CS311 Operating Systems
Assignment 02: Simulate a scheduler
Due : Friday, 27th October, 2023.

FCSE
GIKI

Fall 2023

1 Objectives

1. Refresh your C programming.
2. Understand process scheduling.

2 Description

In this assignment you will write a simulator for a process scheduler using C language. The scheduler will be given info about the processes' runtimes and their arrival times via an input file, as well as the scheduling policy, in command line arguments. The scheduler will simulate the running of these processes according to the given scheduling policy.

2.1 Working

Your executable file, when run, should take two arguments:

- a `.dat` filename that contains process data
- a string representing a policy `FIFO, SFJ, STCF, RR`.
- for example: `./a.out processes.dat RR`

2.2 Input

The process data `.dat` file contains lines which can be either:

- a comment: in this case the very first character is always a hash sign, or
- process info: this will contain fields `process name`, `PID`, `process run time`, `process arrival time`, separated by colons. For example the line "P1:12:7:3" means that the process name is P1, its ID is 12, its runtime is 7, and it arrived at time 3.

You can assume that the data in the file will always in the correct format. The process name can have upto 10 characters and the other fields will have unsigned integer values. A line in the input file can have a maximum of 1000 characters.

Your program should check for inputs and exit with code -1 in case of missing or wrong inputs after displaying an appropriate error message.

2.3 Inner workings

Please respect the following in your implementation:

- context switch is instantaneous
- newly arrived processes are inserted at the end of ready queue.
- if two processes arrive at the same time, then they should be inserted in ready-queue in the order they are found in the input file
- if at end of a time-slice, P1 is getting de-scheduled and, P8 and P9 arrive at the same moment, P1 is inserted first at end-of-ready-queue and then P8 and P9.
- SJF and STCF keep the ready queue sorted at all times (w.r.t time-to-completion)
- Wherever needed, assume that a time-slice is one clock-tick.

2.4 Output

When your program simulates the scheduler, at every step it should output (**on the standard output**) the state of the system in the follwoing colon-separated format:
`time:runing_process_name:names of processes in ready_q, with their time-to-completion in parentheses, comma separated:`
where:

- `time` represents the number of clock-ticks passed since the system started. Assume a clock-tick lasts 1 millisecond. The system starts at 0, and the clock-tick between 0 and 1 ms is represented as 1 in the output.
- `running_process_name` represents the name of the process that was in the running state for this clock-tick. If in a given clock-tick no process is running, the output shows the string "idle" in this place.
- `ready queue names` is a comma-separated list of names of process in ready state during this clock-tick, with their time-to-completion in parentheses. In case the queue contains no procesess, the output should display the string "empty".
- For example, the line "6:P3:P1(7),P2(3),:" means that at clock-tick 6, the process P3 is running on the processor while processes P1 and P2 are in the ready queue with times-to-completion 7 and 3, respectively.

In case of any confusion, refer to the provided sample implementation and run it on the accompanied input file. `processes.dat` is an example of an input file whereas the *.output files are examples of outputs to be expected for the respective policies for this particular input file.

2.5 Code Organization

As a practice in building multi-file C codes, you will organize your codes in separate files:

- `u2021xxx_main.c` containing the `main()` function and other common code.
- `u2021xxx_fifo.c, u2021xxx_sjf.c, u2021xxx_stcf.c, u2021xxx_rr.c` each containing the code functions implementing the respective scheduling policy.
- `*.h` any header files that you may need (keep the naming pattern as for *.c files).
- `Makefile` a make file containing rules about building and running your program. Your makefile should contain at least the following rules:
  - `all`: "make all" should build the executable for your scheduler.

- `clean`: "make clean" should remove any intermediary files generated during compilation, including the executable.
- `rebuild`: a combination of `all` and `clean`.
- `runfifo`: should run the executable with `processes.dat` and "FIFO" as arguments.
- `runsjf`: should run the executable with `processes.dat` and "SJF" as arguments.
- `runstcf`: should run the executable with `processes.dat` and "STCF" as arguments.
- `runrr`: should run the executable with `processes.dat` and "RR" as arguments.

Refer to the chapter 15 of the King book "C Programmig A modern approach" mentioned in the syllabus and the PDF on this link `https://pages.cs.wisc.edu/~remzi/OSTEP/lab-tutorial.pdf` on your course book website.

Including `*.c` files in each other will get you a straight **zero** in this assignment.

3 Submission

You will be required to submit:

- all the code (.c) files
- a Makefile containing rules for all, rebuild, run, etc.

4 Rubric

Please make sure that your output EXACTLY matches the sample output!, otherwise you lose half the marks in that section.

Marks:
FIFO + SJF working perfectly: +40 marks
STCF + RR working perfectly: +40 marks
General working: +10 marks
Submission conforms to guidelines: +10 marks

Penalties:
Code doesn't compile: -100 marks
Code has warnings: -10 marks
Program crashes: -30 marks
Late submission: -20 marks + -10*num_days

Marks obtained: = max (0, marks + penalties).