CISC-361: Operating Systems : Programming Project: Scheduling and Deadlock Avoidance

Due: May 21, 2018 11:55 PM

Instructor: Dr. Mashayekhy

Submission Instructions

1. You will work in a group of two.

2. Only one of you should submit the project (source code and report).

3. Please include your name and your partner's name for this project in the comments section on Sakai when you submit this project. Please also include both names at the first line of your program as a comment, and in your report.

4. The report should be in .pdf format.

5. You are required to use C/C++ to program the assignment.

6. Submit a separate .c/cpp file for your code. Do not paste your code into a pdf.

7. Write your name at the top of each file.

8. Appropriately comment your code.

9. Provide compile and execute instructions either in a readme file or the accompanying report.

10. Segmentation fault and compile errors would result in a 0 grade.

Grading Criteria

Your project grade will be made up of 2 components:

- Source Code -70%

- Report and Code Quality- 30% - Describe your design approach and the output of your program for the test inputs provided to you.

Assignment

In this project, you will design and implement a program that simulates the job scheduling and CPU scheduling of an operating system. In addition to the scheduling algorithms, you must implement a deadlock avoidance method by implementing the Banker's Algorithm.

The input stream to the program describes a set of arriving jobs and their actions. The following diagram describes job and process transitions.

A graphic view of the simulator
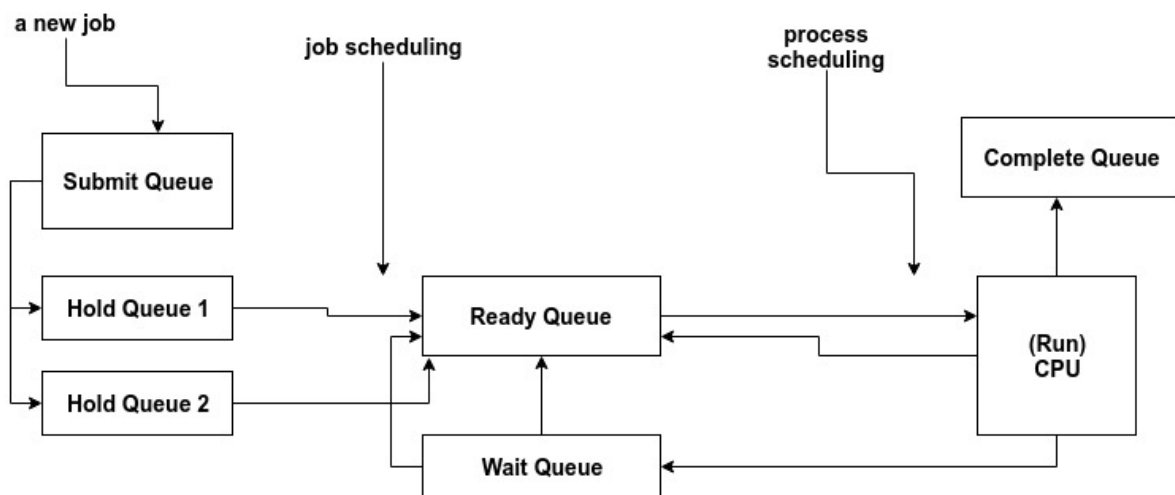
When a job arrives, one of three things may happen:

1. If there is not enough total main memory or total number of devices in the system for the job, the job is rejected never gets to one of the **Hold Queues**.

2. If there is not enough available main memory for the job, the job is put in one of the **Hold Queues**, based on its priority, to wait for enough available main memory.

3. If there is enough main memory for the job, then a process is created for the job, the required main memory is 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 the two **Hold Queues** are based on priority. There are two external priorities: 1 and 2 with 1 being the highest priority. **Priority is only used for the Hold Queue**.

- Job scheduling for **Hold Queue 1** is **Shortest Job First (SJF)**.

- Job scheduling for **Hold Queues 2** is **First In First Out (FIFO)**.

- Process scheduling will be **Round Robin** (FIFO).

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 input text file will contain one type "C" (system configuration) command. All input will be *syntactically and semantically* correct, but you should detect and report other types of errors to the TA. There will always be *exactly* one blank after each number in the input file.

1. **System Configuration:**

   <div align="center">C 9 M=45 S=12 Q=1</div>

   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; 12 serial devices; and a time quantum or time slice of 1.

2. **Job arrival:**

   <div align="center">A 10 J=1 M=5 S=4 R=3 P=1</div>

   The example above states that job number 1 with priority 1 arrives at time 10, requires 5 units of main memory, holds no more than 4 devices at any point during execution(**max demand**), and runs for 3 units of time.times

3. **Request for devices:**

   <div align="center">Q 10 J=3 D=4</div>

   The example above states that at time 10, job number 3 requests for 4 devices. A job only requests devices when it is running on the **CPU**. The Quantum is interrupted to process request. If request is granted process goes to the end of the ready queue, else it goes to the device wait state.

4. **Release for devices:**

   <div align="center">L 10 J=5 D=1</div>

   The example above states that at time 10, job number 5 releases one device. A job only releases devices when it is running on the **CPU**. Quantum is interrupted. One or more jobs may be taken off the Device Wait queue due to this.

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

   <div align="center">D 11</div>

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

   (a) 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 **Hold Queue**, or finished at time 11), the remaining service time for unfinished jobs and the turnaround time and weighted turnaround time for finished jobs.

   (b) The contents of each queue.

   (c) The system turnaround time and system weighted turnaround only at the last display. Assume that the input file has a "D 9999" command at the end, so that you dump the final state of the system.

   Note: As long as the display is readable and has the required information, that is fine.

Implementation Hints

1. Implement the **Hold Queues** as sorted linked lists.

2. You will be graded in part on the maintainability of your code. Therefore, use #define to avoid embedding numeric constants in the code.

3. The end of a time slice is an internal event. You may assume a context switch will take zero time.

4. Consider all input as **integer values**.

Other Hints

1. If there is a completion of a job, check **Device Wait queue** then the 2 **Hold Queues**.

2. When a job completes, it releases main memory and implicitly releases the devices. Now, check the **Wait queue** and 2 **Hold Queues** in that order.

3. The only constraints to move from one of the **Hold Queues** to the **Ready Queue** are main memory and devices.

4. If more resources are needed for a job than the system **contains** then **do not** even consider that job.

5. If jobs have same run-time and same priority, use FIFO scheduling (FIFO within SJF).

6. Devices are only requested by jobs while running and if this happens the job's time slice is interrupted.

7. Under job arrival, max demand is the most devices ever to be requested but, none at start.

8. **Internal Event**: The events that are related to the CPU quantum i.e execution completion and quantum interruption are considered as internal events.

9. **External Event**: Arrival of new jobs, display events are external events.

10. Handle **all** internal events before external.

11. Your simulation must contain 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 reached the time of the next event.

12. A job can only request devices when a process is running.

13. There will never be two external events at the same time.

14. Pretend to allocate devices before calling Banker's.

15. Absolutely under no circumstances do you want to read the entire input file in the beginning of the program (i.e. pre-process the input file).

Sample Input

```
C 1 M=200 S=12 Q=4
A 3 J=1 M=20 S=5 R=10 P=1
A 4 J=2 M=30 S=2 R=12 P=2
A 9 J=3 M=10 S=8 R=4 P=1
Q 10 J=1 D=5
A 13 J=4 M=20 S=4 R=11 P=2
Q 14 J=3 D=2
```

A 24 J=5 M=20 S=10 R=9 P=1
A 25 J=6 M=20 S=4 R=12 P=2
Q 30 J=4 D=4
Q 31 J=5 D=7
L 32 J=3 D=2
D 9999