Indian Institute of Technology Jodhpur
Operating Systems Lab (CSL 3030)
Assignment 4

Dated 13th September, 2022 **Total marks: 40**

In this assignment you will be implementing a virtual scheduler on top of existing Linux scheduler (you wont be modifying this). The virtual scheduler is supposed to choose one process from the 'Ready Queue' as per the policy imposed by the scheduling algorithm in execution. Next it will send a 'notify' signal to the selected process so that it can (re)start execution and a 'suspend' signal to all other processes in the Ready Queue so that they can go back in the waiting state until the next time it gets picked up by the scheduler. The scheduler will terminate after all the processes terminate and the Ready Queue is empty.
After setting up the simulation environment, you will be performing a comparative evaluation among the various scheduling algorithms.

Process Classification: You will be generating two types of processes: CPU bound and I/O bound. CPU bound processes will be iterating more number of times (higher 'Round' value), requiring more amount of CPU cycles and occasionally go to sleep (i.e. perform I/O) for a short duration. For I/O bound process, it goes to sleep (perform I/O) more frequently and for longer duration with a lower value of 'Round' associated with.

Process Generation: Implement a Generator whose responsibility would be to generate N processes at T interval. It randomly decides the status (CPU bound or I/O-bound for each process) and accordingly sets the following parameters:

1. Round: The number of iterations a process will run which signifies the computation duration of a process.
2. Priority: In a range of 1-10, lower number indicates higher priority
3. Sleeptime probability: I/O request is simulated via sleep(). The Sleep probability specifies the probability of I/O request (i.e. sleep) in each iteration.
4. Sleeptime: The sleeptime indicates the I/O service duration. For a specific process, all I/O requests are of same duration.

Process States: Each process can have four states: Ready, Running, Waiting and Terminated. The state transition occur in following ways:

1. **Ready→ Running:** Initially, after creation, the process sends its process id and priority information to the scheduler. The scheduler adds the process at the rear of the `Ready Queue' which makes the process runnable.
   At the same time, the process invokes pause() and waits in the 'Ready Queue'. Once the scheduler schedules the process, it sends the 'notify' signal and the process resumes execution. Again, at any point of time, if the process receives 'suspend' signal from the scheduler, the process gets suspended (using pause()) and it waits for the `notify' signal.
2. **Running→ Waiting:** During execution, a process may request for I/O. This is simulated using sleep(). In each iteration, a process may request for i/o with 'Sleep Probability' for the service duration of 'Sleep Time'. The process informs scheduler before requesting I/O and after completion of I/O. If the process requests I/O, the scheduler removes it from the 'Ready Queue' and once it completes the I/O, the scheduler adds the process at the rear of the 'Ready Queue'. Hence, the scheduler removes the process from the 'Ready Queue' before sleep() and reinserts the process at the rear end of the 'Ready Queue' after sleep(), accordingly.
3. **Running →Terminated** : When a process completes its rounds and exits the loop, it informs the scheduler and the scheduler removes it from the 'Ready Queue'.

Implementation :
Create two programs: generator.c and process.c to spawn all the processes and run individual process activities,

respectively. Let the generator create N processes and then use system call with required parameters to execute individual process on separate terminal. (N processes in N terminals).

The Scheduler:

The scheduler has two major responsibilities: implementing the scheduling policy and managing the ready queue.
1. **Scheduling:** The scheduling algorithm selects one process from among multiple processes in the Ready Queue to be executed next as per the scheduling policy mandated by the algorithm in execution. In this assignment, you have to implement two scheduling algorithms (a) *regular round robin (RR) scheduling* and (b) *priority based round robin (PRR) scheduling*. The scheduler takes the scheduling decision from the current ready queue once the time quantum expires or a process requests for an I/O or terminates before the time quantum expires.
2. **Ready Queue Management :** Based on the scheduling decision, scheduler sends a 'notify' signal to the process that is scheduled next and sends a 'suspend' signal to all other processes.

Implementation:
You will be implementing scheduler.c that creates a Ready-Queue to contain the process ID of the runnable processes. This queue is private to the scheduler.
You can simulate the timer and time quantum of the RR scheduling using a for() loop with fixed iteration. It can come out of the loop if one of the following three conditions occurs:
1. The process that is running, can decide to go for an I/O (break from the loop).
2. The process can terminate before the time quantum expires (break from the loop).
3. The time quantum expires (normal exit from loop).

So the scheduler should continuously sense 'I/O request' or 'terminate' signal from the running process in the loop for detecting the first two cases and should break out of the loop if any of those occurs. Once time-quantum expires normally (case 3), the scheduler sends a 'suspend' signal to the running process. Next, it decides which process from the 'Ready Queue' is to be scheduled next based on the scheduling algorithm and sends a 'notify' signal to the specific process.

You are going to implement the Scheduler process using two dedicated threads 'Adder' and 'Remover' to add and remove PIDs into/from the Ready Queue, respectively. There will be total four threads of control in Scheduler program. The main thread is responsible to perform the stated scheduling policy and sending the 'notify'/'suspend' signal to the correct process time to time. The 'Adder' thread will listen to the incoming requests from the processes so that the corresponding PID can be added to the Ready Queue. Whereas the 'Remover' thread can be implemented to listen to the incoming requests from the processes that need to terminate or perform I/O. Adder and Remover should sync up with each other to manage the Ready Queue consistently and coordinate with the main scheduler thread as well.

During the execution of one process, a set of other processes may complete I/O and inform the scheduler. The 'Adder' thread will be responsible to maintain the order of such arriving requests and reflect it while adding entries into the 'Ready Queue'. Once the time-quantum of the running process expires (i.e. it comes out of the loop), the scheduler via its 'Adder' thread adds all those processes in the 'Ready Queue'. In this way, the scheduler updates the current 'Ready Queue', based on which the scheduler decision will be taken. The scheduler terminates when it receives N terminating signals from N processes via the 'Remover' thread, which keeps a count on the number of received requests.

Inter-Process Communication:
For inter-process communication, develop the scheduler program as multi-threaded. You can use socket programming to perform IPC in a multi-threaded environment.

Process communicating with scheduler:
1. After creation: It sends the process-id & priority information to the scheduler. In response, the Adder thread in

scheduler adds the process to the Ready Queue.

2. Before requesting I/O: It sends 'I/O Request' signal to the scheduler and in response the Remover thread in scheduler removes the process from 'Ready Queue'

3. Completion of I/O: It sends 'I/O Completion' message to the message queue. In response, the Adder thread in scheduler adds it back to the 'Ready Queue'.

4. Terminate: It sends 'Terminate' signal to the scheduler. The Remover thread in scheduler increments a count and checks whether all the N processes have terminated or not.

Scheduler communicating with process:
1. Suspend a process: The scheduler sends 'suspend' signal to the running process. In response, the signal handler executes pause().

2. Resume a process: The scheduler sends a 'notify' signal to a process in pause(). In response, the process resumes execution.

Hint:

- Using pause() system-call a process can wait for a particular signal.
- Total 4 signals are to be handled in this assignment - 'Suspend' & 'Notify' for scheduler, 'I/O Request' & 'Terminate' for processes.

Process Parameters :

- You have to create 4 processes. First two will be CPU bound and last two will be I/O bound. The time-interval T will be 1 second.
- For CPU Bound processes the Priority will be between 1-5, Rounds will be 10000,
  Sleep Time will be 1 second and sleep probability will be 0.3.
- For I/O Bound processes the Priority will between 6-10, No. of Iterations will be 4000, SleepTime will be 3 second and sleep probability will be 0.7.
- For similar priority processes of same type, a tie can be broken considering the FIFO order in the Ready Queue.

Comparative Analysis :
You have to implement round robin (RR) scheduling and priority based round robin (PRR) scheduling.
For both the algorithms use time quantum = 1000 iterations and 2000 iterations.
First you should execute scheduler.c in a terminal. It will take the scheduling algorithm type as a command line argument.
1. 'RR' : Regular round robin
2. 'PRR' : Priority based round robin.
The scheduler should sense the incoming signals to understand that the processes have been created.
Then you should run generator.c in another terminal which will create 4 processes described above. One xterm should be created for each process and the process that is scheduled should print "PID:<PID>, loop-counter" when it runs its iterations. The other processes should wait at that time and should not print anything in their respective xterms.

Upon finishing, a file (result.txt) should be created which will have the
1. Response time.
2. Total waiting time.
3. Turn-around time.
for each process with their process id. And finally, the file should contain the average of this metrics for all 4

processes.

The scheduler should display the following status in its terminal:
P1 <pid> is running,
P1 <pid> requests I/O
P3 <pid> is running
P4 <pid> completes I/O….

Note: Student should properly display all the output in the terminal as per the instructions.

References:
https://www.cs.uic.edu/~troy/fall99/eecs471/signals.html
https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html

Evaluation Guidelines:

(a) Implementation of Generator and processes [5]
(b) Implementation of Scheduler with both RR and PRR policy [8]
(c) Ready Queue management through thread coordination [6]
(d) IPC implementation using Socket API [6]
(e) Terminal printing for the process activities [5]
(f) Report (result.txt) generation. Make sure to be comprehensive about your results and analysis. [4]
(g) Comparative analysis of the stated performance metrics [6]

Deadline : 21st September, 2022