**CPU Scheduling Simulation**

One of the most important jobs of a modern operating system is managing the various processes in the system. The goal of this project is to enhance your understanding of process management and your programming skills by implementing three scheduling policies:

1. First Come First Serverd (FCFS)
2. Round Robin (RR)
3. Shortest Remaining Burst First (SRBF)

The input to your program is a text file containing information about the system processes. There is only one CPU. Each line in the text file has three pieces of information:

1. A string (label) that uniquely identifies a process
2. The arrival time for the process
3. The CPU burst time for the process

For example, the line

P2 7 11

states that the process P2 arrives at time 7 and requires 11 time units to run to completion. For simplicity, we assume assume that each process consists of a single CPU burst (no I/O bursts), and processes are listed in the input file in order of their arrival time. For policy RR, a *quantum* time will be required, which will be an input ( in the example below, the *quantum* time input is 2).

**Sample Execution**

Assume that you have created a file process.txt with the following data:

P0 0 3

P1 1 6

P2 5 4

P3 7 3

If you invoke your scheduler (executable cpu-scheduler) using the command

cpu-scheduler process.txt

then your program should have a behavior similar to the following:

-------------------------------------------------

CPU Scheduling Simulation

-------------------------------------------------

Select the scheduling algorithm [1,2,3 or 4]:

1. First Come First Served (FCFS)

2. Round Robin (RR)

3. SRBF

4. Exit

1

-------------------------------------------------

First Come First Served Scheduling

-------------------------------------------------

[0-3] P0 running

[3-9] P1 running

[9-13] P2 running

[13-16] P3 running

Turnaround times: P0[3], P1[8], P2[8], P3[9]

Wait times: P0[0], P1[2], P2[4], P3[6]

Average turnaround time: 7.00

Average wait time: 3.00

Hit any key to continue ...

-------------------------------------------------

CPU Scheduling Simulation

-------------------------------------------------

Select the scheduling algorithm [1,2,3 or 4]:

1. First Come First Served (FCFS)

2. Round Robin (RR)

3. SRBF

4. Exit

2

Enter the time quantum: 2

-------------------------------------------------

Round Robin Scheduling

-------------------------------------------------

[0-2] P0 running

[2-4] P1 running

[4-5] P0 running

[5-7] P1 running

[7-9] P2 running

[9-11] P3 running

[11-13] P1 running

[13-15] P2 running

[15-16] P3 running

Turnaround times: P0[5], P1[12], P2[10], P3[9]

Wait times: P0[2], P1[6], P2[6], P3[6]

Average turnaround time: 9.00

Average wait time: 5.00

Hit any key to continue ...

**Project Specifications**

In your AWS Linux home directory, create a new directory called lab3 and a C program file called cpu-scheduler.c. Use *incremental programming* techniques to develop the code for your scheduler in consecutive stages:

1. Write a function
2. void ReadProcessTable(char \* filename);

that reads the input file into a global data structure. This structure may be an array of processes, with each entry of the following type:

typedef struct {

char \* name;

int arrival;

int cpuburst;

int turnaround;

int wait;

} Process;

Process processtable[MAX\_PROCESS];

Note that you will need to allocate memory dynamically for name, since no static memory has been allocated by the definition above. Use [fopen](http://www.tutorialspoint.com/c_standard_library/c_function_fopen.htm), [fscanf](http://www.tutorialspoint.com/c_standard_library/c_function_fscanf.htm) and [fclose](http://www.tutorialspoint.com/c_standard_library/c_function_fclose.htm) to handle the input file.

1. Once your code for the first step compiles and runs without errors, write a function
2. void PrintProcessTable();

that simply outputs the data stored in the global processtable structure filled in by the ReadProcessTable call. Verify that the output is identical to the data stored in the input file.

1. Next implement a function
2. void FCFS();

You will need to keep track of the current time by means of a variable (global or local)

int current\_time;

Your FCFS function should initialize this variable to zero and advance it to the time of the next event in each iteration through the process list. This function should only update the processtable data structure (entries turnaround and wait). To print out the statistics on the turnaround times and wait times, write another function called

void PrintStatistics();

This function should work independent on the scheduling algorithm used (FCFS or RR).

1. Once you have FCFS working properly, move on to implementing the Round Robin scheduling method

void RR(int quantum);

This function must iterate through the list of processes multiple times. In each round (loop iteration), it schedules all active processes to run quantum time units or less, depending on the cpu burst time left for each process ( the quantum is an input – see above). To keep track of the remaining cpu burst for each process, you could use a local copy of the cpu bursts

int cpuburstcopy[MAX\_PROCESS];

and update it in each round. The local copy is necessary so that the global structure processtable’s cpuburst and arrival fields remains unaltered (and could be used by FCFS, or SRBF for instance). In addition to this structure, you might also need to maintain an index in the process table, say first, that marks the entry for the first process with positive (non-zero) remaining cpu burst time.

Alternatively, you could use a queue to keep the processes in the right order for getting the CPU next.

1. Implement SRBF in a similar manner

void SRBF();

In this policy, when a new process arrives, the scheduler needs to be called to decide if the newly arrived process should get the CPU as its burst time is less than the remaining burst time of the currently running process. Work out the data structures yourself.

NOTE: You are free to use any other set of data structures and methods to implement the simulation. The above is only a suggestion. If you find the suggestions are incomplete or confusing, devise your methods.

**Submission Instructions**

Descriptions of your code should should be integrated into your code as comments.

Leave the source code for all exercises in your directory lab3. Do not make any changes to these files after the due date for this assignment.

Execute the program with 4 different inputs, including the one given as an example above. Keep the inputs in different files, process1.txt, process2.txt, etc. Your program should run with the given input file, for example: cpu-scheduler process3.txt.

Note that with one input set, you should run all the three policies one by one so that you can compare how the three policies perform on the same input.

Submit a results.txt file giving an analysis of each run and comparing the performance of each policy with that run.

DUE DATE : 10 /11 February 2021