King Abdul Aziz University

Faculty of Computing & Information Technology
Computer Science Department

CPCS-361 Operating Systems 1 Project

Assign Date:	Friday 19/4/2024
First submission Date:	Wednesday 1/5/2024
Project Presentation:	Tuesday 7/5/2024
Final submission Date:	Sunday 12/5/2024
Course Coordinator:	Dr. Asaad Ahmed
Course Instructors:	Prof. Maher Khemakhem & Dr. Mai Fadel

CLO: 8 SO: 2

- This project is worth 10% of the overall module marks (100%).
- No project will be accepted after 11:59 pm for any reasons.
- This is a GROUP project - 3 students per group.

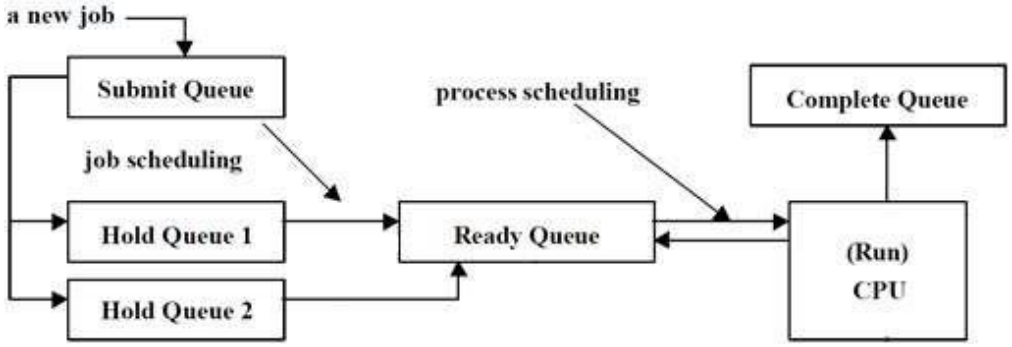
****

Part I

1. Introduction

Design and implement a program that simulates some of the job scheduling and CPU scheduling of an operating system. Your simulator must conform to the criteria established in these specifications. The input stream to the program describes a set of arriving jobs and their actions. The following diagram describes Job and process transitions.

2. A graphic view of the simulator



When a job arrives, one of three things may happen:

1. If there is not enough *total* main memory or total number of devices in the system for the job, the job is rejected and never gets to one of the **Hold Queue**s.
2. If there is not enough available main memory or available devices for the job, the job is put in one of the **Hold Queues**, based on its priority, to wait for enough available main memory (**Preallocation**).
3. If there is enough main memory and devices for the job, then a process is created for the job, the required main memory and devices are allocated to the process, and the process is put in the **Ready Queue.**

When a job terminates, the job releases any main memory. The release of main memory may cause one or more jobs to leave one of the **Hold Queues** and move to the **Ready Queue**.

Assume that all the two **Hold Queues** are based on priority. There are two external priorities: 1 and 2, with 1 being the highest priority. **Priority <u>is only used</u> to put processes in Hold Queues** (Hold Queue1>Priority 1 and Hold Queue2>Priority 1):

- Job scheduling for **Hold Queue 1** is Based on **the Requested Units of Main Memory** (Ascending Order).
- Job scheduling for **Hold Queues 2** is First In First Out (FIFO).

The process scheduling will be **Dynamic Round Robin**. The operating system adjusts the *time quantum according to the burst time of the existed set of processes in the ready queue*. For the first process, it begins with time quantum equals to the burst time of first process, which is subject to change after the end of the first time quantum. Repeatedly, when a new process is loaded into the ready queue to be executed,

the operating system calculates the average of sum of the burst times of processes found in the ready queue including the new arrival process. This method needs two global variables:

- **SR:** to store the sum of the remaining burst times in the ready queue.
- **AR**: to store the average of the burst times by dividing the value found in the SR by the count of processes found in the ready queue.

The algorithm of **Dynamic Round Robin** computation can be formally described by the following pseudo code (`TQ` = time quantum, `BT` = burst time, `AR` = described above):

```
While there are expected new events
  If new Process arrives
    P Enter ready queue
    Update SR and AR
  End If
  Process P is loaded from ready queue into the CPU to be executed
  If (Ready Queue is empty)
    TQ = BT (p)
  Else
    TQ = AR
  End if
  CPU execute P by TQ time
  If (P is NOT terminated)
    Return P to the ready queue with its updated burst time
  End if
  Update SR and AR
End While Loop
```

3. Input specification

The input to your program will be text. Each line in the file will contain one of the commands listed below. Each command consists of a letter in column one followed by a set of parameters.

Each text file contains multiple type "C" (system configuration) commands. (A sample for the format of input file are attached.) There will always be *exactly* one blank after each number in the input file.

a. **System Configuration:**

C 9 M=45 S=12
The example above states that the system to be simulated starts at time 9, and that the system has a main memory consisting of 45 units; and 12 serial devices.

b. **A job arrival:**

A 10 J=1 M=5 S=4 R=5 P=1
The example above states that job number 1 arrives at time 10, requires 5 units of main memory, holds no more than 4 devices at any point during execution, runs for 5, and has priority 1.

c. **A display of the current system status in Readable format (with headings and properly aligned):**

D 11

The example above states that at time 11 an external event is generated, and the following should be printed:

1. A list of each job that has entered the system; for each job, print the state of the job (e.g. running on the **CPU**, in **the Ready** or **Hold Queues**, or finished at time 11), the remaining service time for unfinished jobs and the turnaround time.
2. The contents of each queue.
3. The system turnaround time only at the last display. Assume that the input file has a "D ∞" command at the end, so that you dump the final state of the system. ∞ is to be taken as a large number (i.e. the very last event of the input file and subsequently of the system. Infinity will be denoted by 999999.

4. Helpful Hints

- Let *i* denote the time on the next input command, if there is still unread input: otherwise, *i* is infinity.
- Let *e* denote the time of the next internal event, which will be the time at which the currently running job terminates or experiences a time quantum expiration.
- The "inner loop" of your program should calculate the time of the next event, which is the *minimum* of the *i* and e.
- If *i* = e, then process the *internal event before the external event* (input file event). Notice that if this is not strictly followed, your results will not match the expected output.
- Your simulation must have a variable to denote the "current time". This variable must always be advanced to the time of the next event by a single assignment. (The variable cannot be "stepped" by a fixed increment until it reaches the time of the next event).

5. Implementation Hints

- Implement the **Hold Queues** as sorted linked lists.
- Implement the process table as an array of size 100; the D command dumps the process table. (Do not confuse this table with the PCB's).
- The end of a time slice is an internal event. You may assume a context switch will take zero time.

6. Other Hints:

- When a job is completed, it releases the main memory and implicitly releases the devices. Thus, check the **Hold Queue 1** before the **Hold Queue 2**.
- The only constraints to move from one of the **Hold Queues** to the **Ready Queue** are main memory and devices.

- If more resources are needed than the system contains (**NOT** available, **Actually** contains) then **do not** even consider the jobs. (Kick it out and **do not** "count" the job)
- If jobs have **the same Requested Units of Main Memory** and **same priority**, use FIFO scheduling.
- Handle **all** internal events before external.
- Do not use an array of size 100 for jobs.
- A display event is external.
- There will never be two external events at the same time.
- Absolutely under no circumstances **do not** read the entire input file in the beginning of the program (i.e. **Do not** pre-process the input file).

****

Part II

To evaluate the performance of the simulator of Part I (based on **Dynamic Round Robin)**, implement the same simulator with traditional Round Robin (fixed quantum = 10 + Tn ms, where Tn is your Team number, if your team number is 3 thus the quantum will be 13) and compare the *turnaround* and *waiting* times of both simulators for each input file. Discuss the results and conclude.

Hints:

- Try to reuse the RR developed in Lab session.
- Use Excel graphs to compare results.

****

Deliverables

You should submit **one zip file (CPCS361Project*YourTeamID*.zip) containing:**

1. **your report CPCS361GroupXPOurReport.docx** (Only word File is accepted) where **CPCS361** is your course name, **X** is your group number, and **P** is for project**.** Your report should include:
   - A copy of a printout of your source code, containing a comment with your name, compiler name and version, hardware configuration, operating system, and version.
     NOTICE: You must use standard Java.
   - A copy of the program output on each test data file.
   - The comparative study of PART II.

2. **Your code CPCS361GroupXPOurCode.zip** where **CPCS361** is your course name, **X** is your group number, and **P** is for project. This zip file should include:
   - The source codes [Part I and II]

- The executable files [Part I and II]
- The execution outputs for all test cases [Part I and II]
- Instructions to run your programs.

After the presentation, you should submit **Your Presentation CPCS361GroupXPOurPr.zip** (Only slide show file is accepted e.g.: PowerPoint file or similar format) where **CPCS361** is your course name, **X** is your group number, and **P** is for project**.**

A set of input data files will be made available on Blackboard. Attend classes for further details, answers to questions, and hints on this project. **You will be responsible for all that your Lab instructor says about this assignment in class.**

7. Scoring

Your project will be scored based on the following:

Task	Mark
Input specification	5
Implementation of Scheduling Algorithms and Queues	30
Required Data	10
System requirements	15
Implementation of events	10
Output Specification	10
Results	10
Presentation	10

Late submissions will be penalized by deducting 20% of the score per one day.