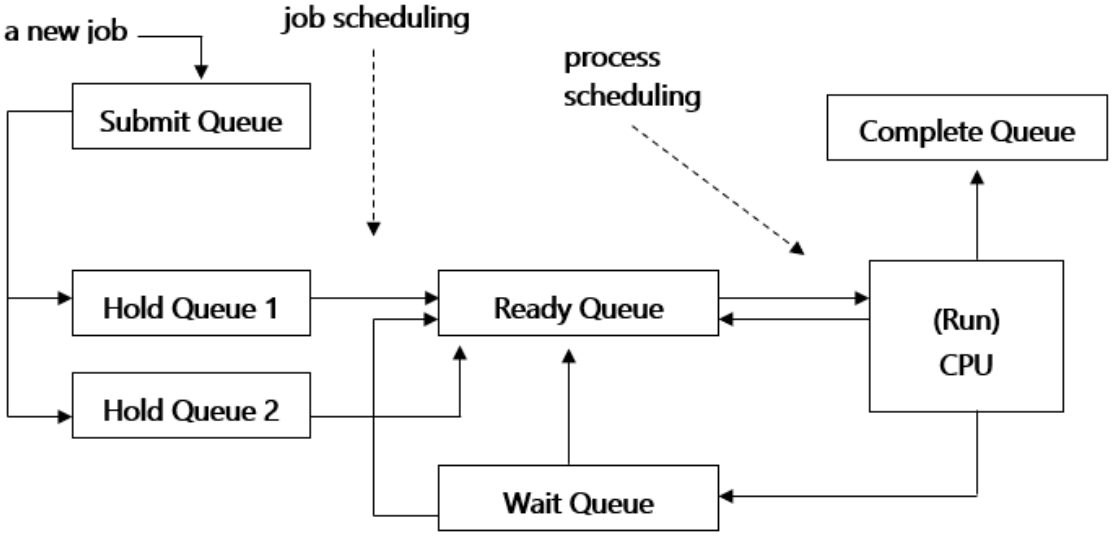
CISC361: Operating Systems
Programming Project: Scheduling and Deadlock Avoidance
Mauriello
Spring 2022

In this project, you will design and implement a program that simulates the job scheduling, CPU scheduling, and deadlock avoidance of an operating system.

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**.

Pre-allocation of memory is required, but device allocation is **dynamic**. Banker's algorithm is used for device allocation.

When a job terminates, the job releases any main memory and devices(s) that it held. The release of main memory may cause one or more jobs to leave one of the **Hold Queues** and move to the **Ready Queue**. The implicit release of devices may cause one or more jobs to move from the **Wait Queue** to the **Ready Queue**.

Assume that the two **Hold Queues** are based on priority. There are two priorities: 1 and 2, with 1 being the highest priority. **Priority is only used for the job scheduler**.

- 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** (with a quantum).

Jobs may acquire and release devices during execution. A device can only be used by one process at a time, because it is a serial (not a random access) device (tapes, printers, etc.)

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. 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:

C 9 M=45 S=12 Q=1

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. A job arrival:

A 10 J=1 M=5 S=4 R=3 P=1

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, and runs for 3 units of time.

3. A request for devices:

Q 10 J=3 D=4

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 the request. If request is granted, the process goes to the end of the **Ready Queue**, else it goes to the **Wait Queue**.

4. A release of devices:

L 10 J=5 D=1

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 **Wait Queue** due to this.

5. 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:

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 **CP**U, in the **Hold Queue**, or finished at time 11), the remaining service time for unfinished jobs, and the turnaround time and waiting time for finished jobs.

b) The contents of each queue.

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

Note: If the display is readable and has the required information, that is fine.

Deadlock

When a running job requests devices, the Banker's algorithm is used to determine if the request can be satisfied. If the request cannot be satisfied, the process is immediately switched from the **CPU** to the **Wait Queue**.

Whenever a running job releases devices and the **Wait Queue** is not empty, the Banker's algorithm is executed in FIFO order on each job in the **Wait Queue** to determine if any jobs in the **Wait Queue** can be allocated its last request of devices. If necessary, the entire **Wait Queue** is checked to restart as many jobs as possible. A similar remark applies when a job that holds devices terminates, because a job implicitly releases devices upon termination.

The **Wait Queue** is checked **FIRST** (before the **Hold Queues**) upon a completion.

Implementation Hints

- Implement the **Hold Queues** as sorted linked lists.
- Implement the process table as an array of size 100; the D command presents the process table.
- The end of a time slice is an internal event. You may assume a context switch will take zero time.
- Consider all input as **integer values**.
- 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 either 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 to grade your project!
- Your simulation needs to 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 reached the time of the next event.)
- You will be graded in part on the maintainability of your code. Therefore, use #define to avoid embedding numeric constants in the code.

Other Hints:

- If there is a completion of a job, check **Wait Queue** then the two **Hold Queues**.
- When a job completes, it releases main memory and implicitly releases the devices.
- The only constraints to move from one of the **Hold Queues** to the **Ready Queue** are main memory and devices.
- Priority is for the job scheduling, not process scheduling.
- if more resources are needed for a job than in the system then **do not** consider that job.
- If jobs have same run-time and same priority, use FIFO scheduling (FIFO within SJF).
- Handle **all** internal events before external.
- Internal Event: The events that are related to the CPU quantum (i.e., execution, completion, and quantum interruption) are considered as internal events. When a device requests or releases devices that is an internal event.
- External Event: Arrival of new jobs (reading from the file) and display events are considered as external events.
- There will never be two external events at the same time.
- The program must not read the entire input file in the beginning of the program to pre-process or use any advanced information. Execution is line by line and your program's operations should not, for example, use future information about device requests.
- When an external release occurs and jobs are still in **Ready Queue**, check the **Wait Queue** to determine if something can start (i.e., get into the **Ready Queue**).
- Devices are only requested by jobs while running on the CPU, and if this happen, the job's time slice is interrupted.
- On an arrival, the "S=" denotes the maximum number of device(s) the job will ever use.
- Pretend to allocate devices before using Banker's.
- A job that requests a device in the middle of a time quantum, whether it gets the device or not, does NOT finish its time quantum. It blocks immediately.
- Do not use an array of size 100 for jobs.

Submissions and Deliverables

- You will work in a group of three.
- Only one of you should submit the project (source code and report).
- Please include your name and names of your partners for this project in the comment section on Canvas when you submit your project. Please also include both names at the first line of your program as a comment, and in your report.
- The report should be in .pdf format and contains the copy of the program output.
- You are required to use C/C++ to program the assignment.
- Submit a separate .c/cpp file for your code. Do not paste your code into the report.
- Appropriately comment your code.
- Provide compile and execute instructions preferably in makefile/readme files (or in the accompanying report).
- Segmentation fault and compile errors would result in a 0 grade.

Your project grade will be made up of 2 components:

• 70% Code

• 30% Final Report including your design approach, code quality, and screenshots of the output of your program given the sample input provided in the following section.

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
```