مقدمة الملف

أهلًا فيكم !

أتمنى أن يكون هذا الملف مساعدًا لك في فهم الدروس بسهولة ويسر.

أسأل الله أن يوفقك في كل خطوة، وييسر لك الصعوبات، ويجعل العلم نورًا يهديك في حياتك

@Nubgha

CS222 — Operating Systems

Chapter 1: Introduction to Operating Systems

Term	Definition
Operating System	Intermediary between user and computer hardware
Kernel	The one program always running on the computer
System Program	Ships with OS but not part of the kernel
Application Program	All programs not associated with OS
Middleware	Software frameworks providing extra services (databases, graphics)
Bootstrap	Simple code that initializes system and loads kernel
Interrupt	Signal that transfers control to OS interrupt service routine
Trap / Exception	Software-generated interrupt (error or system call)
System Call	Request to OS to access its services
Process	A program in execution (active); program alone is passive
Thread	Unit of execution within a process
DMA	Transfers data to main memory without CPU intervention
Cache	Faster storage holding copies of data from slower storage
Dual-Mode	User mode (bit=1) vs Kernel mode (bit=0)
Multiprogramming	CPU always has a job; switches when one waits
Multitasking	CPU switches so fast users interact with each (<1 sec)
Distributed System	Networked heterogeneous systems acting as one
Clustered System	Multiple systems sharing storage (SAN), high availability
VMM	Manages virtual machines; runs OSes within OSes
Cloud Computing	Computing/storage/apps delivered as a service over network

CS222 — Operating Systems

Chapter 1: Introduction to Operating Systems

1. Computer System Components

1. Hardware
1. Hardware
2. Operating System2. Operating System3. Application Programs
4. Users

2. Operating System Goals

1. Execute user programs and make solving problems easier
2. Make the computer system convenient to use
3. Use the computer hardware efficiently

3. Storage Hierarchy (Fastest → Slowest)

1. Registers
2. Cache
3. Main Memory (DRAM)
4. SSD / NVM
5. HDD
6. Optical Disk
7. Magnetic Tape

4. OS Management Responsibilities

Responsibility	Description
Process Management	Create, delete, synchronize, deadlock handling
Memory Management	Track, allocate, deallocate memory
File-System Management	Files, directories, backup
Mass-Storage Management	Mounting, scheduling, partitioning, protection
Caching	Data movement between storage levels
I/O Subsystem	Buffering, spooling, hides hardware complexity
Protection & Security	Access control, user/group IDs, defense against attacks

5. I/O Methods

	Synchronous	Asynchronous
Returns	Only upon I/O completion	Immediately, without waiting
CPU	Idles until next interrupt	Continues working
Requests	One at a time	Multiple requests possible

6. Cloud Computing Types

Type	Description
Public Cloud	Available via Internet to anyone willing to pay
Private Cloud	Run by a company for its own use
Hybrid Cloud	Includes both public and private components
SaaS	One or more applications available via Internet
PaaS	Software stack ready for application use via Internet
IaaS	Servers or storage available over Internet

7. Computing Environments

1. Traditional
2. Client-Server
3. Peer-to-Peer (P2P)
4. Virtualized
5. Distributed
6. Clustered
7. Mobile
8. Cloud
9. Real-Time Embedded

. Kernel Data Structures

Data Structure	Notes
Singly Linked List	—
Doubly Linked List	—
Circular Linked List	—
Binary Search Tree	Search O(n); Balanced BST: O(lg n)
Hash Map	Created via hash function
Bitmap	String of n binary digits representing status of n items

9. Interrupt Handling Steps

1. Save CPU state (registers + program counter)
2. Determine which type of interrupt occurred
3. Execute the appropriate code segment for that interrupt type

10. Clustered System Types

Type	Description
Asymmetric	One machine on hot-standby mode
Symmetric	Multiple nodes monitoring each other, running applications
HPC	High-Performance Computing; requires parallelization

CS222 — Operating Systems

Chapter 2: OS Structures

Term	Definition
User Interface (UI)	Environment allowing users to interact with OS: CLI, GUI, or Batch
CLI (Command Line Interface)	Users enter commands directly; implemented in kernel or shell; main function: fetch command and execute it
Shell	CLI program that fetches commands from user and executes them; multiple versions (e.g., Bash)
GUI (Graphical User Interface)	Uses mouse, keyboard, icons; icons represent files/programs/actions; invented at Xerox PARC
System Call	Programming interface to OS services; typically written in C/C++; usually accessed via API
API	Application Programming Interface; programs use API instead of direct system calls for portability and stability
System Program	Programs providing convenient environment for development and execution; most users interact via these
Policy	What to do — the decision
Mechanism	How to do it — the implementation
Microkernel	Moves most OS functionality from kernel to user space; modules communicate via message passing
Modules	Object-oriented OS components; communicate via known interfaces; loadable as needed; similar to layered but more flexible
Virtual Machine (VM)	Extends layered approach fully; treats hardware and OS kernel as hardware; provides interface identical to underlying hardware
VMM	Virtual Machine Manager; creates illusion each process has its own processor and virtual memory
OVMF	Open Virtual Machine Format; standard allowing a VM to run on multiple host platforms
Bootstrap Loader	Small code that locates and loads the kernel into memory, then starts it
SYSGEN	Program to obtain hardware configuration info for OS generation
Core Dump	Captures memory of a failed application
Crash Dump	Captures kernel memory on OS failure
DTrace	Live monitoring tool; captures state data when code executes (Solaris, FreeBSD, Mac OS X)

1. OS Services — Helpful to the User

Service	Description
User Interface (UI)	CLI, GUI, or Batch interaction with the system
Program Execution	Load program into memory, run it, terminate normally or on error
I/O Operations	Handle input/output involving files or I/O devices
File-System Manipulation	Create, delete, read, write, search files and directories
Communications	Exchange info within same computer or across networks (shared memory or message passing)
Error Detection	Detect errors in CPU, memory, I/O devices, or user programs; take appropriate actions

2. OS Services — Efficient System Operation

Service	Description
Resource Allocation	Allocates CPU, memory, file storage, I/O devices among multiple users/processes running simultaneously
Accounting	Tracks how much and what types of resources each user uses
Protection & Security	Controls access to resources; protects against unauthorized access via user authentication and access validation

3. User Interface Types

Interface	Characteristics	Examples
CLI	Text commands; uses shells; commands built-in or program names; new features added without modifying shell	Bash, PowerShell, Bourne Shell
GUI	Mouse + keyboard + icons; folder navigation; mouse clicks perform actions (display info, show options, execute functions)	Windows, Mac OS X (Aqua)
Both	Many modern OSes include both; Microsoft Windows mainly GUI but has CMD shell; Mac OS X has Aqua GUI + UNIX shells; Solaris mainly CLI + optional GUI	Windows, Mac OS X, Solaris

4. Most Common APIs

API	Platform
Win32 API	Windows
POSIX API	POSIX-based systems: UNIX, Linux, macOS
Java API	Java Virtual Machine (JVM)

5. Why Use APIs Instead of Direct System Calls?

#	Reason
1	System calls differ between platforms; APIs provide a stable portable interface
2	When OS updates system calls, APIs are updated to support these improvements
3	APIs often provide more useful functionality than direct system calls
4	APIs can support multiple OS versions and choose the appropriate one during execution

6. Types of System Calls

Category	Examples
Process Control	end, abort, load, execute, create/terminate process, get/set attributes, wait, allocate/free memory
File Management	create/delete file, open, close, read, write, reposition, get/set file attributes
Device Management	request/release device, read, write, reposition, get/set attributes, attach/detach devices
Information Maintenance	get/set time or date, get/set system data, get/set process/file/device attributes
Communications	create/delete connection, send/receive messages, transfer status, attach/detach remote devices

7. System Call Parameter Passing Methods

Method	Description	Note
Registers	Parameters passed directly in CPU registers	Limited by number of available registers
Memory Block / Table	Parameters stored in memory block; block address passed in register	Used by Linux and Solaris
Stack	Parameters pushed onto program stack; OS pops them off at execution	No limit on number or length of parameters

8. System Programs Categories

Category	Description
File Manipulation	Create, delete, copy, rename, print, dump, list files and directories
Status Information	Show system info: date, time, memory, disk space, users, performance; may use registry
File Modification	Text editors, search and transform file contents
Programming Language Support	Compilers, assemblers, debuggers, interpreters
Program Loading & Execution	Loaders, linkage editors, overlay loaders, debugging systems
Communications	Create connections between processes/users/systems; send messages, email, file transfer
Application Programs	Interface for user-level applications

9. OS Design Goals

Goal Type	Properties
User Goals	Convenient, easy to learn, reliable, safe, fast
System Goals	Easy to design, implement, maintain; flexible, reliable, error-free, efficient
Key Principle	Separate policy (what to do) from mechanism (how to do it) — allows changing policies without changing mechanisms

10. OS Structure Approaches

Approach	Description	Example
Simple Structure	Maximum functionality in minimal space; not modular; interfaces and functionality levels not well separated	MS-DOS
Layered	OS divided into layers; Layer 0 = hardware, Layer N = UI; each layer uses only functions of lower layers; modular and independent	THE OS
Microkernel	Moves functionality from kernel to user space; modules communicate via message passing; easier to extend/port, more reliable/secure; disadvantage: performance overhead	Mach, QNX
Modules	Object-oriented; components communicate via known interfaces; loadable as needed; similar to layered but more flexible	Solaris, Linux
Hybrid	Combines multiple approaches for performance, security, and usability	Mac OS X, iOS, Android

11. Virtual Machine Benefits

#	Benefit
1	First commercial use: IBM mainframes, 1972
2	Multiple execution environments (different OSes) can share the same hardware
3	Isolation: VMs are protected from each other
4	Controlled file sharing between VMs is possible
5	VMs can communicate with each other and physical systems via networking
6	Useful for development and testing
7	Consolidation: combine many low-resource systems into fewer, busier machines
8	OVMF: standard format allowing a VM to run on multiple host platforms

12. OS Debugging

Tool / Concept	Description
Log Files	OS generates logs with error information
Core Dump	Captures memory of a failed application
Crash Dump	Captures kernel memory on OS failure
Performance Tuning	Optimizes system performance beyond fixing crashes
Kernighan's Law	Debugging is twice as hard as writing the code
DTrace	Live monitoring; captures state data when code executes (Solaris, FreeBSD, Mac OS X)

13. OS Generation & Booting

Term	Description
SYSGEN	Program to obtain hardware configuration info for OS generation
Booting	Starting a computer by loading the kernel
Bootstrap Program	Stored in ROM; locates and loads the kernel into memory, starts execution
Bootstrap Loader	Small code to locate kernel, load it, and start it
Boot Block	Sometimes a two-step process: boot block loads bootstrap loader
Firmware	Execution starts at fixed memory location using firmware to hold initial boot code

Term	Definition
Process	A program in execution (active entity); a program stored on disk is passive
Text Section	The program code (also called code section)
Stack	Temporary data: function parameters, return addresses, local variables; grows downward
Heap	Memory dynamically allocated during run time; grows upward
Data Section	Contains global variables
PCB (Process Control Block)	Data structure storing all info associated with a process; also called Task Control Block
Process Scheduler	Selects among available processes for next execution on CPU
Job Queue	Set of all processes in the system
Ready Queue	Set of all processes in main memory, ready and waiting to execute
Device Queue	Set of processes waiting for a specific I/O device
Context Switch	Saving state of old process and loading saved state of new process when CPU switches between processes
PID	Process Identifier; unique number used to identify and manage a process
fork()	UNIX system call that creates a new child process (duplicate of parent); found in UNIX, Linux, BSD, macOS
exec()	System call used after fork() to replace process memory space with a new program
exit()	Process executes last statement; requests OS to delete it; OS deallocates resources; child sends output to parent via wait()
abort()	Parent may terminate child process if: child exceeded resources, task no longer needed, or parent is exiting
wait()	Parent waits for child termination; returns child PID
Zombie Process	Terminated process whose parent has not called wait()
Orphan Process	Child process left without a parent (parent terminated first)
IPC	Interprocess Communication; mechanism for cooperating processes to exchange data
Shared Memory	IPC model: processes share a region of memory; faster once established
Message Passing	IPC model: processes communicate without shared variables; good for distributed systems
Direct Communication	Processes name each other explicitly in send(P, msg) / receive(Q, msg)
Indirect Communication	Messages sent via shared mailboxes with unique IDs; multiple processes can share a mailbox
Swapping	Removing a process from memory to disk and bringing it back to continue execution
LPC	Local Procedure Call; Windows IPC mechanism; works only between processes on the same system; uses ports like mailboxes

CS222 — Operating Systems

Chapter 3: Processes

1. Process Memory Layout (bottom → top)

Section	Location	Contents
Text	Bottom	Program code
Data	Above text	Global variables
Heap	Middle (grows ↑)	Dynamically allocated memory
Stack	Top (grows ↓)	Local variables, return addresses, function parameters

2. Process States

State	Description
New	The process is being created
Running	Instructions are being executed
Waiting	Process is waiting for some event to occur (e.g., I/O completion)
Ready	Process is waiting to be assigned to a processor (in main memory)
Terminated	Process has finished execution (passive/program)

3. PCB (Process Control Block) Contents

Field	Description
Process State	Running, waiting, ready, new, terminated
Program Counter	Location of next instruction to execute
CPU Registers	Contents of all process-centric registers
CPU Scheduling Info	Priorities, scheduling queue pointers
Memory-Management Info	Memory allocated to the process
Accounting Information	CPU used, clock time elapsed, time limits
I/O Status Information	I/O devices allocated to process; list of open files

4. Scheduling Queues

Queue	Description
Job Queue	All processes in the system
Ready Queue	All processes in main memory, ready to execute
Device Queue	Processes waiting for a specific I/O device; each device has its own queue

5. Types of Schedulers

Scheduler	Function	Frequency
Long-Term (Job Scheduler)	Selects which processes enter the ready queue; controls degree of multiprogramming	Infrequent (seconds/minutes) — can be slow
Short-Term (CPU Scheduler)	Selects which process executes next; allocates CPU	Very frequent (milliseconds) — must be fast; sometimes only scheduler in system
Medium-Term Scheduler	Removes processes from memory (swapping) when multiprogramming degree needs to decrease	As needed

6. Process Types

Type	Description
I/O-Bound Process	Spends more time doing I/O than computations; many short CPU bursts
CPU-Bound Process	Spends more time doing computations; few very long CPU bursts

7. Process Creation Options

Category	Options
Resource Sharing	All shared I Subset shared I None shared
Execution	Parent and children execute concurrently I Parent waits until children terminate
Address Space (UNIX)	Child is duplicate of parent (fork) I Child has a new program loaded into it (exec)

8. Process Termination

Call / Case	Description
exit()	Normal termination; OS deallocates resources; child sends output to parent via wait()
abort()	Parent terminates child if: child exceeded resources I task no longer needed I parent is exiting (cascading)
wait()	Parent waits for child termination; receives child PID and exit status
Zombie	Terminated process whose parent has not called wait() yet
Orphan	Child process left without a parent (parent terminated without calling wait())

9. Google Chrome Multiprocess Architecture

Process Type	Role
Browser Process	Manages user interface, disk and network I/O
Renderer Process	Renders web pages, handles HTML/JavaScript; runs in sandbox (restricted disk/network I/O)
Plug-in Process	One for each type of plug-in

10. IPC Models

Model	Description	Best For
Shared Memory	Processes share a region of memory; producer-consumer pattern; unbounded or bounded buffer	Speed; same machine
Message Passing	Processes communicate without shared variables; send() and receive(); direct or indirect	Distributed systems

11. Reasons for Process Cooperation

#	Reason
1	Information sharing — multiple processes may need the same data
2	Computation speedup — task split into subtasks running in parallel
3	Modularity — system divided into separate processes or threads
4	Convenience — a user may perform multiple tasks simultaneously

12. Direct vs Indirect Communication

Feature	Direct	Indirect
How	Processes name each other explicitly	Messages sent via shared mailboxes (unique IDs)
Link	Auto established; exactly one link per pair; usually bi-directional	Link shared among multiple processes possible; one mailbox can have multiple links
Operations	send(P, msg) / receive(Q, msg)	create mailbox, send/receive messages, destroy mailbox

13. Synchronization in Message Passing

Type	Send	Receive
Blocking (Synchronous)	Sender waits until message is received	Receiver waits until message is available
Non-blocking (Asynchronous)	Sender sends message and continues	Receiver gets message or null if none available

14. Producer-Consumer Buffer Types

Buffer Type	Description
Unbounded-Buffer	No practical limit on buffer size; producer never waits
Bounded-Buffer	Fixed buffer size; producer must wait if buffer is full; consumer waits if buffer is empty

15. Multitasking in Mobile Systems

Platform	Behavior
iOS	Single foreground process (user-controlled via UI); multiple background processes with limits: single short task, receiving event notifications, specific long-running tasks (e.g., audio playback)
Android	Runs foreground and background with fewer limits; background uses a service (no UI, small memory); service can run even if background process suspended; service has no user interface

CS222 — Operating Systems

Chapter 4: Process Synchronization

Term	Definition
Race Condition	Multiple processes access shared data concurrently; outcome depends on execution order
Critical Section	Code segment where a process may change shared variables, tables, or files
Entry Section	Code that requests permission to enter the critical section
Exit Section	Code that follows the critical section
Remainder Section	All remaining code after the exit section
Mutual Exclusion	If Pi is in its critical section, no other process can be in theirs simultaneously
Progress	If no process is in critical section, only non-remainder processes decide who enters next; cannot be postponed indefinitely
Bounded Waiting	A limit exists on how many times other processes enter critical sections after a request is made
Peterson's Solution	Software solution for 2 processes; uses shared int turn and Boolean flag[2]; not guaranteed on modern architectures
Mutex Lock	Simplest sync tool; acquire() to lock, release() to free; uses Boolean variable; requires busy waiting (spinlock)
Spinlock	A lock requiring busy waiting; process loops continuously checking the lock
Semaphore	Integer variable S; accessed only via atomic wait() and signal() operations
Counting Semaphore	Integer ranges over unrestricted domain; used for counting finite resources
Binary Semaphore	Integer ranges 0–1 only; acts like a mutex lock
Busy Waiting	Process continuously loops checking a condition; wastes CPU cycles
Block	Places invoking process into the semaphore's waiting queue
Wakeup	Removes one process from waiting queue and places it in ready queue
Deadlock	Two or more processes wait indefinitely for an event only another waiting process can cause
Starvation	Process may never leave semaphore queue — indefinite blocking
Monitor	High-level sync construct; only one process active inside at a time
Checkpoint	Shortens log and recovery time; outputs log records, modified data, and checkpoint record to stable storage

1. Critical Section Structure

Section	Description
Entry Section	Request permission to enter the critical section
Critical Section	Code that accesses or modifies shared data
Exit Section	Signals the end of the critical section
Remainder Section	All other code

CS222 — Operating Systems

Chapter 4: Process Synchronization

2. Critical Section Solution Requirements

#	Requirement	Description
1	Mutual Exclusion	If Pi in critical section → no other process can be in theirs
2	Progress	Selection of who enters cannot be postponed indefinitely
3	Bounded Waiting	Limit on how many times others enter after a request is made

3. Peterson's Solution

Aspect	Detail
Scope	Two processes only
Shared Variables	int turn I Boolean flag[2]
turn	Indicates whose turn it is to enter the critical section
flag[i]	True = process Pi is ready to enter critical section
Guarantees	Mutual exclusion ✓ I Progress ✓ I Bounded waiting ✓
Disadvantage	Works only for 2 processes; may fail on modern architectures due to hardware/memory ordering optimizations

4. Synchronization Hardware

Method	Description	Limitation
Disable Interrupts (Uniprocessor)	Running code executes without preemption	Too inefficient on multiprocessors; not scalable
Atomic Hardware Instructions	Non-interruptible operations: test_and_set() or compare_and_swap(); can test memory word and set its value	Foundation of modern lock implementations
Locking	Protect critical regions using locks — basis of most solutions	Requires acquire() before and release() after

5. Mutex Lock vs Semaphore

Feature	Mutex Lock	Semaphore
Variable Type	Boolean	Integer
Operations	acquire() / release()	wait() / signal() [originally P() / V()]
Busy Waiting	Yes (spinlock)	Eliminated with block/wakeup
Use Case	Mutual exclusion (binary)	Resource counting or ordering
Complexity	Simple	More powerful, more complex

6. Semaphore Types

Type	Value Range	Use
Counting Semaphore	Unrestricted integer	Control access to finite resources; resource counting
Binary Semaphore	0 and 1 only	Mutual exclusion; acts like a mutex lock

7. Semaphore with No Busy Waiting

Operation	Action
block()	Places the invoking process into the semaphore's waiting queue
wakeup()	Removes one process from waiting queue → places in ready queue
Waiting Queue Entry	Contains: value (integer) I pointer to next record in list

8. Classic Synchronization Problems

Problem	Description	Key Shared Data
Bounded Buffer	Producer fills buffer; consumer empties it; avoid overflow/underflow	Buffer[N], in/out counters, semaphores (empty, full, mutex)
Readers-Writers	Multiple readers simultaneously OK; only one writer at a time; priority variations exist	rw_mutex=1, mutex=1, read_count=0
Dining Philosophers	5 philosophers think and eat; need 2 chopsticks; avoid deadlock and starvation	chopstick[5] semaphore initialized to 1

9. Readers-Writers Variations

Variation	Rule	Risk
First Variation	No reader waits unless a writer has permission to access the shared object	Writer starvation
Second Variation	Once a writer is ready, it writes ASAP	Reader starvation
Solution	Kernel provides reader-writer locks to manage access efficiently	—

10. Windows XP Synchronization

Mechanism	Description
Interrupt Masks	Protect global resources on uniprocessor systems
Spinlocks	Used on multiprocessor systems; spinlocking thread never preempted
Dispatcher Objects	User-land: Mutexes, Semaphores, Events, Timers
Events	Act like condition variables
Timers	Notify one or more threads when time expires
Object States	Signaled (available) or Non-signaled (thread will block)

11. Linux Synchronization

Version / Feature	Detail
Before kernel 2.6	Disables interrupts to implement short critical sections
Version 2.6+	Fully preemptive
Provides	Semaphores I Spinlocks I Reader-writer versions of both
Single-CPU systems	Spinlocks replaced by enabling/disabling kernel preemption

12. Checkpoints & Recovery

Step	Action
1	Output all log records in volatile storage → stable storage
2	Output all modified data from volatile → stable storage
3	Output a log record to stable storage
Recovery	Redo transactions Ti that started before most recent checkpoint; skip those after Ti (already in stable storage)

CS222 — Operating Systems

Chapter 5: CPU Scheduling

Term	Definition
CPU–I/O Burst Cycle	Process execution = cycles of CPU execution followed by I/O wait; CPU burst followed by I/O burst
CPU Burst Distribution	Important factor in scheduling; typically many short bursts and few long ones
CPU Scheduler	Short-term scheduler; selects process from ready queue and allocates CPU to it
Dispatcher	Gives CPU control to the process selected by the short-term scheduler
Dispatch Latency	Time for dispatcher to stop one process and start another
Non-preemptive	Once CPU is assigned, process keeps it until it releases (wait or terminate) — cases 1 & 4
Preemptive	CPU can be taken away from a running process — all other cases
Turnaround Time	Total time to execute a process (submission → completion)
Waiting Time	Time a process spends in the ready queue
Response Time	Time from submitting a request until first response (important for time-sharing)
Throughput	Number of processes completed per time unit
FCFS	First-Come, First-Served: processes served in arrival order; non-preemptive
SJF	Shortest-Job-First: shortest next CPU burst scheduled first; minimizes average waiting time (optimal)
SRTF	Shortest-Remaining-Time-First: preemptive SJF; preempts if new process has shorter remaining time
Priority Scheduling	Each process has priority number; CPU given to highest priority (smallest integer = highest)
Starvation	Low-priority processes may never execute if higher-priority processes keep arriving
Aging	Gradually increase priority of waiting processes over time to prevent starvation
Round Robin (RR)	Each process gets a time quantum q (10–100ms); preemptive; added to end of ready queue after expiry
Time Quantum (q)	Small CPU time unit given to each process in Round Robin scheduling
Multilevel Queue	Ready queue split into separate queues; each with its own scheduling algorithm
Multilevel Feedback Queue	Processes can move between queues; aging can be implemented; most complex but most flexible
CFS	Completely Fair Scheduler (Linux v2.6.23+); CPU time proportion-based; tracks virtual runtime (vruntime)
vruntime	Virtual runtime tracked by CFS; scheduler always picks task with lowest vruntime

1. When CPU Scheduling Decisions Occur

Case	Transition	Type
1	Running → Waiting	Non-preemptive
2	Running → Ready	Preemptive
3	Waiting → Ready	Preemptive
4	Terminates	Non-preemptive

2. Dispatcher Functions

Function	Description
Context Switching	Save state of old process and load state of new process
Switch to User Mode	Change CPU mode from kernel to user
Jump to Location	Jump to proper location in user program to resume execution

3. Scheduling Criteria

Criterion	Goal	Description
CPU Utilization	Maximize	Keep CPU as busy as possible
Throughput	Maximize	Processes completed per time unit
Turnaround Time	Minimize	Total time to execute a process
Waiting Time	Minimize	Time spent in the ready queue
Response Time	Minimize	Time from request to first response

4. Scheduling Algorithms Comparison

Algorithm	Rule	Preemptive	Advantage	Disadvantage
FCFS	First come, first served	No	Simple	Convoy effect; long jobs block short ones
SJF	Shortest next burst first	No	Optimal avg wait time	Must know next burst length in advance
SRTF	Shortest remaining time first	Yes	Optimal avg wait time	High overhead; needs remaining time info
Priority	Highest priority first	Both	Flexible	Starvation of low-priority processes
Round Robin	Each gets time quantum q	Yes	Fair; good response	Higher turnaround; overhead if q too small
Multilevel Queue	Separate queues; own algorithms	Both	Specialized per type	Starvation for lower-priority queues
Multilevel Feedback	Processes move between queues	Both	Flexible; no starvation	Complex to configure

5. SJF Schemes

Scheme	Description
Non-Preemptive SJF	Once CPU assigned, process cannot be preempted until it completes its CPU burst
Preemptive SJF (SRTF)	If new process arrives with shorter burst than remaining time of current process → preempt current process
Optimality	SJF minimizes the average waiting time for a given set of processes
Challenge	Hard to know next CPU burst length in advance; one solution: estimate based on past behavior

6. Priority Scheduling

Aspect	Detail
Rule	CPU given to process with highest priority; smallest integer = highest priority
Preemptive	Higher-priority process can preempt currently running process
Non-preemptive	CPU allocated to highest priority but cannot be preempted until it finishes
SJF as Priority	SJF = priority scheduling where priority is inverse of predicted next CPU burst time
Problem	Starvation — low-priority processes may never execute
Solution — Aging	Gradually increase priority of waiting processes over time to prevent starvation

7. Round Robin — Time Quantum Considerations

q Size	Behavior	Effect
Large q	Behaves like FIFO	Higher turnaround; simpler context switching
Small q	More context switches	Overhead too high if q < context switch time
Typical q	10–100ms; context switch < 10ms	Good balance of fairness and efficiency
Timer	Timer interrupt fires every q units	Triggers scheduler to select next process

8. Multilevel Queue Scheduling

Feature	Detail
Concept	Ready queue split into separate queues (e.g., Foreground / Background)
Assignment	Each process permanently assigned to a given queue
Foreground	Interactive processes → Round Robin (RR)
Background	Batch processes → First-Come, First-Served (FCFS)
Fixed Priority	Serve all foreground first, then background; risk: lower-queue starvation
Time Slice	Each queue gets a portion of CPU (e.g., 80% foreground RR, 20% background FCFS)

9. Multilevel Feedback Queue Example (3 Queues)

Queue	Algorithm	Quantum	Behavior
Q0	Round Robin	8 ms	New jobs enter Q0; if not finished → moved to Q1
Q1	Round Robin	16 ms	If still not completed → preempted and moved to Q2
Q2	FCFS	—	Runs until completion; no further demotion

10. Linux CPU Scheduling

Version	Algorithm	Key Details
Up to v2.5	UNIX-style → O(1)	Constant-time scheduling; preemptive SJF, priority-based
v2.5 Priority	Preemptive SJF	Real-time: 0–99 I Time-sharing: nice 100–140 I Lower value = higher priority I Higher priority → larger time slice
v2.6.23+ CFS	Completely Fair Scheduler	CPU time proportion-based I Quantum from nice (-20 to +19) I Tracks vruntime I Picks task with lowest vruntime I Target latency adjusts with active tasks

11. Windows CPU Scheduling

Feature	Detail
Type	Priority-based preemptive with 32 levels
Priority Levels	Variable class 1–15 I Real-time 16–31 I Priority 0 for memory thread
Dispatcher	Selects highest-priority thread to run
Thread Runs Until	Blocks, time slice expires, or preempted by higher-priority thread
Real-time Threads	Can preempt non-real-time; queued by priority; idle thread runs if none ready