CSE 231: Operating System 06/10/2023 **Assignment-3 (Section-A)**

SimpleScheduler: A Process Scheduler in C from Scratch

Due by 11:59pm on 15th October 2023
(Total **10%** weightage)
Instructor: Vivek Kumar

No extensions will be provided. Any submission after the deadline will not be evaluated. If you see an ambiguity or inconsistency in a question, please seek clarification from the teaching staff.

Plagiarism: **This is a pair programming-based assignment that you must do with the group member that you have already chosen. No change to the group is allowed. You are not allowed to discuss the approach/solution outside your group.** You should never misrepresent some other group's work as your own. In case any plagiarism case is detected, it will be dealt as per the new plagiarism policy of IIITD **and will be applied to each member in the group.** Even if you are a single member group, there will not be any relaxations in marking scheme/deadlines, and the same rubric will be followed for each group.

Open-sourcing of this assignment solution is not allowed, even after the course gets over.

General Instructions

a) Hardware requirements:

- a) You will require a machine having any Operating System that supports Unix APIs. You should not use MacOS for OS assignments. You can either have a dual boot system having any Linux OS (e.g., Ubuntu), or install a WSL.

b) Software requirements are:

- a) C compiler and GNU make.
- **b)** You **must** do version controlling of all your code using github. You should only use a **PRIVATE** repository. If you are found to be using a **PUBLIC** access repository, then it will be considered plagiarism. NOTE that TAs will check your github repository during the demo.

Assignment Details

The OS as a juggler is able to schedule limited number of CPUs across a large number of processes by using some CPU scheduling policy, where each process in the ready state gets to execute on the limited number of CPU resources for some quantum. Once the quantum of running process expires, it is moved to the ready queue from the running queue, and another process from the ready queue is brought into the running queue. In this assignment, you will implement such a process scheduler in plain C, although with limited capabilities than the actual scheduler (Hence, the name as SimpleScheduler). Your SimpleScheduler will borrow code snippets from your assignment-2 and would work as mentioned below:

1. Implementation Details

Basic Functionalities

a) For using SimpleScheduler, the user should first start the execution of the SimpleShell. b) During the launch, SimpleShell should take the total number of CPU resources (**NCPU**) and time quantum (**TSLICE** in milliseconds) permitted to execute on a core as the command line input parameter.
c) SimpleShell would execute in an infinite loop until explicitly terminated by the user. d) It will show a command prompt where the users can run their executable as follows (user is **not** allowed to submit any program that has blocking calls, e.g., scanf, sleep, etc., and the user program **cannot** have any command line parameters):

SimpleShell$ submit ./a.out

e) The “submit” command should create a new process to run the executable passed by the user.
f) The process once created should not directly start the execution of the a.out. It would wait for some signal (of your choice, but should be optimal) from the SimpleScheduler.
g) The SimpleScheduler is a daemon (only one instance irrespective of NCPU value) employed by the SimpleShell to carry out CPU scheduling activities. It must use bare minimum CPU cycles. h) SimpleScheduler maintains a ready queue of processes that are ready for execution. Processes are added to the rear of the queue, and taken out from the front of the queue in a round-robin fashion. i) SimpleScheduler’s job is really simple. It should simply take NCPU number of processes from the front of the ready queue and signal them to start running. After the timer expires for TSLICE, SimpleScheduler will signal the NCPU running processes to stop their execution and add them to the rear of the ready queue.
j) For simplicity, you can assume the quantum TSLICE begins and expires for all the currently existing jobs simultaneously, irrespective of their arrival and termination time. If a new process arrives while some processes are already running, the new process can be scheduled from next TSLICE onwards. k) User should be able to submit any number of jobs and at any given time.
l) Termination of SimpleShell will also terminate the SimpleScheduler.
m) SimpleScheduler should terminate only after the natural termination of all the jobs given by the user. Until then, it should continue working as designed.
n) Upon termination, SimpleShell should print the name, pid, execution time, and wait time of all the jobs submitted by the user.
o) We will not evaluate any other functionality or code that you already implemented inside SimpleShell in assignment-2.
p) Users of SimpleScheduler are **not** allowed to submit any jobs using pipes.

Advanced Functionalities

a) Demonstrate priority scheduling using SimpleScheduler, by allowing the user to specify a priority value in the range 1-4 (e.g., priority value 2) as a command line parameter while submitting the job as follows: **SimpleShell$ submit ./a.out 2**
b) If the priority is not specified by the user, the job will have the default priority of 1. c) You are free to come up with your own heuristic for the effect of priority on the scheduling. d) You should come up with some form of statistics that demonstrate the effect on the job scheduling of each job due to the respective priorities assigned to them by the user.

2. User Executable (jobs)

a) The TAs (user) should be able to use any executable of their choice (satisfying the condition 1-d mentioned above) in your SimpleShell.
b) All they would do is simply add the following code snippet in the top of their program immediately after the include for stdio.h.

```
#include “dummy_main.h”
```

c) You are allowed to create your own “dummy_main.h” (only one) and supply it to the users of SimpleShell.
d) The minimum content of the above header file is as follows. No changes should be done to the code shown in red font. You can only add extra code in the place shown in green font.

```
int dummy_main(int argc, char **argv);
int main(int argc, char **argv) {
        /* You can add any code here you want to support your SimpleScheduler
        implementation*/ int ret = dummy_main(argc, argv);
        return ret;
}
#define main dummy_main
```

e) You should find on your own why would you require this dummy_main.h file.

3. Requirements

a. You should strictly follow the instructions provided above.
b. Proper error checking must be done at all places. Its up to you to decide what are those necessary checks.
c. Proper documentation should be done in your coding.
d. Your assignment submission should consist of two parts:
  a. A zip file containing your source files as mentioned above. Name the zip file as “group-ID.zip”, where “ID” is your group ID specified in the spreadsheet shared by the TF.
  b. A design document **inside the above “zip”** file detailing the contribution of each member in the group, detailing your SimpleScheduler implementation, and the link to your **private** github repository where your assignment is saved.
e. There should be **ONLY ONE** submission per group.
f. In case your group member is not responding to your messages or is not contributing to the assignment then please get in touch with the teaching staff immediately.