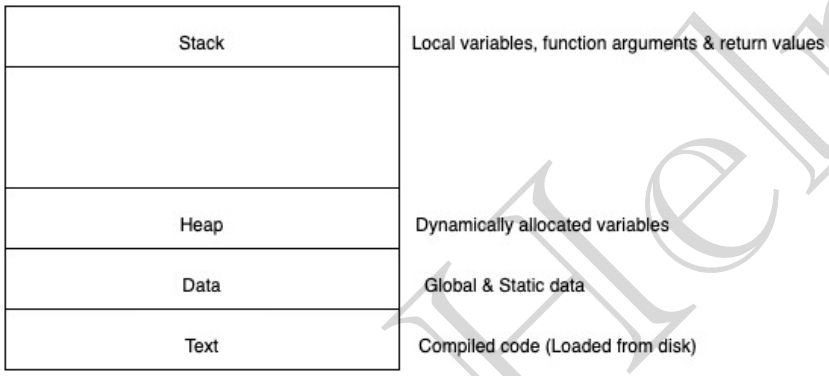
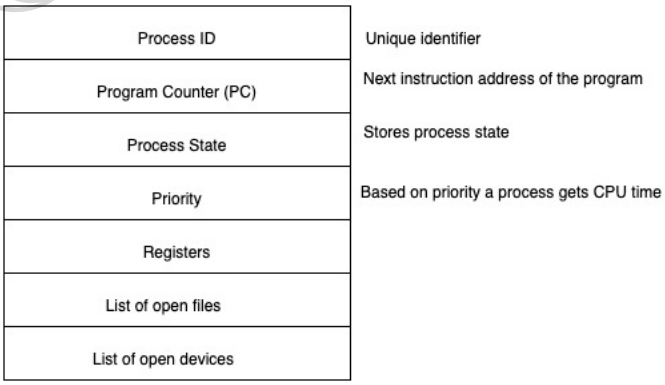
Lec-9: Introduction to Process

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:



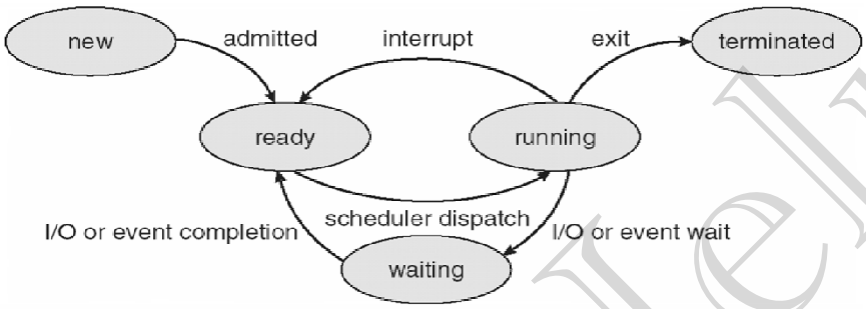
5. **Attributes** of process:
   a. Feature that allows identifying a process uniquely.
   b. Process table
      i. All processes are being tracked by OS using a table like data structure.
      ii. Each entry in that table is process control block (PCB).
   c. PCB: Stores info/attributes of a process.
      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

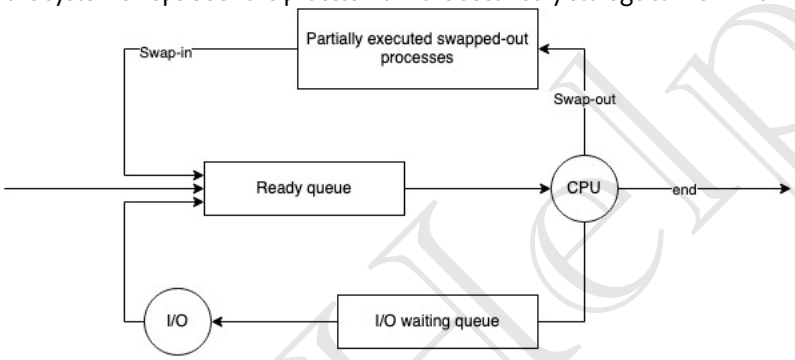
1. **Process States**: As process executes, it changes state. Each process may be in one of the following states.
    a. **New**: OS is about to pick the program & convert it into process. OR the process is being created.
    b. **Run**: Instructions are being executed; CPU is allocated.
    c. **Waiting**: 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.



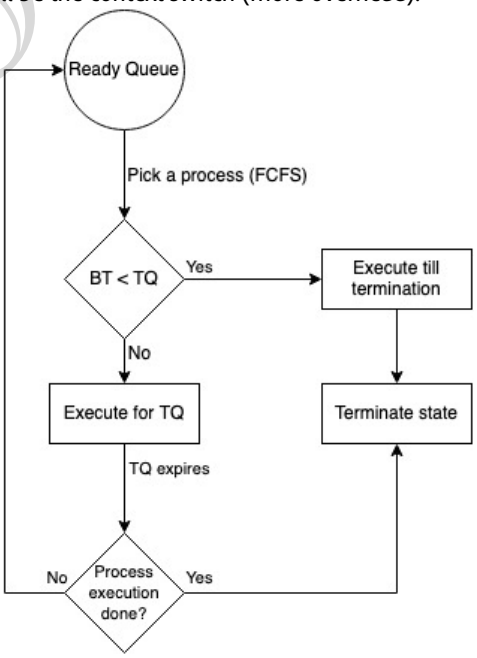
2. **Context-Switching**
   - a. Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process.
   - b. When this occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process 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.

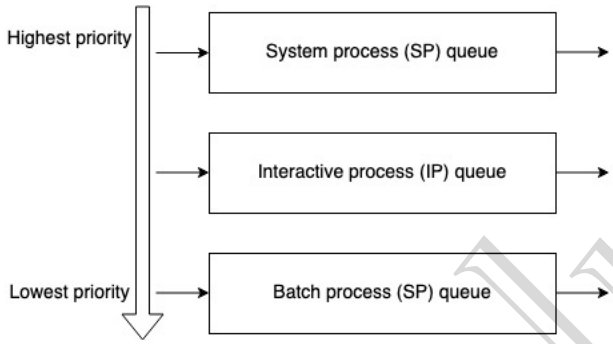
LEC-13: CPU Scheduling | SJF | Priority | RR

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).



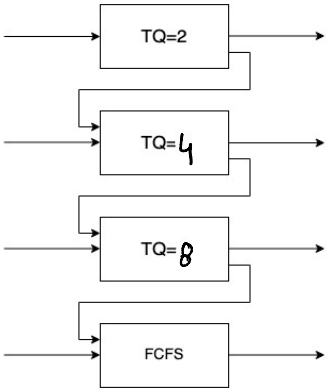
LEC-14: MLQ | MLFQ

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.