# CS 3733 Operating Systems: Assignment 5 - new

---

You are required to submit your work through UTSA Learn (aka. Blackboard or BB Learn) !!! No e-mail submissions !!!

!!!! Please carefully check the DUE DATE on BB Learn !!!!

!!!! NO LATE SUBMISSION WILL BE ACCEPTED !!!

---

Create a directory called `assign5` for this assignment under `cs3733`.
Write all your programs under `assign5`.

---

### Objective

- Learn and practice CPU scheduling algorithms by implementing them
- Learn and practice process/thread synchronization mechanisms by implementing them
- Practice system calls and library functions

---

### Description

In this homework, you are asked to implement a **multithreaded** program that will allow us to measure the performance *(i.e., CPU utilization, Throughput, Turnaround time, and Waiting time in Ready Queue)* of the four basic CPU scheduling algorithms *(namely, FIFO, SJF, PR, and RR)*. Your program will be simulating the processes whose priority, sequence of CPU burst time(ms) and I/O burst time(ms) will be given in an input file.

*Assume that all scheduling algorithms except RR will be non-preemptive, and all scheduling algorithms except PR will ignore process priorities (i.e., all processes have the same priority in FIFO, SJF and RR). Also assume that there is only one IO device and all IO requests will be served using that device in a FIFO manner.*

---

Your program will take the name of the scheduling algorithm, related parameters (if any), and an input file name from command line. Here how your program should be executed:

```
prog -alg [FIFO|SJF|PR|RR] [-quantum [integer(ms)]] -input [file name]
```

The output of your program will be as follows:

```
     Input File Name      : file name

     CPU Scheduling Alg   : FIFO|SJF|PR|RR (quantum)

     CPU utilization      : ....

     Throughput           : ....

     Turnaround time      : ....

     Waiting time         : ....
```

---

The input file is formatted such that each line starts with `proc`, `sleep`, `stop` keywords.

Following `proc`, there will be a sequence of integer numbers: the first one represents the priority (1: lowest, ..., 5: normal, ..., 10: highest). The remaining ones represent CPU burst and I/O burst times (ms) in an alternating manner. The last number will be the last CPU burst time after which that process exits.

Following `sleep`, there will be an integer number representing the time (ms) after which there will be another process. So one of the threads in your program would be responsible for processing this file as follows. As long as it reads `proc`, it will create a new process and put it in a ready queue (clearly this process is not an actual one, it will be just a simple data structure (similar to PCB) that contains the given priority and the sequence of CPU burst and I/O burst times, and other fields). When this thread reads `sleep x`, it will sleep x ms and then try to read new processes from the file.

Upon reading `stop`, this thread will quit.

---

Here is a sample input file (copy/paste this to create an input1.txt you can create other similar files too):

---

```
proc  1 10 20 10 50 20 40 10

proc  1 50 10 30 20 40

sleep 50

proc  2 20 50 20

stop
```

---

In this program you need at least three threads:

As described above, one thread (say FileRead thread) will read the above file, create a process (not a real process just a data structure representing the characteristic of the process that will be simulated), and puts it in a **ready queue**.

CPU scheduler thread will check ready queue; if there is a process, it will pick one according to the scheduling algorithm from ready queue and hold CPU resource for the given CPU burst time (or for quantum time if the scheduling algorithm is RR). That means CPU thread will simply **sleep** for the given CPU burst time to simulate the process. Then it will release CPU resource and put this process into **IO queue** (or ready queue if RR is used) or just terminate if this is the last CPU burst. Then CPU scheduler thread will check ready queue again and repeat the same for the next process.

I/O system thread will check IO queue; if there is a process, it will hold IO device for the given IO burst time. That is IO thread will sleep for the given IO burst time. It then puts this process back into **ready queue**. Finally it will check IO queue and repeat the same ....

---

Basically you will have at least the above mentioned three threads and the main one. These threads need to be synchronized as they cooperate to collect data for performance measures and share data through ready and IO queues. Main thread will wait until the file reading thread is done and the ready queue and IO queue are empty, then it will print the performance evaluation results and terminate the program...

You are free to design and implement this program in C/C++ or Java (but if you use Java, DO NOT use the *"synchronized"* methods or high-level thread-safe Java classes. Instead use some form of semaphore or basic synchronization mechanisms in Java. Also `synchronized(obj){ }` structure is OK to use for protecting critical sections in your methods).

Also you can use any data structures in different parts of your program and share them along with new variables; but when it comes to maintaining the list of processes in ready queue or IO queue, we would like you to use **double linked list,** as this is the case in many practical OS.

---

**Grading:** This is a 60-point homework.

First make sure your program works for FIFO CPU scheduling. This will be **20** points.

Then you can add/test the others: SJF (**10** points), PR (**10** points) and RR (**10** points).

Write a 2-3 page **report** (**10** points) to describe your design choices at the high level and your results.

Do all your work under a directory **assign5**, which should include your source codes, instructions to compile/execute, and some output files showing your test results etc...

Zip **assign5** as **abc123-assign5.zip** and submit it through BB Learn

---

**Submission**

You must submit your work using Blackboard Learn and respect the following rules:

All assignments must be submitted as either a zip file unless it is a single pdf file.

Assignments must include all source code.

Assignments must include an output.txt file which demonstrates the final test output run by the student.

If your assignment does not run/compile, the output.txt file should include an explanation of what was accomplished, what the error message was that prevented the student from finishing the assignment and what the student BELIEVES to be the underlying cause of the error.