*1. Scheduling Lab*

**A Simple Scheduling Simulation**

In this lab you will modify code to simulate the following 4 scheduling algorithms.

1. First Come First Serve (FCFS) - **WORKING SOLUTION COMPLETED**
2. Priority
3. Shortest Job First (SJF)
4. Round Robin (RR)

First, change directories to the SchedSim directory.

$ cd SchedSim

You should see several files such as a *Makefile* and *schedsim\_ref*. *schedsim\_ref* is the reference implementation to test your solution against. You should also see the **schedsim.c** file. This is the only file you will modify. We have provided much of the code. The current code reads input data from a file or from stdin that will represent a list of processes arriving. The file format consist of lines of text, where each line denotes the information regarding a single process. This information is in the following order to match the data structure below.

**typedef** **struct** **Process** {

**int** pid; *// Process ID*

**int** bt; *// Burst Time*

**int** art; *// Arrival Time*

**int** wt; *// waiting time*

**int** tat; *// turnaround time*

**int** pri; *// priority*

}ProcessType;

Ex: To keep the simulation simple, assume all processes will arrive at the same time 0. Should you choose at a later time to account for process arrival at different times, feel free to make additional changes. But that is not required for your assignment.

**First Come First Serve (FCFS)**

Given n processes with their burst times, the task is to find average waiting time and average turn around time using FCFS scheduling algorithm. First in, first out (FIFO), also known as first come, first served (FCFS), is the simplest scheduling algorithm. FIFO simply queues processes in the order that they arrive in the ready queue.  
In this, the process that comes first will be executed first and next process starts only after the previous gets fully executed.  
Here we are considering that arrival time for all processes is 0.

**Implementation**

Completion Time: Time at which process completes its execution.

Turn Around Time: Time Difference between completion time and arrival time. *Turn Around Time = Completion Time – Arrival Time*

Waiting Time: Time Difference between turn around time and burst time. *Waiting Time = Turn Around Time – Burst Time*

**WE HAVE PROVIDED A WORKING SOLUTION FOR FCFS**

**Prioirty**

Priority scheduling is one of the most common scheduling algorithms in batch systems. Each process is assigned a priority. Process with the highest priority is to be executed first and so on.

Processes with the same priority are executed on first come first served basis. Priority can be decided based on memory requirements, time requirements or any other resource requirement.

**Implementation**

1- First input the processes with their burst time and priority.  
2- Sort the processes, burst time and priority according to the priority.  
3- Now simply apply FCFS algorithm.

**YOU MUST WRITE A SOLUTION TO SORT THE LIST OF PROCESSES, THEN APPLY FCFS CODE**

**Sorting**

Use the built in c library function *qsort* for sorting. We used this in Lab 4: Part 1 Function Pointers.

**Shortest Job First (SJF) Pre-emptive or SRTF**

Shortest job first (SJF) or shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJF is a non-preemptive algorithm.

SJF has the advantage of having a minimum average waiting time among all scheduling algorithms. It is a Greedy Algorithm. It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of aging.

It is practically infeasible as Operating System may not know burst time and therefore may not sort them. While it is not possible to predict execution time, several methods can be used to estimate the execution time for a job, such as a weighted average of previous execution times. SJF can be used in specialized environments where accurate estimates of running time are available.

With Shortest Remaining Time First (SRTF), the process with the smallest amount of time remaining until completion is selected to execute. Since the currently executing process is the one with the shortest amount of time remaining by definition, and since that time should only reduce as execution progresses, processes will always run until they complete or a new process is added that requires a smaller amount of time.

**Implmentation**

1 Find Waiting Time  
Traverse until all process gets completely executed.

* Find process with minimum remaining time at every single time lap.
* Reduce its time by 1.
* Check if its remaining time becomes 0
* Increment the counter of process completion.
* Completion time of *current process = current\_time +1;*
* Calculate waiting time for each completed process. *wt[i]= Completion time - arrival\_time-burst\_time*
* Increment time lap by one.  
  2 Find turnaround time (*waiting\_time+burst\_time*).

**YOU MUST WRITE A SOLUTION FOR findWaitingTimeSJF()**

**Round Robin (RR)**

Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way.

It is simple, easy to implement, and starvation-free as all processes get fair share of CPU. One of the most commonly used technique in CPU scheduling as a core. It is preemptive as processes are assigned CPU only for a fixed slice of time at most. The disadvantage of it is more overhead of context switching.

Completion Time: Time at which process completes its execution.

Turn Around Time: Time Difference between completion time and arrival time. *Turn Around Time = Completion Time – Arrival Time*

Waiting Time: Time Difference between turn around time and burst time. *Waiting Time = Turn Around Time – Burst Time*

For this lab we have assumed arrival times as 0, so turn around and completion times are same.

The tricky part is to compute waiting times. Once waiting times are computed, turn around times can be quickly computed.

**Implementation**

1 Find Waiting Time

* Create an array *rem\_bt[]* to keep track of remaining burst time of processes. This array is initially a copy of *plist[].bt* (all processes burst times)
* Store waiting times of processes in plist[].wt. Initialize this array as 0.
* Initialize time : t = 0
* Keep traversing the all processes while all processes are not done. Do following for i’th process if it is not done yet.
  + If rem\_bt[i] > quantum  
    (i) t = t + quantum  
    (ii) bt\_rem[i] -= quantum;
  + Else // Last cycle for this process  
    (i) t = t + bt\_rem[i];  
    (ii) plist[i].wt = t - plist[i].bt  
    (ii) bt\_rem[i] = 0; // This process is over

2 Find Turn Around Time  
Once we have waiting times, we can compute turn around time *tat* of a process as sum of waiting and burst times, i.e., *plist[i].wt + plist[i].bt*

**TESTING**

When testing your solution, use the reference implemtation *schedsim\_ref* and the input files (i.e. input0.txt, input1.txt, and input2,txt).