CS39002: Operating Systems Lab
Spring 2016
Assignment 4
Due on: 15st February 2016

---

1. Virtual scheduler overview:

In this assignment, you have to implement a virtual scheduler. This virtual scheduler runs on top of the existing linux scheduler which you should not change. It consists of two major components: a generator and a scheduler. The generator is responsible for generating the processes at regular interval. The scheduler maintains a 'Ready Queue' which contains the set of runnable processes at any time-instance. It chooses/schedules one of the runnable processes according to the scheduling algorithm for execution. Next it sends a 'notify' signal to only that chosen/scheduled process and 'suspend' signal to all other processes. Depending on the signal ('notify'/'suspend'') received from the scheduler, the processes can either (re)start processing or wait until it's scheduled again. Once all the processes terminate, the scheduler also terminate.

After building this simulation framework, you have to evaluate different scheduling strategies and evaluate their performance in terms of average response time, average waiting time and average turnaround time.

In this assignment, processes and scheduler communicate among themselves in various different ways which are discussed in detail in the 'Communication Protocol' section. Hence, we are not repeating that throughout the assignment.

2. Process classification:

You are expected to generate two types of processes - CPU bound and I/O bound. CPU bound processes do more computation (i.e. execute more iterations) and goes for I/O (i.e. execute sleep) occasionally. For the I/O bound processes, it's other way around.

3. Generator:

The task of the 'generator' is to generate N number of processes at t seconds intervals. The 'generator' decides whether a process will be CPU bound or I/O bound. Accordingly, it passes the following parameters to the newly generated process

1. Number of iterations (NOI) - The number of iterations the process will run. This determines the computation duration of the process.
2. Priority – Specifies the priority of the process.
3. Sleep Probability – The I/O request is simulated visa sleep(). The Sleep probability specifies the probability of I/O request (i.e. sleep) in each iteration.
4. SleepTime - The sleep time emulates the I/O service duration. For a specific process, all I/O s are assumed to be of same duration.

3.1. Process states:

Each process can be in four states - 'Ready', 'Running', 'Waiting' & 'Terminated'. The state-transitions occur in the following ways,

3.1.1. Ready to Running :
Initially, after creation, the process sends its process id and priority information to the scheduler. Next, the scheduler adds the process at the end of its `Ready Queue' which indicates the process is 'Ready'.
Side by side, 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.

3.1.2. Running to 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.

3.1.3. Running to Terminated :
When a process completes its iterations and exits the loop, it informs the scheduler and the scheduler removes it from the 'Ready Queue'.

3.2. Implementation :
You have to implement two main programs in this part: gen.c and process.c. gen.c is the generator code which spawns all the processes and process.c consists of individual process codes. Once the generator creates N processes, N different terminals (xterms) should be created. In each of this terminals, the generator uses system() call with required parameters to execute individual processes (process.c).

4. Scheduler :
There are two major functionalities which implements a scheduler

1. **Scheduling protocol:** Based on processes that are available in the ready queue, the scheduling algorithm selects the process to be scheduled next. In this assignment, you have to implement two scheduling algorithms (a) *regular round robin scheduling* and (b) *priority based round robin scheduling*. The scheduler takes the scheduling decision from the current ready queue once the time quantum expires or a process requests for I/O or terminates before the time quantum expires.
2. **Stopping or Resuming a Process :** 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.

4.1 Implementation:
You have to implement sched.c for the scheduler process. It creates a Ready-Queue to contain process ID of the runnable processes. This queue is private to the scheduler.

You can simulate the timer and time quantum of the Round-Robin 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 ‘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.

During the execution of one process, a set of other processes may complete I/O and inform the scheduler (via message queue). You can handle this event in the following way. Once the time-quantum of the running process expires (i.e. it comes out of the loop), the scheduler (reads the message queue and) 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.

5. Communication protocol:

The scheduler creates a message queue with a key and shares the queue with all the generated processes.

5.1 Process communicating with scheduler:

1. After creation: Using message queue it sends the process-id & priority information to the scheduler. In response, the 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 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 scheduler adds it back to the ‘Ready Queue’.
4. terminate: It sends ‘Terminate’ signal to the scheduler. The scheduler increments a count and checks whether all the N processes have terminated or not.

5.2 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.

6. 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 10, No. of Iterations will be 10000, Sleep Time will be 1 second and sleep probability will be 0.3.
- For I/O Bound processes the Priority will be 5, No. of Iterations will be 4000, Sleep Time will be 3 second and sleep probability will be 0.7.

7. Evaluation :

You have to implement round robin scheduling and priority based round robin scheduling. For both the algorithms use time quantum = 1000 iterations and 2000 iterations.

First you should execute sched.c in a terminal. It will take the scheduling algorithm type as a command line argument.

1. 'RR' : Regular round robin
2. 'PR' : Priority based round robin.

The scheduler should create the message queue and continuously senses the message queue to check if the first process has been created.
Then you should run gen.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

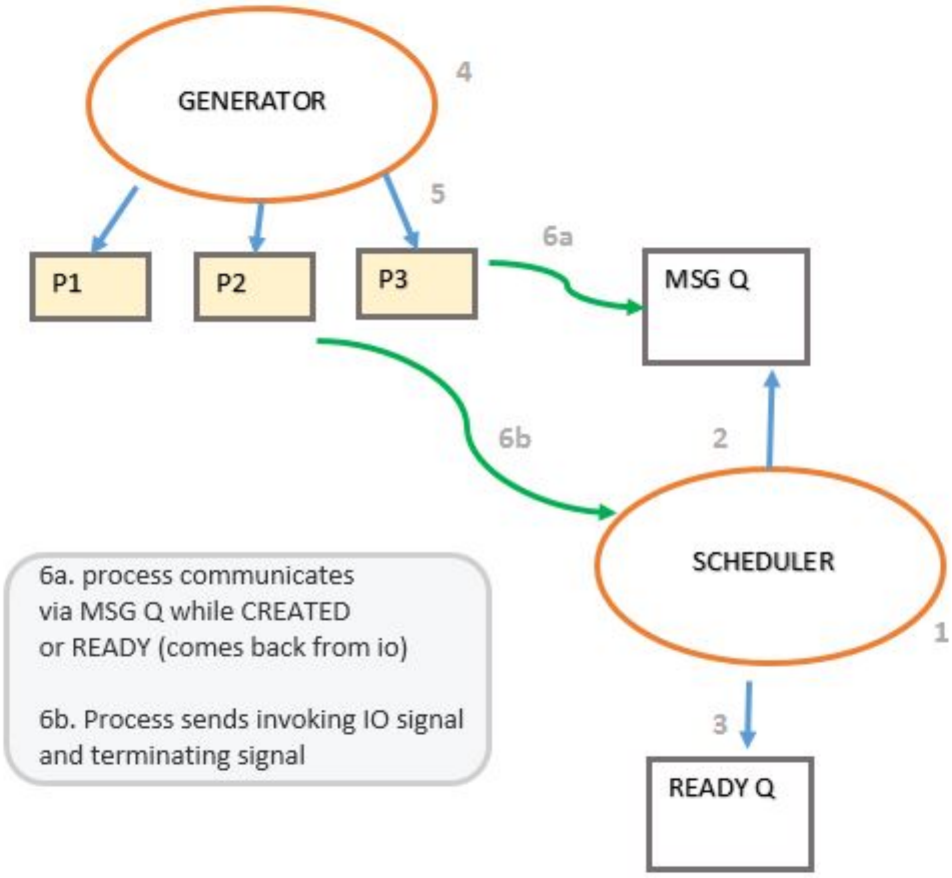
1. Response time.
2. Total waiting time.
3. Turn-around time.

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….

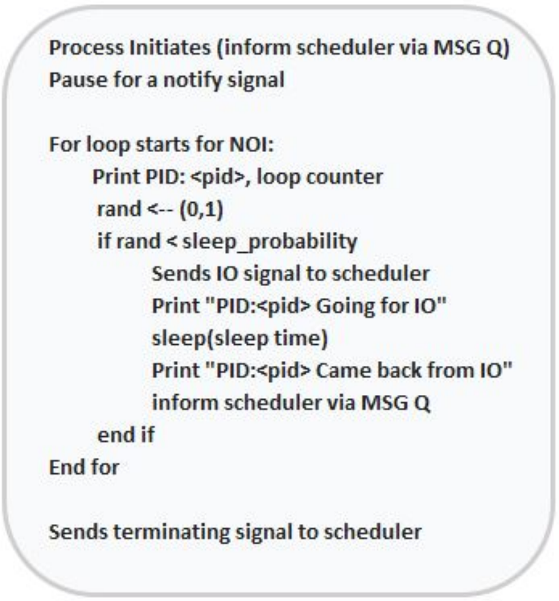
Note: Student should properly display all the output in the terminal as per the instructions.

8. Scheduling System and Process Block Diagrams:



6a. process communicates
via MSG Q while CREATED
or READY (comes back from io)

6b. Process sends invoking IO signal
and terminating signal



```
Process Initiates (inform scheduler via MSG Q)
Pause for a notify signal

For loop starts for NOI:
      Print PID: <pid>, loop counter
      rand <-- (0,1)
      if rand < sleep_probability
            Sends IO signal to scheduler
            Print "PID:<pid> Going for IO"
            sleep(sleep time)
            Print "PID:<pid> Came back from IO"
            inform scheduler via MSG Q
      end if
End for

Sends terminating signal to scheduler
```

References:

Message queue: https://www.cs.cf.ac.uk/Dave/C/node25.html