To Schedule or Not To Schedule?

CMPE 322

1 Due Date

Due Date is **08.01.2024 Monday 04:00**

2 Short Description

In the first project, we will learn how to manage processes (tasks) in an operating system for a single CPU (with a single core). For this purpose, we expect from you to implement a preemptive priority scheduler (with some minor extensions) in C language.

3 GCC Installation

Please refer to the instructions on Project 1. Please cite your tested OS (Ubuntu or Mac) in your report.

4 Preemptive Priority Scheduling

Priority scheduling is a method of scheduling processes based on priority. In this method, the scheduler chooses the tasks to work as per the priority, which is different from other types of scheduling, for example, a simple round robin. Priority scheduling involves priority assignment to every process, and processes with higher priorities are carried out first, whereas tasks with equal priorities are carried out on a first-come-first-served (FCFS) or round robin basis. In this project, we expect you to choose round robin algorithm as the scheduling algorithm for equal priorities.

In preemptive priority scheduling, at the time of arrival of a process in the ready queue, its priority is compared with the priority of the other processes present in the ready queue as well as with the one which is being executed by the CPU at that point of time. The one with the highest priority among all the available processes will be given the CPU next.

The difference between preemptive priority scheduling and nonpreemptive priority scheduling is that, in the preemptive priority scheduling, the job which is being executed can be stopped at the arrival of a higher priority job. Once all the jobs get available in the ready queue, the algorithm will behave as non-preemptive priority scheduling, which means the job scheduled will run till the completion and no preemption will be done.

5 The Process Structure and the Code Files

As you learned in the lectures, a process contains the program code and its current activity (context). Normally, a program code is a binary file, which contains the instructions that are executable by the CPU. However, for the sake of simplicity, we will use text files (e.g., “P1.txt”) that represent the executable code of a process in this project. The structure of the text file is shown in Table 1. Each row of the text file represents an instruction in which the only column presents the name of the instruction.

Table 1: Table of the code file of Process1 (Not exact, just an example).

INSTRUCTIONS
instr1
instr2
instr5
instr1
instr8
instr7
...
exit

You will be provided with ten different text files, named from P1.txt to P10.txt representing the ten different processes. No more processes will be given/needed. These files will not be changed during testing of the project.

6 Set of Instructions

All processes use multiple instructions from the set of instructions (same instruction can be used multiple times in a process). List of instructions will be given in the form of the table below.

Table 2: Table of the set of instructions (Not exact, just an example).

INSTRUCTIONS	DURATION (MS)
instr1	20
instr2	40
instr3	50
instr4	70
instr5	20
instr6	30
...	...
exit	10

You will be provided with a single text file for the instruction set, named instructions.txt. It will contain 20 different instructions (plus and exit instruction) and their duration in ms. This file will not be changed during testing of the project.

Each instruction in this table is an atomic operation, which means if the CPU starts to execute a specific instruction of a process, it cannot be interrupted to schedule another process until the execution of that instruction is completed. Assume that “instr3” is started to be executed, and another process with a higher priority arrived after 20 ms (30 more ms to complete “instr3”). The scheduler should stop the process after “instr3” is completed, add this process to the appropriate place in the ready queue, and start executing the newly arrived (with the highest priority) process. Finally, you might have noticed that the last instruction name is "exit" which means that this process should be finalized and it should be removed from the system after the exit instruction is executed.

In addition to the program code, we need to store the current activity (context) of a process to reschedule this process and continue to execute it with the CPU. Normally, the current activity of a process contains different elements such as the registers in the CPU and stack of the process. In this project, we just need to store the line number of the last executed instruction before sending this process to the ready queue.

Don't get confused by the names of the instructions. These are not real machine codes and you will not really execute. The only important thing is that you have to calculate the execution times of these instructions correctly. So, you can stop a process, send it back to the ready queue, and run another process at the right time. You don't need to implement any real data structure of queues, just properly calculate the order and execution times in your code and it will be enough.

7 The Process Definition File

The process definition file (i.e., "definition.txt") is a text file that provides the initial information of all processes in the system. The following table shows the structure of this file.

Table 3: Table of an example definition file (Not exact, just an example).

PROCESS	PRIORITY	ARRIVAL TIME	TYPE
P1	2	10	GOLD
P2	4	80	SILVER
P5	5	0	SILVER
P7	1	170	SILVER
P8	1	250	PLATINUM
P10	3	280	SILVER

You will be provided with a single text file for definition of processes, named definition.txt. It will contain a non-determined number of processes. This file WILL be changed during testing of the project.

Assumptions on the definition file:

- At least one process will arrive at time 0.
- The number of rows in this file will change between 1-10.
- A process cannot arrive more than once.
- Not all processes have to be present at the file, however the rows (the processes) of the definition file will differ during the testing of the project. You will be given an example definition file, but you are not limited to it.

8 Scheduling Details

- The algorithm does start with a context switch.
- Each context switch is 10ms.
- Platinum processes are to be executed immediately, and they can NOT be preempted.
- If more than one platinum processes arrive at the same time, they will be executed by the order of their priority.
- If more than one same type processes have the same priority and arrive at the same time, they will be executed by the order of their names (P1 should be executed before P2.)
- Silver Proceseses' Time Quantum is 80 ms.
- Gold & Platinum Proceseses' Time Quantum is 120 ms.
- If a silver process gets three quantums of execution, at the end of the third, it becomes a gold process.
- If a gold process gets five quantums of execution (when it is gold), at the end of the fifth, it becomes a platinum process.
- Other than the time quantum, silver and gold processes have no differences in importance of execution.
- You cannot context switch during an instruction, you can context switch at the end of an instruction.
- As the outputs will be checked via a program, please be careful about ANY unnecessary prints. Floating point precision in the outputs is: "One digit after the dot is enough".
- The testing files will follow exactly the same syntax, spacing as the given text files, so there will be no crippled, unnecessary spaced text files.
- Automatic testing part will not be modified for whatever the reason.

Below you may find the expected output for the provided process definition file and programming code files. The waiting time and turnaround time for all processes are needed. The output will be to stdout not to a file. The output should consist of two lines. First line should have only the average waiting time (in ms) for all processes, second line will have only the average turnaround time (in ms) for all processes.

These output numbers ARE real results for your given definition file, additional input&output files are provided in project files.

Table 4: Table of an example output file

617.5
1135

Reminder:
Turnaround time = Time of Completion - Time Of Arrival
Waiting Time = Turnaround Time - Burst Time

9 Compile&Run Details

Depending on your design, whether it is a single file or multi-file program, your program will be compiled and it is executed as a new process by the terminal. A simple "make" command should be enough to compile your program. Please refer to the problem sessions about the "makefile". Your executable should be named "scheduler" (without extension). To sum up, we should be able to compile and run your code by just entering the following commands:

```
make
./scheduler
```

10 Project Requirements

- Your project should have generated the expected output file for the modified process definition file and process code files. (80%) (This part will be done via program, so please be careful with your outputs.
- Use comments to explain your code (10%), also you should not use online schedulers or just printing the outputs without calculations. You need to calculate the waiting time and turnaround time for all processes by yourselves in your code.
- PDF file of the report on your approach. This report should not exceed two pages and contain details of your implementation approach, the difficulties that you have met, etc. (10%)

11 Submission Details

This project must be implemented individually. Place all code files and the report PDF file into a folder named after your student ID (e.g. 2017400200). Zip the folder for submission and name the .zip file with the same name. Submit the .zip file through Moodle.

- Your submission should contain a makefile, a PDF report and the code files. (Zip them into one folder)
- No late submissions or mail submissions. The deadline is VERY strict.
- A submission that cannot be compiled or without makefile is -30 points.
- Please comment on your code, as it will be graded.
- Questions about the project on Moodle forum is greatly appreciated, please don't hesitate to ask. Questions by mail will not be accepted.
- Questions at the last three days before the deadline will not be responded, as they tend to be panic questions.

  .

12 Academic Honesty

Needless to say, honesty and trust are crucial to all of us. Cheating, plagiarism and collusion are serious offences, and they will result in an “-50” grade for the first time and, if repeated, an overall F grade and disciplinary action.

If you are not certain whether an action violates Academic Honesty please ask.