NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES ISLAMABAD

OPERATING SYSTEMS Spring, 2020

ASSIGNMENT 02

Due Date: 9:00 AM 1st, April 2020

Instructions

- Zero marks will be awarded to the students involved in plagiarism.
- All the submissions will be done on slate.
- You have to submit .c/.cpp file named as 5_state_model_simulator along with executable file call in simulator program. Add 5,6 different output screenshots (e.g., Output1, Output2…), each showing processes on different states and their final output. Also, you need to add both files .txt used in this simulator. Naming convention has to be followed strictly. You need to submit zip folder named as ROLL_NUM_SEC (e.g., 18i-1234_A).
- Each part will carry different marks.
- Read the complete instructions given in each part as all of them are related with each other. In case of any query comment below on classroom preferable or you can email to fasial.cheema@nu.edu.pk shehr.bano@.nu.edu.pk or shahnila.rahim@nu.edu.pk.
- Be prepared for viva or anything else after the submission of assignment for two weeks.

CPU Scheduling Simulator

Question 1.

Write a Program in **C/C++** for below given scenario where it has different Operating system scheduling algorithms using inter-process communication primitives. For this task you have to make use of **Linux system calls** such as **fork, exec, pipes** (named/unnamed pipes depending on the requirement of communication), **dup/dup2** as well as **pthread library**. This program will require writing in files every time when it need to dispatch a process from a state.

Note: For reading/writing to files, you cannot use fstream objects (instead you have to use lower level system calls to achieve the same functionality).

Let's consider there are **n** number of processes named as **Proc = 0, 1, …. , n.** Each process will be in new state at first and then move through different states of given model.

For this scheduling task, you have to simulate the given 5 state process model as shown below.

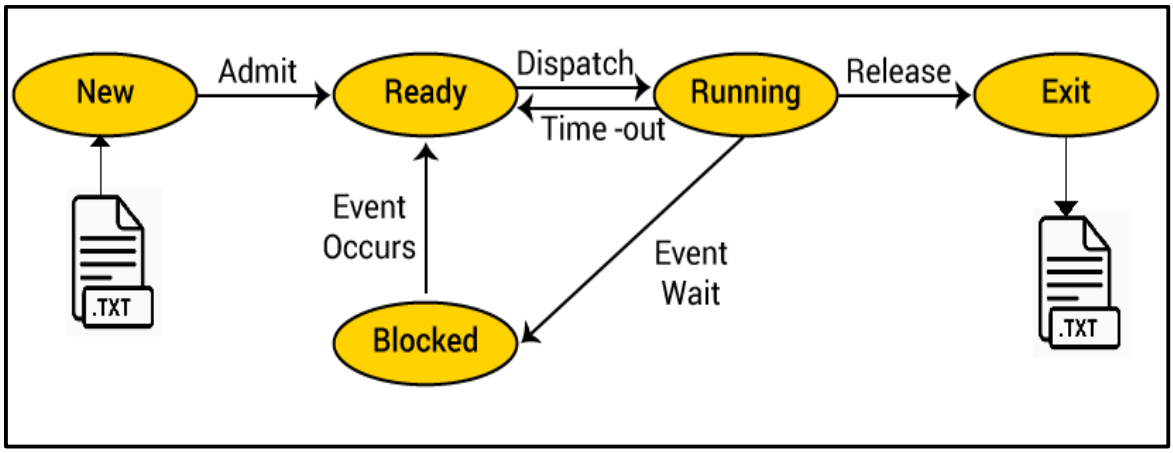

Figure 1: Process 5-State Model

PART A (Fork & EXEC Implementation)

1. Each of the process states (e.g. New, Ready, Running etc.) will serve as an individual process in the simulator program.
2. Each process will be created using fork system call and then call exec for each state implementation. Before calling exec each should create pipes where required and are able to serve them for reading and writing instead of writing to the terminal.
3. You have to establish the inter process communication between different processes as shown in the figure above.

Note: The default behavior of pipes is that read is blocking once write end is opened. You need to handle this in your simulator program at every point where you will create and inherit a pipe(s). Below are the two solution to handle this.

a) To make the pipe reading unblocking, you have to set file status flags. The following link will guide you (https://www.geeksforgeeks.org/non-blocking-io-with-pipes-in-c/). You can also refer the man page of pipe from given link. http://man7.org/linux/man-pages/man7/pipe.7.html

b) You can save the copy of descriptors using dup and pass it as exec parameters to new image process.

PART B (States Implementation)

4. The **New process** will read from a text file **processes.txt**. The processes file contains information about which scheduling algorithm to run and incoming processes (known as **Procs**) – the sample file is provided below. The intial row(s) contains the choice of scheduling algorithm whereas the next rows contain information about **Procs** (i.e. Arrival time, Burst time and Priority). The priority in case of non-priority algorithm will be 0 for all **Procs**. The New process will create a data structure for each **Proc** holding process information. Each **Proc** will also hold its execution information (waiting time, completion time etc.). See the below example of process.txt file having multiple processes information.

Sample format of FCFS/SJF/STRF
FCFS
Proc1
Arrival time
Burst time
Proc2
Arrival time
Burst time

Sample format of RR
RR
Quantum
Proc1
Arrival time
Burst time
Proc2
Arrival time
Burst time
.
.
.

Figure 2: Format of Processes.txt

FCFS **Proc1** 0 5 **Proc2** 0 8	SJF **Proc1** 0 5 **Proc2** 0 8
RR 5 **Proc1** 3 9 **Proc2** 7 10 . . .	SRTF **Proc1** 7 15 **Proc2** 3 7 . . .

Figure 3: Processes.txt

Note: Processes.txt will have different samples of process with respect to their scheduling algorithm.

5. The **Ready process** is the core process of the simulator. The incoming **Procs** from *New / Blocked / Running* processes will enter the ready queue maintained by the Ready process for scheduling. The queue can be in form of a simple queue, multiple queues and min heap (depending on choice of scheduling). The scheduler will dispatch the processes to the Running processes as per the scheduling policy selected. The Ready process will wait from input from *New, Blocked and Running* processes, as well notification from Running process to send next Proc for execution.

6. The **Running process** mimics a uniprocessor CPU. Initially the time in CPU is 0. For simplicity of time management assume 1 tick of time to be simulated by sleep (1). Scheduler increments time until the time = arrival time of the first **Proc**. There will be a random binary number generated after 5 ticks. If binary number is 1 the current **Proc** continues its execution, in case of binary 0 the **Proc** is sent to blocked queue for I/O operation. In case of preemptive scheduling, **Proc** will be sent back to Ready queue upon completion of its time slice. In case, the **Proc** completes its execution, it will be sent to the Exit process, and Ready process will be notified to dispatch next **Proc** for execution.
7. The **Blocked process**s will maintain a blocked queue. Like the Running process, assume 1 tick of time to be simulated by sleep (1). Each **Proc** in the blocked queue will be blocked for 15 – 25 ticks (randomly decided). Upon completion of the ticks, the **Proc** will be sent back to the Ready queue.
8. The **Exit process**s will accept all the completed **Procs**. It will output different **Proc** statistics to a text file named *processes_stats.txt* and also display about completion of that process on output screen. For each completed **Proc**, it will print Gantt chart with information stating Arrival time, Burst time, Turnaround time and Waiting time. After every 30 ticks, the Exit process will append the cumulative stats to file containing Throughput, Average Turn around and Average Waiting time.
9. Implement following scheduling algorithms. You have to use proper data structures for implementing them.
    a. **FCFS algorithm using queues.**
    b. **RR algorithm using queues.**
    c. **SJF algorithm using heaps.**
    d. **SRTF algorithm using heaps.**

PART C (Multithreading Implementation)

10. At the moment the **Ready process** has to wait for three different events i.e. incoming **Procs** from New, Event completed **Procs** from Blocked and Timed-out **Procs** from Running. This will decrease the efficiency of the scheduler, as all three queues will be needed to be checked at regular intervals for incoming **Procs**.. Implement multi-threading in Ready process such that waiting activities are separated from core tasks.

    Hint: Three threads should be implemented here and there synchronization will be depending on the scheduling algorithm you have read implemented.

11. At the moment, you implemented just one Blocked queue. Implement three different threads to hold different queues e.g. Input queue, Output queue, printer queue. Each incoming Proc from Running will have its blocked type (randomly generated) and will be placed in the concerned queue.

    You can use multithreading in any part of your program where is improves the efficiency of each state.