Cairo University
Faculty of
Engineering
Computer Engineering Department

OS Scheduler

Document Structure

Objectives

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

Platform *Linux*

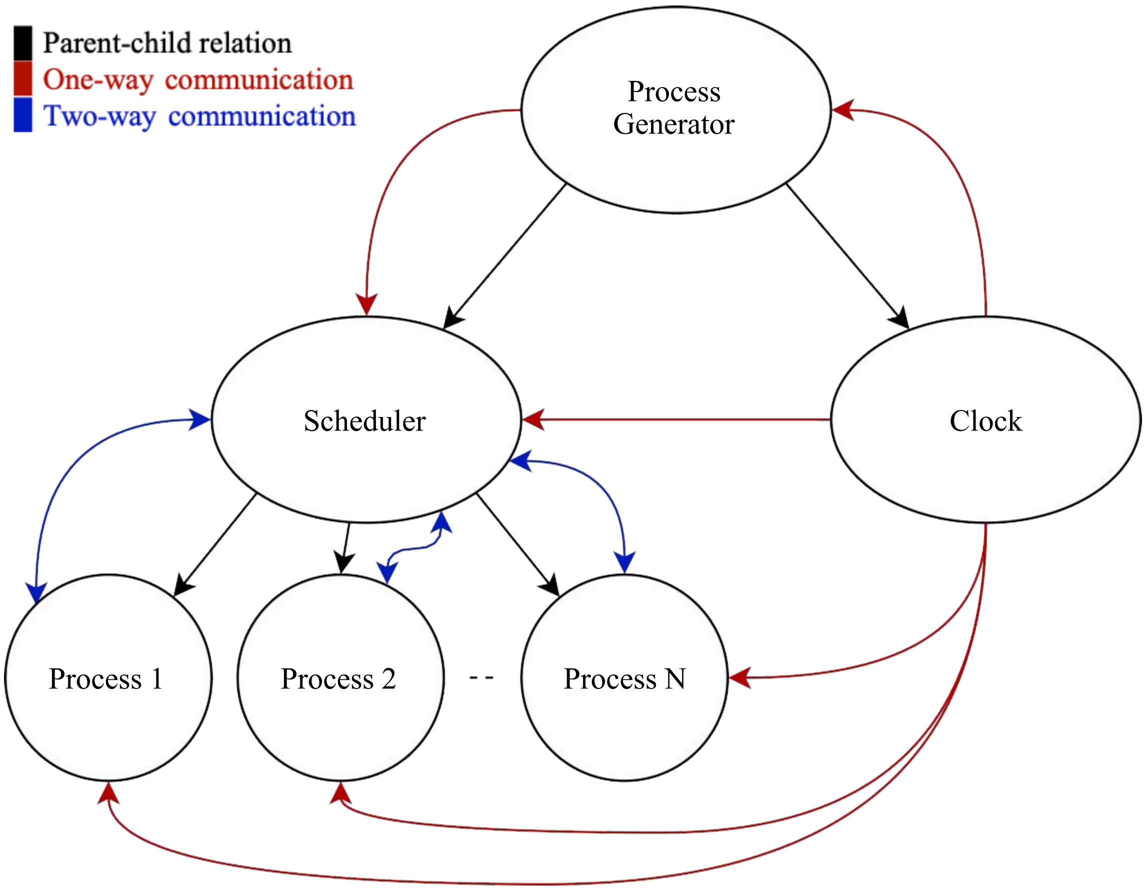
Language *C*

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 trackof the processes in the system and their status.
A process, upon creation, has one of the three states: *Running*, *Ready*, *Blocked* (doing I/O, using other resources than CPU or waiting on an 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 machine with 1-CPU and infinite memory. It is required to make a scheduler with its complementary components as sketched in the following diagrams.



Part I: Process Generator (Simulation & IPC)

CODE FILE *process generator.c*

The process generator should simulate a real operating systems as follows:

- It reads the input files containing the information about processes (check theinput/output section below).
- It gets the chosen scheduling algorithm and its parameters, if any, as command line arguments. These arguments are specified after the name of theprogram in the system command line, and their values are passed to the program during execution.
- It initiates and creates the scheduler and clock processes.
- It creates a data structure for processes and fills it with its parameters (e.g.arrival time, running time, etc.)
- It sends the information to the scheduler *at the appropriate time* (when a process arrives), so that the scheduler places it correctly in its turn among the existing processes. A process should not be sent to the scheduler until itarrives to the system.
- At the end of simulation, the process generator should clear all IPC resources.

Part II: Clock (Simulation & IPC)

CODE FILE *clk.c*

The clock module is used to emulate an integer time clock. *This module is already built for you. Do not change that file.*

Part III: Scheduler (OS Design & IPC)

CODE FILE *scheduler.c*

The scheduler is the core of your work. It should keep track of the processes and theirstates and it decides - based on the used algorithm - which process will run and for how long.

You are required to implement the following scheduling algorithms:

1. Shortest Job First (SJF)
2. Preemptive Highest Priority First (HPF)
3. Round Robin (RR)
4. Multiple level Feedback Loop

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

The scheduler should be able to:

1. Start a new process whenever it arrives. (Fork it and give it its parameters)
2. Switch between two processes according to the scheduling algorithm. (stop the old process and save its state and start/resume another one.)
3. Keep a *process control block (PCB)* for each process in the system. A PCB should keep track of the state of a process; running/waiting, execution time, remaining time, waiting time, etc.
4. Delete the data of the process when it finishes its job. The scheduler knows that a process has finished if its remaining time reaches zero. Note that *the scheduler does NOT terminate the process.* It just deletes its data from the process control block and its data structures.
5. Report the following information:

   a) CPU utilization. idle clocks

   ~~b)~~ Average Weighted Turnaround Time

   ~~c)~~ Average Waiting Time
6. Generate two files: (check the input/output section

   below)(a)Scheduler.log

   (b) Scheduler.perf

Notes:

1. CPU utilization is the fraction of time when there are actual processes running on the system. *If at the end of our simulation, the clock elapsed 20 seconds divided as follows: 15 seconds for useful work (running processes) and 5 seconds (waiting for a process to arrive or idle for any reason), then the CPU utilization should be (15/20)*100.*

2. Waiting time is computed as the total idle clock cycles for each process startingfrom the instant each process arrives to the system.

3. For some algorithms (such as Round Robin), if a process arrives at the same time slot that another process has the turn to run, then you should schedule the process having the turn and the arriving process should be added to the end of the list.
   For example, if we have 3 processes in the system (A, B, and C). Suppose that A is currently running while B and C are in the ready queue (B, C). If a new process D arrives to the system at the same time instant that A finishes its quantum and B is ready to start, then you should (i) place D at the end of the queue (B, C, D) (ii) place A at the end of the queue (B, C, D, A) (iii) schedule the next process in turn which is process B, then the ready queue becomes (C, D, A)

4. In case of ties (e.g. two processes having the same priority in HPF, or two processes arriving at the same time in FCFS), you are free to design which onewould run.

5. Priority values range from 0 to 10 where 0 is the *highest priority* and 10 is the *lowest priority.*

Part IV: Process (Simulation & IPC)

CODE FILE *process.c*

Each process should act as if it is CPU-bound.

Again, *when a process finishes, the scheduler does NOT terminate (kill) the process.It should terminate by itself.*

Part V: Input/Output (Simulation & OS Design Evaluation)

Input File

processes.txt example

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

- Comments are added as lines beginning with # and should be ignored.
- Each non-comment line represents a process.
- Fields are separated with *one tab character* ‘*\t*’.
- You can assume that processes are sorted by their arrival time. *Take care that 2 or more processes may arrive at the same time.*
- You can use the *test generator.c* to generate a random test case.
- All processes will start at t ≥ 1.

Output Files

scheduler.log example (HPF)

```
#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 9  WTA 1.5
```

- Comments are added as lines beginning with # and should be ignored.
- Approximate numbers to the nearest 2 decimal places, e.g. 1.666667 becomes 1.67 and 1.33333334 becomes 1.33.
- Allowed states: *started*, *resumed*, *stopped*, *finished*.
- TA & WTA are written only at *finished* state.
- You need to stick to the given format because files are compared automatically.

scheduler.perf example

```
CPU utilization = 100%Avg
WTA= 1.25
Avg Waiting = 1.5
```

- If your CPU is idle for some clock cycles, then, your utilization should be less than 100%.

Guidelines

- Read the document carefully. You can specify any other additional input toalgorithms or any assumption but after taking permission from your TA.
- The running command is expected to be as follows:

  ```
  ./process_generator.o testcase.txt -sch 5 -q 2,
  ```

  where testcase.txt is the input file name, and 5 is the scheduling algorithm. (Check the scheduling algorithms numbers in Page 4). You may specify extra parameters such as quantum for RR as in the given example. For the other scheduling algorithms, this option can be omitted:

  ```
  ./scheduler.o testcase.txt -sch 3
  ```

- Your 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 willrun for 1.5 second or 2.3 seconds. Only integer values are allowed.
- You can use any IDE (VSCode, Eclipse, CodeBlocks, NetBeans, KDevelop, CodeLite, etc.) you want of course, though it would be a good experience to use make files and standalone compilers and debuggers if you have time for that.
- Spend a good time in design and it will make your life much easier in implementation.
- The code should be clearly commented and the variables names should be indicative.

Grading Criteria

- NON compiling code = ZERO grade.
- Correctness & understanding (70%).
- Design complexity & data structures used (20%).
- Modularity, naming convention, and code style (5%).
- Team work (5%).

Deliverables:

You should deliver code files, and a report containing the following information:

- Discussion of the data structures used in the project for each schedulingalgorithm.
- Your algorithm explanation and results.
- Your assumptions.
- Workload distribution.

Keep the document as simple as possible and do not include unnecessary informationwe do not evaluate by word count!