Lab 3 - Process Scheduling Algorithms

- Process scheduling is used to timeshare a processor amongst multiple processes that are ready to run, enabling concurrent execution of several processes. It is one of the central mechanisms that enables multiprogramming and efficiency.

- When the scheduler is invoked, it must pick one of the several ready processes to run. A scheduling policy specifies how a process is picked. The scheduling policy also determines the data structure that is used to store the list of ready processes (or PCBs). Different scheduling policies have different goals. Some may want to support fast response time for interactive processes. Some may want to maximize the efficiency and throughput of the CPU and I/O devices. Some may want to maximize fairness across processes. Different operating systems have different goals, and may end up with different scheduling policies in their implementations. Some operating systems let users configure priorities for processes that can help influence the scheduling decision.

- The simplest scheduling policy is the **First-Come-First-Serve (FCFS)** policy. Every process that is ready to run is placed at the end of a ready queue, and the scheduler goes over each process and runs it until it finishes or gives up the CPU voluntarily (i.e., blocks). This policy usually leads to longer average waiting times, e.g., when a lot of short processes get stuck in the queue behind a long one.

- A preemptible version of FCFS is the **Round Robin (RR**) policy. Here, the scheduler runs every process for a certain time quantum or slice, and moves on to the next process in the queue once the slice is finished. The performance of the RR algorithm heavily depends on the size of the quantum. Very small slices lead to good response times, but waste a lot of CPU cycles on context switching.

- In the **Shortest Job First (SJF)** policy, the process with the smallest execution time in the ready queue is picked for execution. This policy is provably optimal, achieving the lowest average wait time. However, predicting the execution time of a process based on past sizes of its CPU bursts is somewhat inaccurate, hence this policy is somewhat impractical to implement. A preemptive version of this policy is called the shortest remaining time first policy. Under this policy, when a new process arrives with a shorter execution time than that of the currently running process, the running process is preempted in favor of the new process. A heap-like data structure is more suited to storing the list of ready processes with such policies.

- With **priority scheduling,** processes are assigned a numerical priority, and the highest priority processes are scheduled before the lower priority ones. SJF is a special case of priority scheduling, where the priority is inversely proportional to the run time. Priority scheduling can be preemptive or non-preemptive. Priority can be defined by the user, or can be internally arrived at by the kernel. For example, it makes sense to prioritize I/O-bound processes over CPU-bound processes, in order to fully utilize the I/O devices. A potential problem with this policy is the starvation of low priority processes, which can be fixed by increasing the priority of processes as they wait longer.

1. Write algorithm and implement the same for the following scheduling algorithms.
    a. First Come First Serve
    b. Shortest Job First(With and without Preemption)
    c. Round Robin (Time slice with 2)

Instructions:

- ❏ Implement above algorithms in any programming language.
- ❏ Calculate the number of context switches in case of Round Robin.
- ❏ Run the above programs to find the average TurnAroundTime and average waiting time for the problems discussed in class.