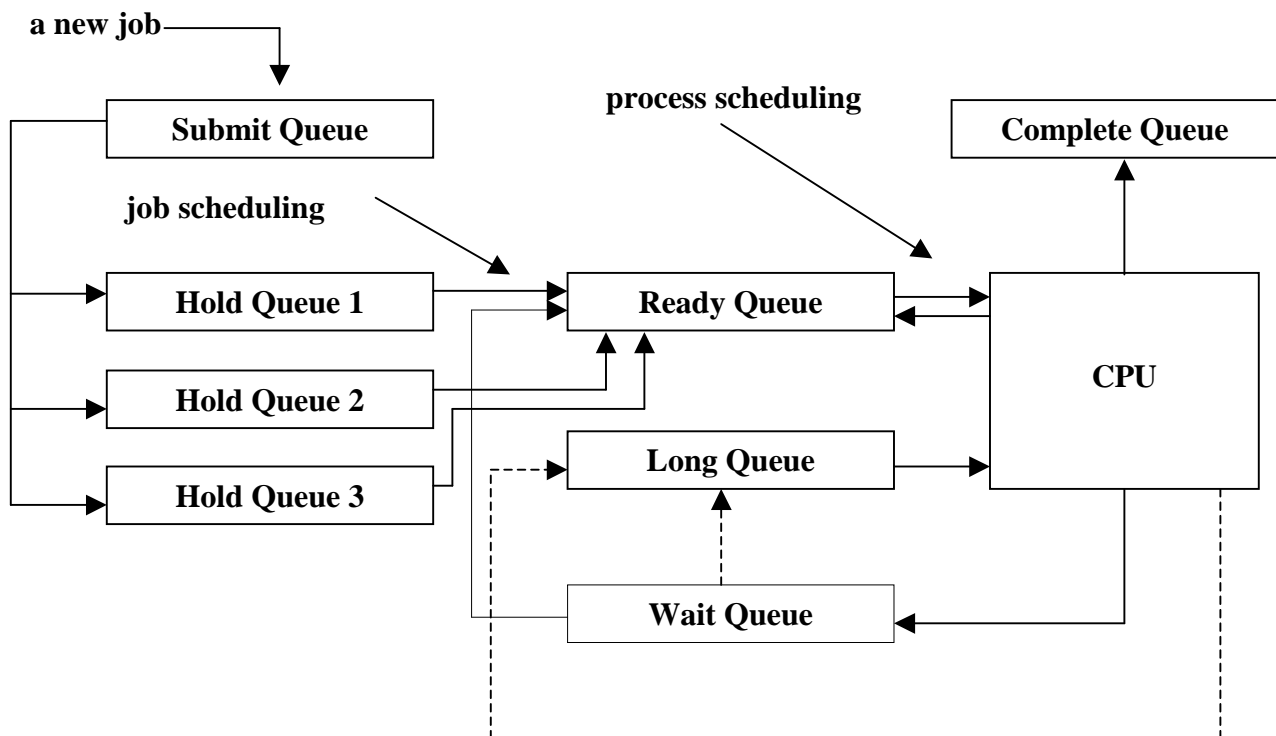


CS 3204 – Operating Systems
Programming Project #2 – Deadlock
Dr. Sallie Henry - Fall 1999
Due Date: 29 September 1999 at 8:00am in the lab

Design and implement a program that simulates some of the job scheduling, CPU scheduling and deadlock avoidance of an operating system. Your simulator must conform to the criteria established in these specifications. A **detailed design** is due on September 14, 1999 in class.

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

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 in the system for the job, the job is rejected and never gets to one of the **Hold Queues** (Note: a check for devices is made; for total devices only).
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 and devices are allocated to the process, and the process is put in the **Ready Queue**.

Preallocation 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 all three **Hold Queues** are based on priority. There are three external priorities: 1, 2, 3 with 1 being the highest priority. **Priority is only used for the job scheduler.**

- Job scheduling for **Hold Queue 1** is **First In First Out (FIFO)**
- Job scheduling for **Hold Queues 2 & 3** is **Shortest Job First (SJF)**

Process scheduling will be **Limited Round Robin** (to be discussed soon). Once a job has exceeded the **threshold** of CPU time, it is considered a long job and may not run until the **Ready Queue** is empty. At that time the **Long Queue** becomes the **Ready Queue** and **Round Robin** process scheduling is used. A process moving from the **Ready Queue** to **Long Queue** generates an internal event. The long queue is FIFO.

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 exactly one type "C" command, which will be the on the first line. All input will be *syntactically* correct, but you should detect and report other types of errors to Shiv. There will always be *exactly* one blank after each number in the input file.

The input will be syntactically and semantically correct, however, due to possible human error, you are requested to please check for the following errors, which may occur in the input data:

- The time on a command is either less than or equal to the time on an earlier command.
- A job requests or releases a device when it is not running.

1. System Configuration:

C 98 M=45 L=6 S=12 Q=1

The example above states that the system to be simulated starts at time 98, and that the system has a main memory consisting of 45; a time excess (threshold) of 6 determines long jobs; 12 serial devices; and a time quantum or time slice of 1.

2. A job arrival:

A 100 J=1 M=5 S=4 R=5 P=1

The example above states that job number 1 arrives at time 100, requires 5 units of main memory, holds no more than 4 devices at any point during execution, and runs for 5. Jobs are numbered 1, 2, 3, ... and arrive in this order. There will never be more than 100 jobs.

3. A request for devices:

Q 102 J=3 S=4

The example above states that at time 102, job number 3 requests 4 devices. A job *only* requests devices when it is running on the **CPU**.

4. A release for devices:

L 106 J=5 S=1

The example above states that at time 106, job number 5 releases 1 device. A job *only* releases devices when it is running on the **CPU**.

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

D 110

The example above states that at time 110 and external event is generated and the following should be printed:

1. A list of each job that has entered the system, ordered by job id; for each job, print the state of the job (e.g. running on the **CPU**, in wait queue, 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.
2. The contents of each queue.
3. The system turnaround time and system weighted turnaround **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 large number (i.e. the very last event of the input file and subsequently of the system. Infinity will be denoted as 999999).

DEADLOCK PREVENTION

When a running job requests devices, the Banker's algorithm (to be discussed soon) 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.

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 either terminates, exceeds the threshold 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 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 reached the time of the next event.)

You may also wish to use a .h file; put your #defines, global variables, and global type and structure definitions in to improve the code readability.

Notice that unlike examples given in class, we will be giving you all the integer values to simplify the problem. At any stage in the computations if you reach no integer values, use **truncation**. For example, if you use integer data types, then this will be automatically done for you. You will use ints only for parameters specified in the input specification and their computations.

You must turn in the following

- 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 C++ / Java. Do not use machine / compiler dependent constructs. Your code must compile and execute in the McBryde 116 lab.
- A copy of the program output on each test data file .
- A floppy disk, labeled with your name, and instructions for running your program, containing
 - source code
 - executable file
 - execution output for all test cases

A set of input data files will be made available for distribution on the 16th of September. Attend class for further details, answers to questions, and hints on this assignment. **You will be responsible for all that your instructor says about this assignment in class.**

Implementation Hints

1. Implement the **Hold Queues** as sorted linked lists. On what key are they sorted?
2. 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.)
3. You will be graded in part on the maintainability of your code. Therefore use #define to avoid embedding numeric constants in the code.
4. The end of a time slice is an internal event. You may assume a context switch will take zero time.

Other Hints:

If there is a completion of a job, check the 3 **Hold Queues** before the **Long Queue**.

When a job completes, it releases main memory and implicitly releases the devices. Now, check the 3 **Hold Queues** and then the **Long Queue**.

Priority is only meaningful in the 3 **Hold Queues**, and **NOT** in the **Long Queue**.

Priority is job scheduling not process scheduling.

The only constraints to move from one of the 3 **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.

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

An event is generated by a process going long. When the next event occurs, then check whether to move it to the **Long Queue** or not.

Handle **all** internal events before external.

Do not use an array of size 100 for jobs.

A display event is external.

Information hiding is important.

Any new job entering the **Ready Queue** sends a **Long Queue** job back to the **Long Queue** (i.e. the **Ready Queue** must be empty for a job on the **Long Queue** to get on the **CPU**.)

When a process is in the **Long Queue**, it does **not** give up its resources.

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

Do **NOT** use a **MEGANODE!!!** Use information hiding unless you plan on losing points.

You can implement the **Submit Queue** as a **Reject Queue**.

Hint: The MEGANODE issue can be solved through inheritance. Think about it.

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).

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.

Check the **Wait Queue** before the three **Hold Queues**.

The **Wait Queue** is FIFO.

External device requests (the only type of device requests that there are) can only be made by jobs on the **CPU**.

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.

Summary of Events and their consequences (transitions):

External

Initial -- simply start the simulation and initializes the "clock"

Arrival -- possible actions:

1. Job is rejected if the max memory is greater than total memory OR if the max devices are greater than system devices
2. Job moves to Hold Queue -- if there is not enough memory available
3. Job moves to Ready -- if there is available memory

Request -- "pretend" to make the request, call Banker's and Banker's will either allow the request to be granted (job goes to the ready queue) or Banker's will determine an unsafe state (job goes to the Wait queue). *If the job moves to wait, there is a possibility that the long queue could become the ready queue (i.e. the job going to wait was the last job in the ready queue and the long queue is not empty).*

Release -- job releases devices, the releasing job goes to the ready queue, the wait queue is then checked (in FIFO order) to see if one or more jobs can move to the ready queue. *If a job moves from the Wait queue, there is a possibility that the Long queue will be affected (ready <-> long).*

Display -- the contents of the queues etc. will be dumped. The display does NOT cause the running job to be interrupted. It lets it finish its time quantum.

Internal

End of quantum -- a context switch occurs.

Job reaches threshold -- job moves to Long queue, if the job was the last in the ready queue, the long queue becomes the ready queue.

Completion -- the job releases its devices (this can cause one or more jobs to move from Wait to ready, or wait to long)

The job releases its memory (this can cause one or more jobs to move from Hold to Ready. (May cause the ready queue to move to long).

*whenever a job moves to the ready queue (arrival, completion, release), the long queue may be affected.