UNIT 1
AN INTRODUCTION TO OPERATING SYSTEMS

Application software performs specific task for the user.
System software operates and controls the computer system and provides a platform to run application software.

An **operating system** is a piece of software that manages all the resources of a computer system,both hardware and software, and provides an environment in which the user can execute his/her programs in a convenient and efficient manner by hiding underlying complexity of the hardware and acting as a resource manager.

Why OS?

1. What if there is no OS?
    a. Bulky and complex app. (Hardware interaction code must be in app's code base)
    b. Resource exploitation by 1 App.
    c. No memory protection.
2. What is an OS made up of?
    a. Collection of system software.

An operating system function -

- Access to the computer hardware.
- interface between the user and the computer hardware
- **Resource management (Aka, Arbitration) (memory, device, file, security, process etc)**
- **Hides the underlying complexity of the hardware. (Aka, Abstraction)**
- facilitates execution of application programs by providing isolation and protection.

User

Application programs
Operating system
Computer hardware

The operating system provides the means for proper use of the resources in the operation of the computer system.

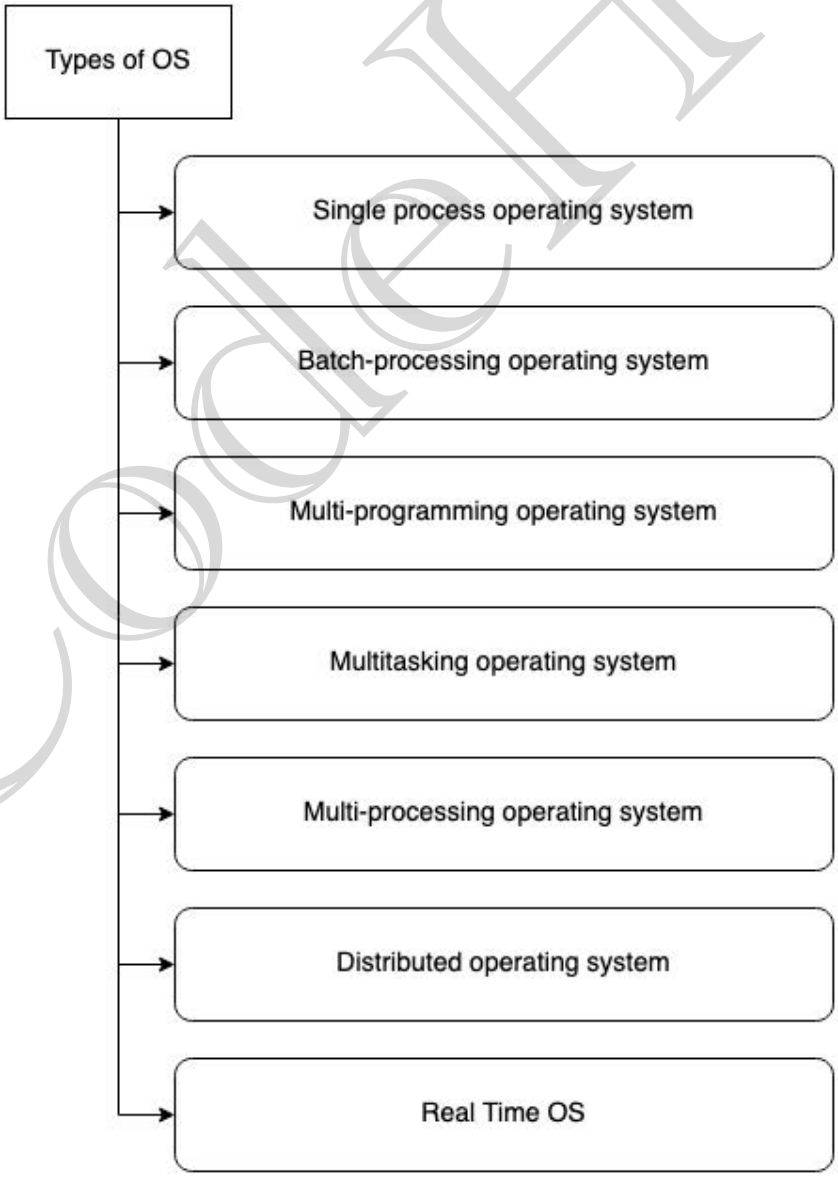
LEC-2: Types of OS

OS goals –

- Maximum CPU utilization
- Less process starvation
- Higher priority job execution

Types of operating systems –

- Single process operating system [MS DOS, 1981]
- Batch-processing operating system [ATLAS, Manchester Univ., late 1950s – early 1960s]
- Multiprogramming operating system [THE, Dijkstra, early 1960s]
- Multitasking operating system [CTSS, MIT, early 1960s]
- Multi-processing operating system [Windows NT]
- Distributed system [LOCUS]
- Real time OS [ATCS]

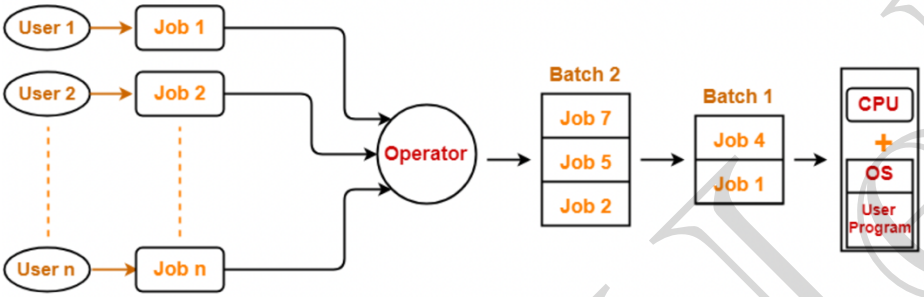


Single process OS, only 1 process executes at a time from the ready queue. [Oldest]

Batch-processing OS,

1. Firstly, user prepares his job using punch cards.
2. Then, he submits the job to the computer operator.
3. Operator collects the jobs from different users and sort the jobs into batches with similar needs.
4. Then, operator submits the batches to the processor one by one.
5. All the jobs of one batch are executed together.

- Priorities cannot be set, if a job comes with some higher priority.
- May lead to starvation. (A batch may take more time to complete)
- CPU may become idle in case of I/O operations.



Multiprogramming increases CPU utilization by keeping multiple jobs (code and data) in the**memory** so that the CPU always has one to execute in case some job gets busy with I/O.

- Single CPU
- Context switching for processes.
- Switch happens when current process goes to wait state.
- CPU idle time reduced.

Multitasking is a logical extension of multiprogramming.

- Single CPU
- Able to run more than one task simultaneously.
- Context switching and time sharing used.
- Increases responsiveness.
- CPU idle time is further reduced.

Multi-processing OS, more than 1 CPU in a single computer.

- Increases reliability, 1 CPU fails, other can work
- Better throughput.
- Lesser process starvation, (if 1 CPU is working on some process, other can be executed on other CPU.

Distributed OS,

- OS manages many bunches of resources, >=1 CPUs, >=1 memory, >=1 GPUs, etc
- **Loosely connected autonomous,** interconnected computer nodes.
- collection of independent, networked, communicating, and physically separate computational nodes.

RTOS

- **Real time** error free, computations within tight-time boundaries.
- Air Traffic control system, ROBOTS etc.

LEC-3: Multi-Tasking vs Multi-Threading

Program: A Program is an executable file which contains a certain set of instructions written to complete the specific job or operation on your computer.

- It's a compiled code. Ready to be executed.
- Stored in Disk

Process: Program under execution. Resides in Computer's primary memory (RAM).

Thread:

- Single sequence stream within a process.
- An independent path of execution in a process.
- Light-weight process.
- Used to achieve parallelism by dividing a process's tasks which are independent path of execution.
- E.g., Multiple tabs in a browser, text editor (When you are typing in an editor, spell-checking, formatting of text and saving the text are done concurrently by multiple threads.)

Multi-Tasking	**Multi-Threading**
The execution of more than one task simultaneously is called as multitasking.	A process is divided into several different sub-tasks called as threads, which has its own path of execution. This concept is called as multithreading.
Concept of more than 1 processes being context switched.	Concept of more than 1 thread. Threads are context switched.
No. of CPU 1.	No. of CPU >= 1. (Better to have more than 1)
Isolation and memory protection exists. OS must allocate separate memory and resources to each program that CPU is executing.	**No isolation and memory protection**, resources are shared among threads of that process. OS allocates memory to a process; multiple threads of that process share the same memory and resources allocated to the process.

Thread Scheduling:
Threads are scheduled for execution based on their priority. Even though threads are executing within the runtime, all threads are assigned processor time slices by the operating system.

Difference between Thread Context Switching and Process Context Switching:

Thread Context switching	Process context switching
OS saves current state of thread & switches to another thread of same process.	OS saves current state of process & switches to another process by restoring its state.

Doesn’t includes switching of memory address space. (But Program counter, registers & stack are included.)	Includes switching of memory address space.
Fast switching.	Slow switching.
CPU’s cache state is preserved.	CPU’s cache state is flushed.

LEC-4: Components of OS

1. **Kernel:** A **kernel** is that part of the operating system which interacts directly with the hardware andperforms the most crucial tasks.
    a. Heart of OS/Core component
    b. Very first part of OS to load on start-up.
2. **User space:** Where application software runs, apps don't have privileged access to the underlying hardware. It interacts with kernel.
    a. GUI
    b. CLI

A **shell**, also known as a command interpreter, is that part of the operating system that receives commands from the users and gets them executed.

Functions of Kernel:

1. **Process management:**
    a. Scheduling processes and threads on the CPUs.
    b. Creating & deleting both user and system process.
    c. Suspending and resuming processes
    d. Providing mechanisms for process synchronization or process communication.
2. **Memory management:**
    a. Allocating and deallocating memory space as per need.
    b. Keeping track of which part of memory are currently being used and by which process.
3. **File management:**
    a. Creating and deleting files.
    b. Creating and deleting directories to organize files.
    c. Mapping files into secondary storage.
    d. Backup support onto a stable storage media.
4. **I/O management:** to manage and control I/O operations and I/O devices
    a. Buffering (data copy between two devices), caching and spooling.
        i. Spooling
            1. Within differing speed two jobs.
            2. Eg. Print spooling and mail spooling.
        ii. Buffering
            1. Within one job.
            2. Eg. Youtube video buffering
        iii. Caching
            1. Memory caching, Web caching etc.

Types of Kernels:

1. Monolithic kernel
    a. All functions are in kernel itself.
    b. Bulky in size.
    c. Memory required to run is high.
    d. Less reliable, one module crashes -> whole kernel is down.
    e. High performance as communication is fast. (Less user mode, kernel mode overheads)
    f. Eg. Linux, Unix, MS-DOS.

2. Micro Kernel
    a. Only major functions are in kernel.
        i. Memory mgmt.
        ii. Process mgmt.
    b. File mgmt. and IO mgmt. are in User-space.
    c. smaller in size.
    d. More Reliable
    e. More stable
    f. Performance is slow.
    g. Overhead switching b/w user mode and kernel mode.
    h. Eg. L4 Linux, Symbian OS, MINIX etc.

3. Hybrid Kernel:
    a. Advantages of both worlds. (File mgmt. in User space and rest in Kernel space.)
    b. Combined approach.
    c. Speed and design of mono.
    d. Modularity and stability of micro.
    e. Eg. MacOS, Windows NT/7/10
    f. IPC also happens but lesser overheads
4. Nano/Exo kernels…

Q. How will communication happen between user mode and kernel mode?
Ans. Inter process communication (**IPC**).

1. Two processes executing independently, having independent memory space (Memory protection), But some may need to communicate to work.

2. Done by shared memory and message passing.

LEC-5: System Calls

How do apps interact with Kernel? -> using system calls.

Eg. Mkdir laks

- Mkdir indirectly calls kernel and asked the file mgmt. module to create a new directory.
- Mkdir is just a wrapper of actual system calls.
- Mkdir interacts with kernel using system calls.

Eg. Creating a process.

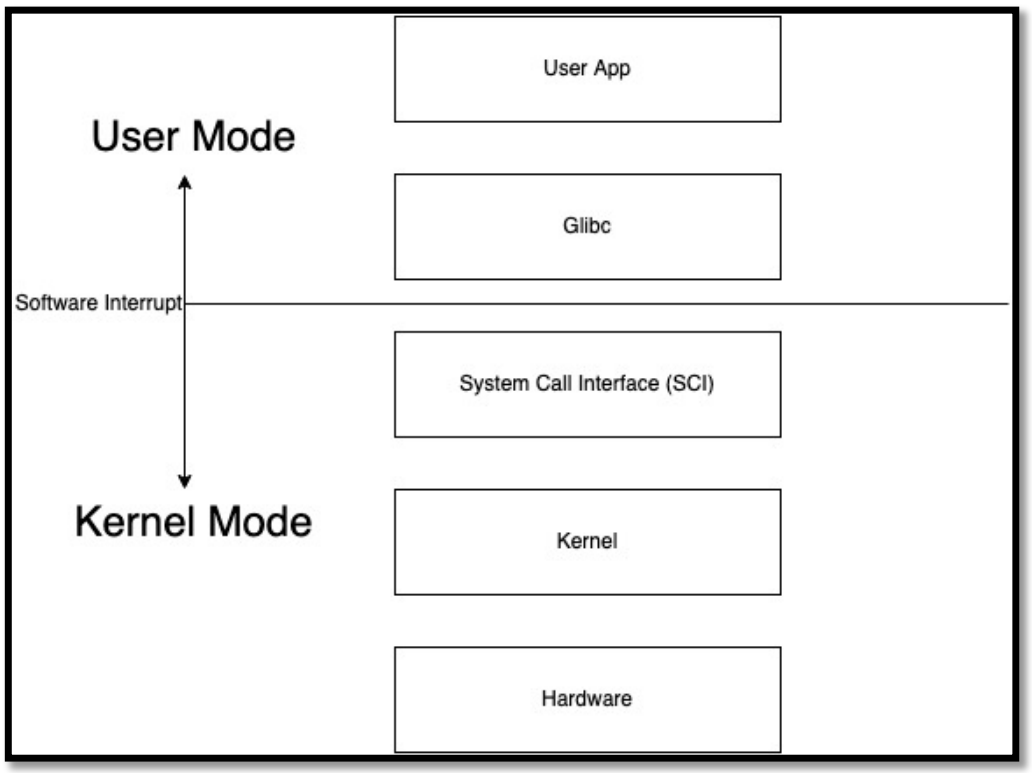
- User executes a process. (User space)
- Gets system call. (US)
- Exec system call to create a process. (KS)
- Return to US.

Transitions from US to KS done by software interrupts.

System calls are implemented in C.

A **system call** is a mechanism using which a user program can request a service from the kernel for which it does not have the permission to perform.
User programs typically do not have permission to perform operations like accessing I/O devices and communicating other programs.

System Calls are the only way through which a process can go into **kernel mode from user mode.**



Types of System Calls:

1) Process Control
   a. end, abort
   b. load, execute
   c. create process, terminate process
   d. get process attributes, set process attributes
   e. wait for time
   f. wait event, signal event
   g. allocate and free memory

2) File Management
   a. create file, delete file
   b. open, close
   c. read, write, reposition
   d. get file attributes, set file attributes

3) Device Management
   a. request device, release device
   b. read, write, reposition
   c. get device attributes, set device attributes
   d. logically attach or detach devices

4) Information maintenance
   a. get time or date, set time or date
   b. get system data, set system data
   c. get process, file, or device attributes
   d. set process, file, or device attributes

5) Communication Management
   a. create, delete communication connection
   b. send, receive messages
   c. transfer status information
   d. attach or detach remote devices

Examples of Windows & Unix System calls:

Category	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Management	CreateFile() ReadFile() WriteFile() CloseHandle() SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	open () read () write () close () chmod() umask(chown()
Device Management	SetConsoleMode() ReadConsole() WriteConsole()	ioctI() read() write()
Information Management	GetCurrentProcessID() SetTimer() Sleep()	getpid () alarm () sleep ()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe () shmget () mmap()

LEC-6: What happens when you turn on your computer?

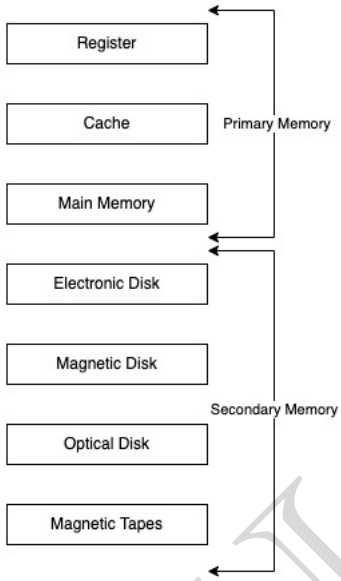
i. PC On
ii. CPU initializes itself and looks for a firmware program (BIOS) stored in BIOS Chip **(**Basic input-output system chip is a ROM chip found on mother board that allows to access & setup computer system at most basic level.)
    1. In modern PCs, CPU loads UEFI (Unified extensible firmware interface)
iii. **CPU** runs the BIOS which tests and initializes system hardware. Bios loads configuration settings. If something is not appropriate (like missing RAM) error is thrown and boot process is stopped.
This is called **POST** (Power on self-test) process.
(UEFI can do a lot more than just initialize hardware; it's really a tiny operating system. For example, Intel CPUs have the Intel Management Engine. This provides a variety of features, including powering Intel's Active Management Technology, which allows for remote management of business PCs.)
iv. **BIOS** will handoff responsibility for booting your PC to your OS's bootloader.
    1. BIOS looked at the MBR (master boot record), a special boot sector at the beginning of a disk. The MBR contains code that loads the rest of the operating system, known as a "bootloader." The BIOS executes the bootloader, which takes it from there and begins booting the actual operating system—Windows or Linux, for example.
    In other words,
    the BIOS or UEFI examines a storage device on your system to look for a small program, either in the MBR or on an EFI system partition, and runs it.
v. The bootloader is a small program that has the large task of booting the rest of the operating system (Boots Kernel then, User Space). Windows uses a bootloader named Windows Boot Manager (Bootmgr.exe), most Linux systems use GRUB, and Macs use something called boot.efi

Lec-7: 32-Bit vs 64-Bit OS

1. A 32-bit OS has 32-bit registers, and it can access 2^32 unique memory addresses. i.e., 4GB of physical memory.
2. A 64-bit OS has 64-bit registers, and it can access 2^64 unique memory addresses. i.e., 17,179,869,184 GB of physical memory.
3. 32-bit CPU architecture can process 32 bits of data & information.
4. 64-bit CPU architecture can process 64 bits of data & information.
5. Advantages of 64-bit over the 32-bit operating system:
    a. **Addressable Memory**: 32-bit CPU -> 2^32 memory addresses, 64-bit CPU -> 2^64 memory addresses.
    b. **Resource usage**: Installing more RAM on a system with a 32-bit OS doesn't impact performance. However, upgrade that system with excess RAM to the 64-bit version of Windows, and you'll notice a difference.
    c. **Performance**: All calculations take place in the registers. When you're performing math in your code, operands are loaded from memory into registers. So, having larger registers allow you to perform larger calculations at the same time.
    32-bit processor can execute 4 bytes of data in 1 instruction cycle while 64-bit means that processor can execute 8 bytes of data in 1 instruction cycle.
    (In 1 sec, there could be thousands to billons of instruction cycles depending upon a processor design)
    d. **Compatibility**: 64-bit CPU can run both 32-bit and 64-bit OS. While 32-bit CPU can only run 32-bit OS.
    e. **Better Graphics performance**: 8-bytes graphics calculations make graphics-intensive apps run faster.

Lec-8: Storage Devices Basics

What are the different memory present in the computer system?



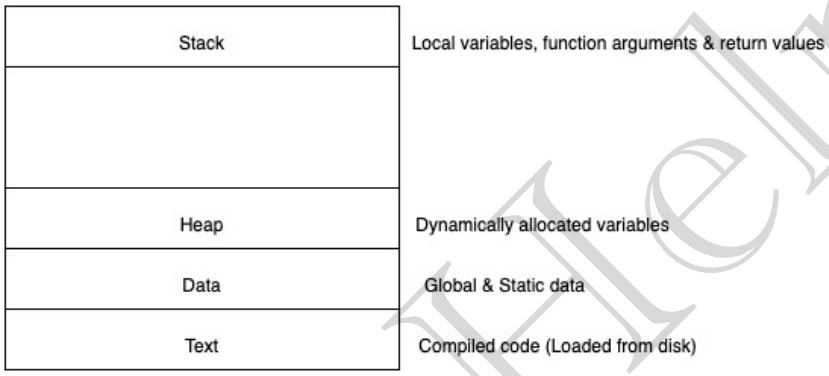
1. **Register**: Smallest unit of storage. It is a part of CPU itself.
   A register may hold an instruction, a storage address, or any data (such as bit sequence or individual characters).
   Registers are a type of computer memory used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU.
2. **Cache**: Additional memory system that temporarily stores frequently used instructions and data for quicker processing by the CPU.
3. **Main Memory**: RAM.
4. **Secondary Memory**: Storage media, on which computer can store data & programs.

Comparison

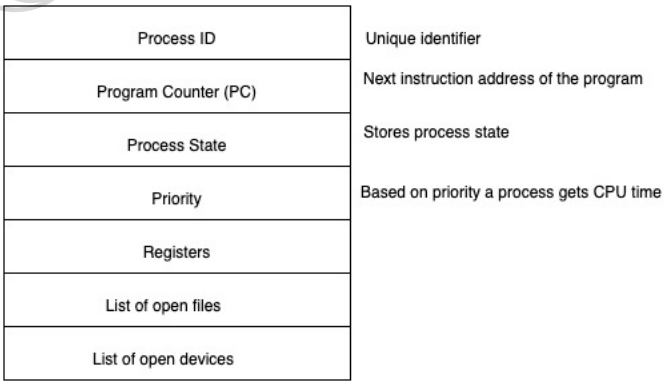
1. **Cost**:
   a. Primary storages are costly.
   b. Registers are most expensive due to expensive semiconductors & labour.
   c. Secondary storages are cheaper than primary.
2. **Access Speed**:
   a. Primary has higher access speed than secondary memory.
   b. Registers has highest access speed, then comes cache, then main memory.
3. **Storage size**:
   a. Secondary has more space.
4. **Volatility**:
   a. Primary memory is volatile.
   b. Secondary is non-volatile.

Lec-9: Introduction to Process

1. What is a program? Compiled code, that is ready to execute.
2. What is a process? Program under execution.
3. How OS creates a process? Converting program into a process.
   STEPS:
   a. Load the program & static data into memory.
   b. Allocate runtime stack.
   c. Heap memory allocation.
   d. IO tasks.
   e. OS handoffs control to main ().
4. **Architecture** of process:



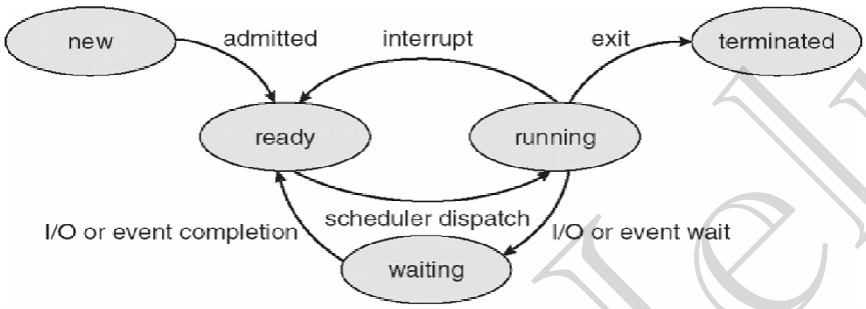
5. **Attributes** of process:
   a. Feature that allows identifying a process uniquely.
   b. Process table
      i. All processes are being tracked by OS using a table like data structure.
      ii. Each entry in that table is process control block (PCB).
   c. PCB: Stores info/attributes of a process.
      i. Data structure used for each process, that stores information of a process such as process id, program counter, process state, priority etc.
6. **PCB structure**:



Registers in the PCB, it is a data structure. When a processes is running and it's time slice expires, the current value of process specific registers would be stored in the PCB and the process would be swapped out. When the process is scheduled to be run, the register values is read from the PCB and written to the CPU registers. This is the main purpose of the registers in the PCB.

Lec-10: Process States | Process Queues

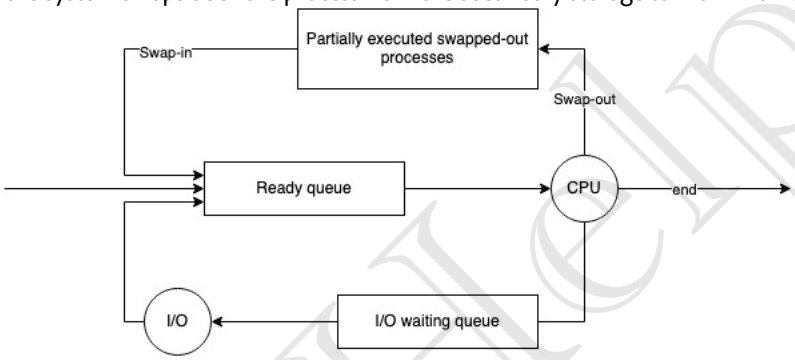
1. **Process States**: As process executes, it changes state. Each process may be in one of the following states.
    a. **New**: OS is about to pick the program & convert it into process. OR the process is being created.
    b. **Run**: Instructions are being executed; CPU is allocated.
    c. **Waiting**: Waiting for IO.
    d. **Ready**: The process is in memory, waiting to be assigned to a processor.
    e. **Terminated**: The process has finished execution. PCB entry removed from process table.



2. **Process Queues:**
    a. Job Queue:
        i. Processes in new state.
        ii. Present in secondary memory.
        iii. **Job Schedular (Long term schedular (LTS))** picks process from the pool and loads them into memory for execution.
    b. Ready Queue:
        i. Processes in Ready state.
        ii. Present in main memory.
        iii. **CPU Schedular (Short-term schedular)** picks process from ready queue and dispatch it to CPU.
    c. Waiting Queue:
        i. Processes in Wait state.
3. **Degree of multi-programming**: The number of processes in the memory.
    a. **LTS** controls degree of multi-programming.
4. **Dispatcher:** The module of OS that gives control of CPU to a process selected by **STS**.

LEC-11: Swapping | Context-Switching | Orphan process | Zombie process

1. **Swapping**
   - **a.** Time-sharing system may have medium term schedular (MTS).
   - **b.** Remove processes from memory to reduce degree of multi-programming.
   - **c.** These removed processes can be reintroduced into memory, and its execution can be continued where it left off. This is called **Swapping.**
   - **d.** Swap-out and swap-in is done by MTS.
   - **e.** Swapping is necessary to improve process mix or because a change in memory requirements has overcommitted available memory, requiring memory to be freed up.
   - f. Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.



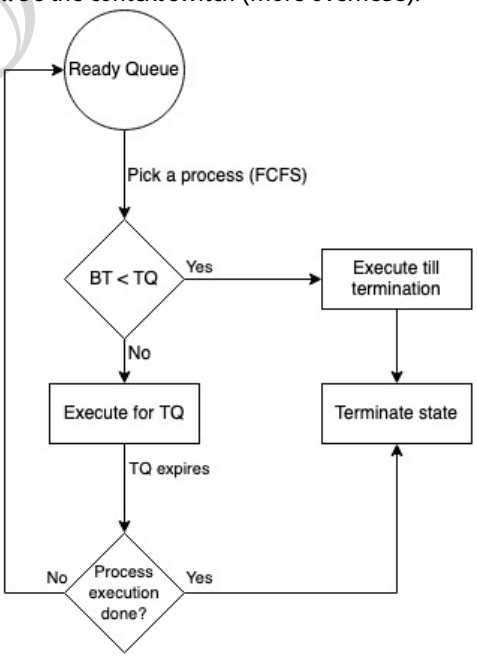
2. **Context-Switching**
   - a. Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process.
   - b. When this occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.
   - c. It is pure overhead, because the system does no useful work while switching.
   - d. Speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied etc.
3. **Orphan process**
   - **a.** The process whose parent process has been terminated and it is still running.
   - **b.** Orphan processes are adopted by init process.
   - **c.** Init is the first process of OS.
4. **Zombie process / Defunct process**
   - a. A zombie process is a process whose execution is completed but it still has an entry in the process table.
   - b. Zombie processes usually occur for child processes, as the parent process still needs to read its child's exit status. Once this is done using the wait system call, the zombie process is eliminated from the process table. This is known as **reaping** the zombie process.
   - c. It is because parent process may call wait () on child process for a longer time duration and child process got terminated much earlier.
   - d. As entry in the process table can only be removed, after the parent process reads the exit status of child process. Hence, the child process remains a zombie till it is removed from the process table.

LEC-12: Intro to Process Scheduling | FCFS | Convoy Effect

1. **Process Scheduling**
    a. Basis of Multi-programming OS.
    b. By switching the CPU among processes, the OS can make the computer more productive.
    c. Many processes are kept in memory at a time, when a process must wait or time quantum expires, the OS takes the CPU away from that process & gives the CPU to another process & this pattern continues.
2. **CPU Scheduler**
    a. Whenever the CPU become ideal, OS must select one process from the ready queue to be executed.
    b. Done by STS.
3. **Non-Preemptive scheduling**
    a. Once CPU has been allocated to a process, the process keeps the CPU until it releases CPU either by terminating or by switching to wait-state.
    b. Starvation, as a process with long burst time may starve less burst time process.
    c. Low CPU utilization.
4. **Preemptive scheduling**
    a. CPU is taken away from a process after time quantum expires along with terminating or switching to wait-state.
    b. Less Starvation
    c. High CPU utilization.
5. **Goals of CPU scheduling**
    a. Maximum CPU utilization
    b. Minimum Turnaround time (TAT).
    c. Min. Wait-time
    d. Min. response time.
    e. Max. throughput of system.
6. **Throughput**: No. of processes completed per unit time.
7. **Arrival time (AT)**: Time when process is arrived at the ready queue.
8. **Burst** time (BT): The time required by the process for its execution.
9. **Turnaround** time (TAT): Time taken from first time process enters ready state till it terminates. (CT - AT)
10. **Wait** time (WT): Time process spends waiting for CPU. (WT = TAT – BT)
11. **Response** time: Time duration between process getting into ready queue and process getting CPU for the first time.
12. **Completion** Time (CT): Time taken till process gets terminated.
13. **FCFS** (First come-first serve):
    a. Whichever process comes first in the ready queue will be given CPU first.
    b. In this, if one process has longer BT. It will have major effect on average WT of diff processes, called **Convoy effect.**
    c. Convoy Effect is a situation where many processes, who need to use a resource for a short time, are blocked by one process holding that resource for a long time.
        i. This cause poor resource management.

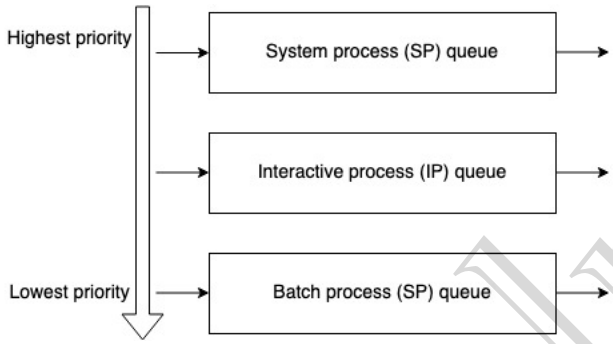
LEC-13: CPU Scheduling | SJF | Priority | RR

1. Shortest Job First (SJF) [Non-preemptive]
    a. Process with least BT will be dispatched to CPU first.
    b. Must do estimation for BT for each process in ready queue beforehand, Correct estimation of BT is an impossible task (ideally.)
    c. Run lowest time process for all time then, choose job having lowest BT at that instance.
    d. This will suffer from convoy effect as if the very first process which came is Ready state is having a large BT.
    e. Process starvation might happen.
    f. Criteria for SJF algos, AT + BT.
2. SJF [Preemptive]
    a. Less starvation.
    b. No convoy effect.
    c. Gives average WT less for a given set of processes as scheduling short job before a long one decreases the WT of short job more than it increases the WT of the long process.
3. Priority Scheduling [Non-preemptive]
    a. Priority is assigned to a process when it is created.
    b. SJF is a special case of general priority scheduling with priority inversely proportional to BT.
4. Priority Scheduling [Preemptive]
    a. Current RUN state job will be preempted if next job has higher priority.
    b. May cause indefinite waiting (Starvation) for lower priority jobs. (Possibility is they won't get executed ever). (True for both preemptive and non-preemptive version)
        i. Solution: Ageing is the solution.
        ii. Gradually increase priority of process that wait so long. E.g., increase priority by 1 every 15 minutes.
5. Round robin scheduling (RR)
    a. Most popular
    b. Like FCFS but preemptive.
    c. Designed for time sharing systems.
    d. Criteria: AT + time quantum (TQ), Doesn't depend on BT.
    e. No process is going to wait forever, hence very low starvation. [No convoy effect]
    f. Easy to implement.
    g. If TQ is small, more will be the context switch (more overhead).



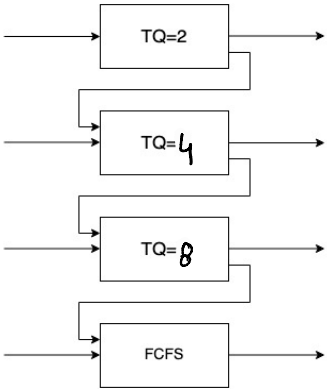
LEC-14: MLQ | MLFQ

1. Multi-level queue scheduling (**MLQ**)
    a. Ready queue is divided into multiple queues depending upon priority.
    b. A process is permanently assigned to one of the queues (inflexible) based on some property of process, memory, size, process priority or process type.
    c. Each queue has its own scheduling algorithm. E.g., SP -> RR, IP -> RR & BP -> FCFS.



    d. System process: Created by OS (Highest priority)
    Interactive process (Foreground process): Needs user input (I/O).
    Batch process (Background process): Runs silently, no user input required.
    e. Scheduling among different sub-queues is implemented as **fixed priority preemptive** scheduling. E.g., foreground queue has absolute priority over background queue.
    f. If an interactive process comes & batch process is currently executing. Then, batch process will be preempted.
    g. Problem: Only after completion of all the processes from the top-level ready queue, the further level ready queues will be scheduled.
    This came starvation for lower priority process.
    h. Convoy effect is present.

2. Multi-level feedback queue scheduling (**MLFQ**)
    a. Multiple sub-queues are present.
    b. Allows the process to move between queues. The idea is to separate processes according to the characteristics of their BT. If a process uses too much CPU time, it will be moved to lower priority queue. This scheme leaves I/O bound and interactive processes in the higher-priority queue.
    In addition, a process that waits too much in a lower-priority queue may be moved to a higher priority queue. This form of ageing prevents starvation.
    c. Less starvation then MLQ.
    d. It is flexible.
    e. Can be configured to match a specific system design requirement.
    Sample MLFQ design:



3. Comparison:

	FCFS	SJF	PSJF	Priority	P-Priority	RR	MLQ	MLFQ
Design	Simple	Complex	Complex	Complex	Complex	Simple	Complex	Complex
Preemption	No	No	Yes	No	Yes	Yes	Yes	Yes
Convoy effect	Yes	Yes	No	Yes	Yes	No	Yes	Yes
Overhead	No	No	Yes	No	Yes	Yes	Yes	Yes

LEC-15: Introduction to Concurrency

1. **Concurrency** is the execution of the multiple instruction sequences at the same time. It happens in the operating system when there are several process threads running in parallel.
2. **Thread**:
    - Single sequence stream within a process.
    - An independent path of execution in a process.
    - Light-weight process.
    - Used to achieve parallelism by dividing a process's tasks which are independent path of execution.
    - E.g., Multiple tabs in a browser, text editor (When you are typing in an editor, spell checking, formatting of text and saving the text are done concurrently by multiple threads.)
3. **Thread Scheduling:** Threads are scheduled for execution based on their priority. Even though threads are executing within the runtime, all threads are assigned processor time slices by the operating system.
4. **Threads context switching**
    - OS saves current state of thread & switches to another thread of same process.
    - Doesn't includes switching of memory address space. (But Program counter, registers & stack are included.)
    - Fast switching as compared to process switching
    - CPU's cache state is preserved.
5. **How each thread get access to the CPU?**
    - Each thread has its own program counter.
    - Depending upon the thread scheduling algorithm, OS schedule these threads.
    - OS will fetch instructions corresponding to PC of that thread and execute instruction.
6. **I/O or TQ, based context switching is done here as well**
    - We have TCB (Thread control block) like PCB for state storage management while performing context switching.
7. **Will single CPU system would gain by multi-threading technique?**
    - Never.
    - As two threads have to context switch for that single CPU.
    - This won't give any gain.
8. **Benefits of Multi-threading.**
    - Responsiveness
    - Resource sharing: Efficient resource sharing.
    - Economy: It is more economical to create and context switch threads.
        1. Also, allocating memory and resources for process creation is costly, so better to divide tasks into threads of same process.
    - Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

LEC-16: Critical Section Problem and How to address it

1. Process synchronization techniques play a key role in maintaining the consistency of shared data
2. **Critical Section (C.S)**
    a. The critical section refers to the segment of code where processes/threads access shared resources, such as common variables and files, and perform write operations on them. Since processes/threads execute concurrently, any process can be interrupted mid-execution.
3. **Major Thread scheduling issue**
    a. Race Condition
        i. A race condition occurs when two or more threads can access shared data and they try to change it at the same time. Because the thread scheduling algorithm can swap between threads at any time, you don't know the order in which the threads will attempt to access the shared data. Therefore, the result of the change in data is dependent on the thread scheduling algorithm, i.e., both threads are "racing" to access/change the data.
4. **Solution to Race Condition**
    a. Atomic operations: Make Critical code section an atomic operation, i.e., Executed in one CPU cycle.
    b. Mutual Exclusion using locks.
    c. Semaphores
5. Can we use a simple flag variable to solve the problem of race condition?
    a. No.
6. **Peterson's solution** can be used to avoid race condition but holds good for only 2 process/threads.
7. **Mutex/Locks**
    a. Locks can be used to implement mutual exclusion and avoid race condition by allowing only one thread/process to access critical section.
    b. Disadvantages:
        i. Contention: one thread has acquired the lock, other threads will be busy waiting, what if thread that had acquired the lock dies, then all other threads will be in infinite waiting.
        ii. Deadlocks
        iii. Debugging
        iv. Starvation of high priority threads.

LEC-17: Conditional Variable and Semaphores for Threads synchronization

1. **Conditional variable**
    a. The condition variable is a synchronization primitive that lets the thread wait until a certain condition occurs.
    b. Works with a lock
    c. Thread can enter a wait state only when it has acquired a lock. When a thread enters the wait state, it will release the lock and wait until another thread notifies that the event has occurred. Once the waiting thread enters the running state, it again acquires the lock immediately and starts executing.
    d. Why to use conditional variable?
        i. To avoid busy waiting.
    e. **Contention** is not here.
2. **Semaphores**
    a. Synchronization method.
    b. An integer that is equal to number of resources
    c. Multiple threads can go and execute C.S concurrently.
    d. Allows multiple program threads to access the finite instance of resources whereas mutex allows multiple threads to access a single shared resource one at a time.
    e. Binary semaphore: value can be 0 or 1.
        i. Aka, mutex locks
    f. Counting semaphore
        i. Can range over an unrestricted domain.
        ii. Can be used to control access to a given resource consisting of a finite number of instances.
    g. To overcome the need for busy waiting, we can modify the definition of the wait () and signal () semaphore operations. When a process executes the wait () operation and finds that the semaphore value is not positive, it must wait. However, rather than engaging in busy waiting, the process car block itself. The block- operation places a process into a waiting queue associated with the semaphore, and the state of the process is switched to the Waiting state. Then control is transferred to the CPU scheduler, which selects another process to execute.
    h. A process that is blocked, waiting on a semaphore S, should be restarted when some other process executes a signal () operation. The process is restarted by a wakeup () operation, which changes the process from the waiting state to the ready state. The process is then placed in the ready queue.