Project - 1
CSC 4103 – Operating Systems
Due: October 20th @11:59pm

In this project, you will learn how to simulate Operating Systems and how to implement some Operating System components in OSP environment. Your task in this project will be to implement the **CPU Scheduling** component of an Operating System. Please read all instructions carefully before starting the project.

1. Preparation

Before beginning your work, read the OSP handbook carefully (especially Chapters 1.1-1.5 & 1.7 from OSP 1.0 or Chapters 1-4 from OSP 2.0). It is important to understand the usage of the variables and their descriptions before starting programming. You may also need to revisit the related materials in your text book (Silberschatz) to get the conceptual background necessary for the project. For this purpose you should read Chapter 5.3 from Silberschatz for different CPU scheduling policies

2. Programming Task - Implement CPU Scheduling Policies in OSP:

The objective of this task is to implement a CPU module with a specific scheduling policy in OSP. You are required to implement a **Multilevel Feedback Queue** scheduling algorithm in this task. The general descriptions of this algorithm can be found in Section 5.3.5 of Silberschatz. The followings are specific requirements of this task:

1. The ready-queue is partitioned into three separate queues, namely, the foreground, intermediate, and background queues. The foreground and intermediate queues will use Round-Robin scheduling, and the background queue will use Shortest-Remaining-Job-First scheduling.

2. Every new process is added to the foreground queue. These processes will be assigned to the CPU in **Round-Robin** fashion. However, if a process has been assigned to the CPU for **three** times, and yet has not finished its execution, the process will be re-allocated to the intermediate queue. Then, again if the process from the intermediate queue has not finished its execution after being assigned to CPU for **three** times, the process will be shifted to the background queue. At the background queue, a **Shortest-Remaining-Job-First** scheduling mechanism will be applied. (Note that you may use the priority field in the PCB structure as a counter of the number of times the process has been assigned to the CPU).

3. Based on a **fixed-priority preemptive scheduling** scheme, no process in the intermediate queue could run unless the foreground queue is empty. Likewise, a process in the background queue will not be allocated to the CPU unless the intermediate queue is empty. If a process enters the foreground queue during the execution of a process from the intermediate or background queues, the running process must be preempted and put back to the end of the appropriate queues; the preempted process may be put back to its old queue or to the new queue following

the rule in the first requirement depending on the number of trips it was assigned to the CPU.

Files needed for this task can be found in directory:

- **C version:** /classes/cs4103/cs4103_kos/project-1/prj1_C
- **Java version:** /classes/cs4103/cs4103_kos/project-1/prj1_Java

Create a directory called "*prog1*" in your home directory, and copy all the files in the above directory into your new directory.

3. Implementation

For C programmers:

Open the template file *cpu.c*. You will find all the external data structures and functions which are needed by the scheduling module have already been declared. You will also find the bodies of the three functions, waiting for you to complete them.

In order to incorporate your CPU scheduling policy to OSP, you have to implement three functions of the CPU module. They are:

1. *cpu_init()* is a function to set up the initial environment for CPU scheduling (e.g., initiate variables, set up the queue structure).
2. *insert_ready()* will insert a process to the ready-queue and change status of the process to ready. It must also verify the validity of the process to be inserted into the queue.
3. *dispatch()* will choose a process in the ready-queue and assign it to the CPU.

Detailed descriptions of these functions can be found in OSP 1.0 Handbook section 1.7.

For Java programmers:

Open the template class files *ThreadCB.java and TimeInterrupHandler.java.* You need to implement the missing methods in these two classes, including:

1. init()
2. do_create(TaskCB task)
3. do_dispatch()
4. do_kill()
5. do_suspend(Event event)
6. do_resume()
7. do_handle_Interrupt()

Detailed descriptions of these functions can be found in OSP 2.0 Handbook section 4.3.

4. Running your Simulator

After finishing the programming task, type UNIX command "make" in your directory. Your module will automatically compile and link into the OSP system. An executable file named **OSP** *(or a jar file if you are using java)* will be created in your current directory. To run your program, just type "OSP" *(or using your java virtual machine)*. The system will prompt you to input simulation parameters and will run with them. Using "OSP.demo" *(or Demo.jar)* you can get the running result of a fine tuned program provided by OSP. You may compare the result with that of yours and adjust your program accordingly.

5. How to Submit

After you finish each task of your project, you can submit it by typing "*p_copy 1*". Your program will be copied automatically to the instructor's account. After submission, you still can modify your program and hand in again. A new submission will overwrite the previous one.

6. Requirements

No report is required for this project. However, you are expected to submit the program with clear and sufficient comments and explanations. Presentation will be counted as a part of grading. All the programs should be submitted by **October 20th, @11:59pm.**