Cairo University
Faculty of Engineering
Computer Engineering Department

Operating Systems
CMP N 303

OS Scheduler

Objectives

1. Evaluating different scheduling algorithms.
2. Practice the use of IPC techniques.
3. Best usage of algorithms, and data structures.

Introduction

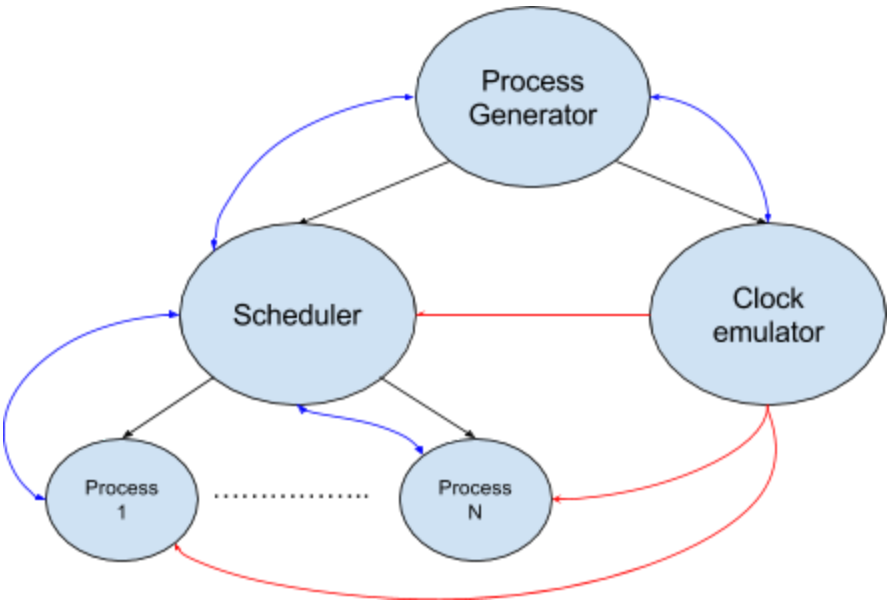
A CPU scheduler determines an order for the execution of its scheduled processes; it decides which process will run according to a certain **data structure** that keeps track of the processes in the system and their status.

A process after creation has one of the three states: Running, Ready, Blocked (doing I/O , using other resources than CPU or waiting on unavailable resource).

A bad scheduler will make a very bad Operating System, so your scheduler should be as much optimized as possible in terms of memory and time usage.

System Description

Consider a Computer with 1-CPU and infinite memory, it is required to make a scheduler with its complementary components as the following



Parent-Child relation
One Way Communication
Two way Communication

Part 1: Process generator (Objective: For simulation & IPC)

Code file: processGenerator.cpp

Process generator: responsible for

- Reading the input files (check the input/output)
- Ask the user about the chosen scheduling Algorithm and its parameters if exists.
- Initiate and create Scheduler and Clock processes.
- Creating a data structure for process and provide it with its parameters
- Send the information to the scheduler **at the appropriate time** (only when a process arrives) so that it will be put it in its turn.
- At the end it Clear IPC Resources.

Part 2: Clock (Objective: For simulation & IPC)

Code file: clk.cpp

the clock module is used to emulate a integer time clock

This module is already built for you.

Part 3 : Scheduler (Objective: OS Design, IPC)

Code file: scheduler.cpp

The scheduler is the core of your work, it should keep track of the processes and their states
And it decides – based on the used algorithm - which process will run & for how long.

You are required to implement 3 Algorithms

1. non-preemptive HPF
2. Shortest Remaining time Next
3. Round Robin

The scheduling algorithm only works on the processes in **the ready queue.** (Process that already arrived).

The Scheduler should be able to

- Start a new process according to the scheduling algorithm.(Fork it and give its parameters)
- Switch between two processes according to the scheduling algorithm. (Stop the old process and save its state and start/resume another one)
- At anytime, it should have a process control Block that keeps track of the state of the processes in the system (running/waiting/execution time/ remaining time/ waiting time/ ...etc)
- If the scheduler is notified that a process finished, it delete its data.
    - When a process finishes it should notify the scheduler on termination, the scheduler DOESN'T terminate the process.

For each algorithm you should report:

- CPU utilization.
- average weighted turnaround time.

- average waiting time.
- standard deviation for average weighted turnaround time.

The Scheduler Generates two files: (check the input/output)
- Scheduler.log
- Scheduler.perf

Part 4 : process (Objectives: simulation, IPC)

Code file: process.cpp

The process should act as if runs in the execution time,
the process might or might not communicate with the clock, based on your design

Part 5 : Input / output (Objectives: simulation, evaluation OS Design)

Input:

- You will have a folder called "processes.txt" : it will contains a line for each process with the following format

processes.txt format

```
#id   arrival   runtime   priority
1   1   6   5
2   3   3   3
```

- Comments are added as lines beginning with "#" and should be ignored
- Different Fields are separated with single space
- You can always assume that processes are sorted by their arrival time.
  - Take care 2 or more processes might arrive at the same time.
- you can use the FileGenerator.cpp to generate a random test case.

Output:

scheduler.log format

```
#At time x process y state arr  w total z  remain y wait k
At time 1 process 1 started arr 1 total 6  remain 6 wait 0
At time 3 process 1 stopped arr 1 total 6  remain 4 wait 0
At time 3 process 2 started arr 3 total 3  remain 3  wait 0
At time 6 process 2 finished arr 3 total 3  remain 0 wait 0 TA 3 WTA 1
At time 6 process 1 resumed arr 1 total 6  remain 4 wait 3
At time 10 process 1 finished arr 1 total 6  remain 0 wait 3 TA 10 WTA 1.67
```

- Comments are added as lines beginning with "#" and should be ignored

- Different Fields are separated with **single space**
- Approximate numbers to the nearest 2 decimal places:
  - i.e. 1.666667 ≃ 1.67
  - 1.33333334 ≃ 1.33
- Allowed states: started, resumed, stopped, finished
- Only At finished state it writes the TA & WTA
- If your Algorithm do a lot of processing processes might not start and stop at the same time.
- **You need to stick to format because we automatically compare files**

scheduler.perf format

```
CPU utilization=100%
Avg WTA=1.34
Avg Waiting=1.5
Std WTA=0.34
```

- If your Algorithm do a lot of processing processes & the processes doesn't start and stop at the same time then your utilization should be less than 100%

GuideLines:

- Read the document Carefully at least once.
- You can specify any other additional input to algorithms or any assumption but after taking permission from your TA.
- A user should be able to choose between different scheduling algorithms.
- You should specify how your algorithm handles Ties
- Priority values range from 1 to 10 where 1 is the least priority and 10 is the highest priority.
- The program must not crash.
- You need to release all the IPC resources upon exit.
- The measuring unit of time is 1 sec, there are no fractions so no process will run for 1.5 second or 2.3 seconds it only takes integer values.
- You can use any IDE (eclipse, codeblocks ,Netbeans,kDevelop,code lite,...etc) you want of course though it would be a good experience to use make files and standalone compilers and debuggers if you had time for that .
- Spend a good Time in Design & it will make your life much easier in implementation
- The code should be clearly commented and the variables names should be indicative.
- Late delivery is **NOT** acceptable.

Platform: Linux
Language: C/C++
Teams: 2-3 members

Grading Criteria

- NON compiling code == ZERO GRADE
- CHEATING == -1 x Assignment grade for both teams.
- Correctness & understanding (50%)
- Modularity, Naming Convention , Code styling (20%)
- Design Complexity & DataStructue used(20%)
- Team work (10%)

Deliverables:

- Upload your project Codefiles, testcases and report as zipped folder on elearning.
- Report contains
  - Data Structure used
  - Your Algorithm explanation and results.
  - Your assumptions
  - Workload Distribution
  - A table for time taken for each task (this will not affect your grade so be honest please)
  - Keep the document as simple as possible and don't include unnecessary information we don't evaluate by word count

Deadline: 28/10/2017 midnight.