithic with modular elements

2.4.2 Layered Architecture

Structure: The OS is divided into hierarchical layers. Layer N can only use the services of layers 0 through N-1. This creates clear boundaries and modularity.

Classic layers ("THE" Operating System by Dijkstra, 1968):

- Layer 5: User programs / applications
- Layer 4: I/O management
- Layer 3: Process communication (IPC, pipes)
- Layer 2: Memory management (paging, segmentation)
- Layer 1: CPU scheduling (process scheduling, context switching)

- Layer 0: Hardware (CPU, memory, devices)

Layered Architecture Comparison

Aspect	Monolithic	Layered
Structure	Single large program	Hierarchical layers
Performance	Faster (direct calls)	Slower (layer overhead)
Modularity	Low	High
Reliability	Low (any bug crashes system)	Better (isolation)
Extensibility	Hard	Easier
Debugging	Difficult	Easier (layer by layer)
Real-world use	Linux, UNIX	THE, OS/2

Section 2.5: Microkernel Architecture

2.5.1 Definition

The Microkernel architecture moves most OS services from kernel space to user space. The kernel contains only the absolute minimum required to function. All other services run as user-space servers.

2.5.2 What Is (and Is Not) Inside the Microkernel

Inside the Microkernel

Component	Why It Must Be in the Kernel
Inter-Process Communication (IPC)	Essential for servers to communicate with each other
Process/Thread Scheduling	Must control CPU directly
Basic Memory Mapping	Required for maintaining address spaces
Interrupt Handling	Requires direct hardware access

Outside the Microkernel (User-Mode Servers)

Server	Function
File Server	File system management
Memory Server	Advanced memory allocation
Device Drivers	Hardware control
Network Stack	TCP/IP, networking
GUI Server	Windowing system
Security Server	Authentication and permissions

2.5.3 Microkernel Advantages & Disadvantages

Advantages	Disadvantages
Reliability: server crash doesn't crash kernel	Performance: message passing overhead
Extensibility: add servers without modifying kernel	IPC cost: many context switches per operation
Portability: small kernel is easier to port	Complexity: inter-server communication is harder
Security: most code runs in user mode	Maturity: less mature than monolithic systems
Maintainability: smaller kernel is easier to debug	

2.5.4 Microkernel Examples

OS	Type	Notes
QNX	Pure microkernel	Used in cars (infotainment systems)
MINIX	Pure microkernel	Teaching OS by Tanenbaum
Mach	Microkernel	Basis of Mac OS X kernel (XNU)

OS	Type	Notes
L4	Microkernel	High-performance microkernel
Windows NT	Hybrid	Mix of monolithic and microkernel elements

2.5.5 Performance Comparison: Monolithic vs Microkernel

Reading a file on a Monolithic OS:

- Application issues system call → Kernel (file + disk driver) → Returns to application
- Result: ~2 context switches — fast!

Reading a file on a Microkernel OS:

- Application → Microkernel → File Server → Microkernel → Disk Driver → (reverse path)
- Result: 6+ context switches — slower, but more reliable!

Section 2.6: Client-Server Architecture

2.6.1 Concept

The Client-Server architecture is a natural extension of the microkernel model. Services are provided by dedicated server processes that clients access via IPC (message passing).

Key characteristics:

- Clients request services from servers
- Servers provide services and respond to client requests
- Communication is via message passing (IPC)
- Servers can be on the same machine or on different machines over a network

2.6.2 Distributed Client-Server

In a distributed system, clients do not need to know (or care) where servers are located. A client machine can request services from a file server, database server, or web server all running on different physical machines on the network. This is the foundation of modern cloud architecture.

Section 2.7: Virtual Machines

2.7.1 Concept

A Virtual Machine (VM) is software that emulates a complete computer system, allowing multiple operating systems to run simultaneously on one physical machine. The layer managing VMs is called the Hypervisor (or VMM – Virtual Machine Monitor).

2.7.2 Types of Hypervisors

Feature	Type 1 Hypervisor (Bare Metal)	Type 2 Hypervisor (Hosted)
Runs on	Directly on hardware	On top of a host OS
Performance	Better (no host OS layer)	Slightly lower
Use case	Production servers, data centers	Developer workstations
Examples	VMware ESXi, Xen, KVM	VirtualBox, VMware Workstation

2.7.3 Virtual Machine Benefits

Benefit	Explanation
Isolation	VMs are completely isolated from each other
Consolidation	Run many VMs on one physical server
Snapshots	Save and restore the complete VM state
Live Migration	Move running VMs between physical servers
Testing	Test different OSes without rebooting
Cloud Computing	Foundation of AWS, Azure, GCP

Section 2.8: System Boot Process

2.8.1 Boot Process Steps

1. **Power On → CPU jumps to BIOS/UEFI address (reset vector)**
- Executes Power-On Self Test (POST)
- Initializes and tests hardware components
2. **BIOS/UEFI loads Bootloader from disk (MBR/GPT)**
- Reads first sector of boot device
- Loads bootloader into memory and transfers control
3. **Bootloader runs (GRUB, LILO, Windows Boot Manager)**
- Presents OS selection menu (if multiple OSes installed)
- Loads kernel into memory along with initial RAM disk (initrd/initramfs)
4. **Kernel Initializes**
- Sets up memory management
- Detects hardware and loads device drivers
- Mounts the root filesystem
- Creates the first process (init / systemd)
5. **Init / systemd starts system services**
- Spawns login prompt or graphical desktop

2.8.2 Key Boot Components

Component	Description
BIOS	Basic Input/Output System – traditional firmware
UEFI	Unified Extensible Firmware Interface – modern replacement for BIOS
MBR	Master Boot Record – traditional partition table (first 512 bytes of disk)
GPT	GUID Partition Table – modern partition scheme
Bootloader	Program that loads the OS kernel (GRUB, systemd-boot)
Kernel	Core of the operating system
initrd/initramfs	Initial RAM disk – temporary root filesystem used during boot
init / systemd	First user-space process (PID 1); starts all other services

Section 2.9: Comparison of Architectures

Feature	Monolithic	Layered	Microkernel	Virtual Machine
Kernel size	Large	Medium	Tiny	N/A (hypervisor)
Performance	Best	Medium	Lower	Lower (overhead)
Reliability	Lower	Medium	High	High
Security	Lower	Medium	High	Very High
Modularity	Low	Medium	High	High
Portability	Hard	Medium	Easy	Easy
Debugging	Hard	Medium	Easier	Easier
Context switches	Few	More	Many	Many
Examples	Linux, UNIX	THE, OS/2	QNX, MINIX	VMware, VirtualBox

2.9.2 Decision Guide

Choose	When
Monolithic	Maximum performance is critical; hardware is known; embedded or specialized systems
Microkernel	Reliability is paramount; medical devices, automotive, critical infrastructure
Virtual Machine	Running multiple OSes; cloud servers; isolation for testing

Section 2.10: Quick Revision Notes

OS Services Summary

User-Oriented Services:

- User Interface (CLI, GUI, Touch, Batch)
- Program Execution (load, run, terminate)
- I/O Operations (read, write, control devices)
- File System (create, delete, read, write, rename)
- Communication (IPC: shared memory / message passing)
- Error Detection (hardware and software errors)

System-Oriented Services:

- Resource Allocation (CPU, memory, I/O devices)
- Accounting (track usage for billing and quotas)
- Protection & Security (access control, authentication)

System Call Categories

Category	Example Calls
Process Control	fork(), exec(), wait(), exit()
File Management	open(), read(), write(), close()
Device Management	ioctl(), read(), write()
Information Maintenance	getpid(), gettimeofday()
Communication	pipe(), shmget(), socket()

Key Points to Remember

- System calls are the ONLY way user programs can access kernel services.
- Monolithic kernels are faster but less reliable; microkernels are more reliable but slower.
- A microkernel contains only: IPC, scheduler, basic memory mapping, interrupt handling.
- System programs USE system calls; they are NOT part of the kernel.
- Virtual machines provide hardware isolation; they are the foundation of cloud computing.
- Shell commands (like ls) are system programs, not system calls.
- Context switches between user/kernel mode incur overhead.

Section 2.11: Practice Questions

Multiple Choice Questions (with Answers)

#	Question	Answer
1	Which of the following is NOT an OS service? a) Program execution b) I/O operations c) Database management d) File system manipulation	c
2	The interface between a user program and the OS kernel is called: a) Library call b) System call c) Shell command d) API	b
3	Which OS architecture has the best performance? a) Monolithic b) Layered c) Microkernel d) Virtual machine	a
4	Which component is typically found in a microkernel? a) File server b) Device driver c) IPC d) Network stack	c
5	The first program run by the Linux kernel is: a) BIOS b) GRUB c) init/systemd d) shell	c
6	Which type of hypervisor runs directly on hardware? a) Type 1 b) Type 2 c) Type 3 d) All of the above	a
7	Which system call creates a new process? a) exec() b) wait() c) fork() d) exit()	c
8	The bootloader is stored in: a) RAM b) First sector of disk (MBR) c) CPU cache d) CMOS	b

Short Answer Questions (2–3 marks)

- List any five OS services with a brief description of each.
- What is a system call? Give three examples.
- Differentiate between system calls and system programs.
- What is a monolithic kernel? Name one OS that uses it.
- What is a microkernel? What components does it typically contain?
- What is a hypervisor? Differentiate Type 1 and Type 2.
- List the steps in the system boot process.
- Differentiate between read() (system call) and fread() (library call).

Medium Answer Questions (5–7 marks)

- Explain the five categories of system calls with examples.
- Draw and explain the monolithic kernel architecture. List its advantages and disadvantages.
- Draw and explain the microkernel architecture. Compare with monolithic.
- Explain the client-server model in operating systems.
- Describe the layered OS architecture with a diagram.

Long Answer Questions (10–15 marks)

- Explain in detail the various OS services (both user-oriented and system-oriented).
- Draw and explain the complete system call flow from a user program (e.g., printf()) to the kernel and back.

- Compare and contrast monolithic, layered, and microkernel architectures with examples.
- Explain the virtual machine architecture and how it enables running multiple OSes on one machine.
- Describe the complete system boot process from power-on to login prompt.

Summary: Key Takeaways

Topic	Key Takeaway
OS Services	User-oriented (UI, exec, I/O, files, comm, errors) + System-oriented (resource, accounting, security)
System Call	Request from user program to kernel; causes TRAP to switch to kernel mode
System Program	User-level utility (ls, cp, gcc) that uses system calls
Monolithic	All OS in one kernel; fast but unsafe; used by Linux, UNIX
Microkernel	Minimal kernel; safe but slower; used by QNX, MINIX
Type 1 Hypervisor	Runs directly on hardware — VMware ESXi, KVM, Xen
Type 2 Hypervisor	Runs on host OS — VirtualBox, VMware Workstation
Boot Process	BIOS/UEFI → Bootloader → Kernel → init/systemd → Login

End of Module 2 Study Material | For reference: Silberschatz, Galvin & Gagne – Operating System Concepts (Chapters 2–3)