Zewail City of Science and Technology

مدينة زويل للعلوم والتكنولوجيا

OS Scheduler

Document Structure

- Objectives
- Introduction
- System Description
- Guidelines
- Grading Criteria
- Deliverables

Objectives

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

Platform *Linux*

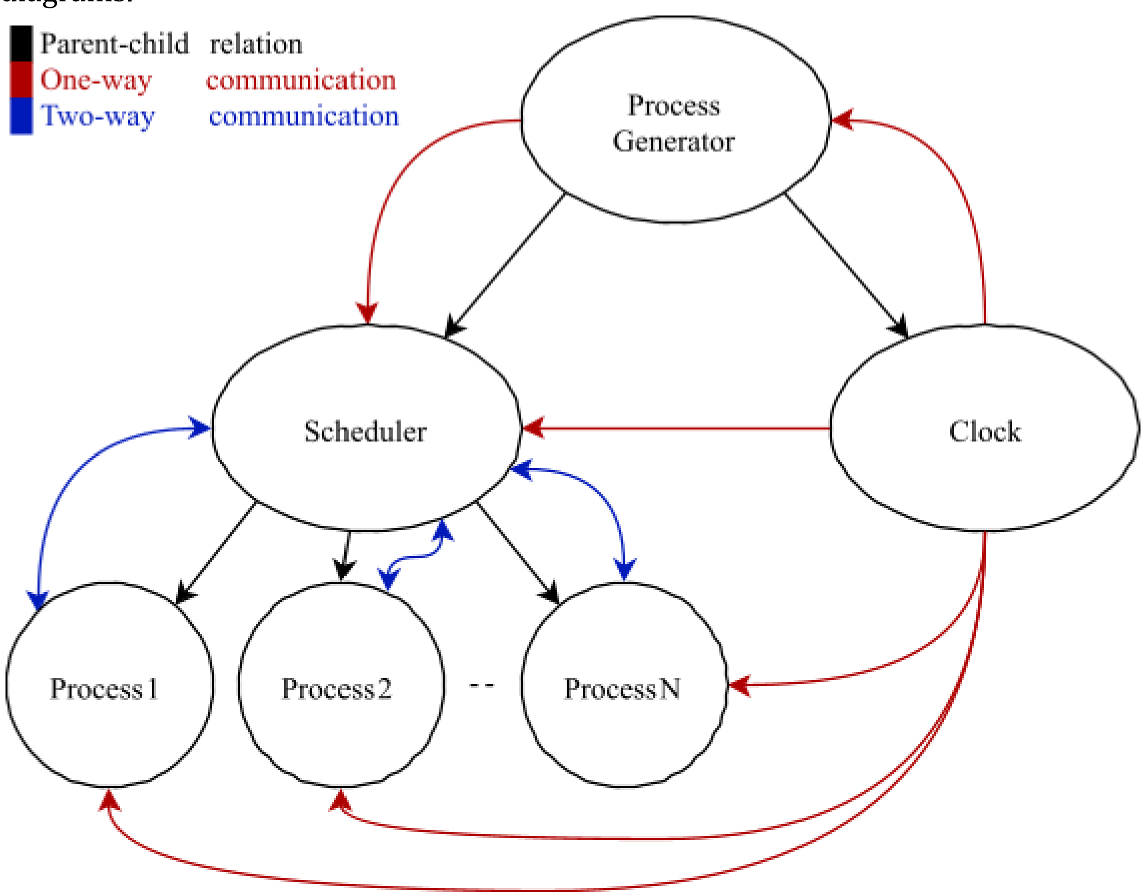
Language *C*

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 track of 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 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 sketched in the following diagrams.



Part I: Process Generator (Simulation & IPC)

Code File *process generator.c*

The process generator should do the following tasks...

- Read the input files (check the input/output section below).
- Ask the user for the chosen scheduling algorithm and its parameters, if there are any.
- Initiate and create the scheduler and clock processes.
- Create a data structure for processes and provide it with its parameters.
- Send the information to the scheduler *at the appropriate time* (when a process arrives), so that it will be put it in its turn.
- At the end, clear 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.*

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 their states and it decides - based on the used algorithm - which process will run and for how long.
You are required to implement the following THREE algorithms...

1. Preemptive Highest Priority First (HPF).
2. Shortest Job Next (SJN).
3. Round Robin (RR).

The scheduling algorithm only works on the processes in the *ready queue*. (Processes that have already arrived.)
The scheduler should be able to

1. Start a new process. (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 a process when it gets notifies that it finished. *When a process finishes it should notify the scheduler on termination, the scheduler does NOT terminate the process.*
5. Report the following information
   (a) CPU utilization.
   (b) Average weighted turnaround time.
   (c) Average waiting time.
   (d) Standard deviation for average weighted turnaround time.
6. Generate two files: (check the input/output section below)
   (a) Scheduler.log
   (b) Scheduler.perf

Part IV: Process (Simulation & IPC)

Code File *process.c*

Each process should act as if it is CPU-bound.
Again, *when a process finishes it should notify the scheduler on termination, the scheduler does NOT terminate the process.*

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.

Output Files

scheduler.log example

```
#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.
- 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 1.34
Avg Waiting = 1.5
Std WTA = 0.34

- If your algorithm does a lot of processing, processes might not start and stop at the same time instance. 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.
- The user should be able to choose between different scheduling algorithms.
- You should specify how your algorithm handles ties.
- Priority values range from 0 to 10 where *0 is the highest priority and 10 is the lowest priority*.
- 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 will run for 1.5 second or 2.3 seconds. Only integer values are allowed.
- You can use any IDE (Eclipse, Code::Blocks, 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 (50%).
- Modularity, naming convention, code style (20%).
- Design complexity & data structures used (20%).
- Team work (10%).

Deliverables

You should deliver code files, test cases and report containing the following information...

- Data Structure used.
- Your algorithm explanation and results.
- Your assumptions.
- Workload distribution.
- A table for time taken for each task. *It will not affect your grade so please be honest.*

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