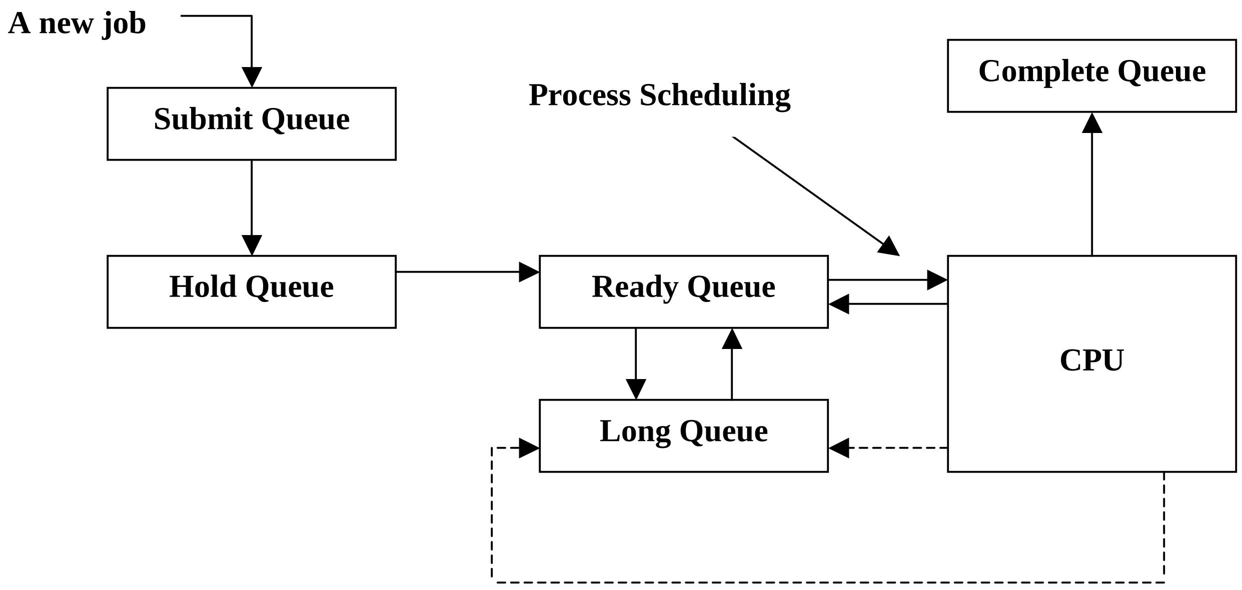
CS 3204 – Operating Systems
Programming Project #1 – Job / CPU Scheduling
Dr. Sallie Henry – Spring 2001
Design Due on 2 February 2001, 23:59:59 PM
Due on 15 February 2001, 23:59:59 PM
(Sign-up sheets available)

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.

A graphic view of the simulator



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

1. If there is not enough free main memory for the job, the job is put in the **Hold Queue** to wait for main memory and devices.
2. 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 for the process, and the process is put on the **Ready queue.**

When a job terminates, the job releases any main memory that it holds. The release of the main memory may cause the jobs to leave the **Hold Queue.**
Assume that the **Hold Queue** is based on the priority. There are three external priorities: 1, 2, 3 with 1 being the highest priority. The priority is only for the Hold Queue.

- Job Scheduler will use preallocation of main memory and devices.
- Job scheduling for queues one and two is **Shortest Job First (SJF).**
- Job scheduling for queue three is **First Come First Served (FCFS).**

Process Scheduling will be **Limited Round Robin.** (To be discussed soon.) Once a job has accrued 0.6 CPU time, it is considered a long job and may not run until the Ready/Run is empty. At that time the long queue becomes the Ready queue and Round Robin is used. A process moving from the **Ready Queue** to the **Long Queue** generates an internal event.

Input Specification

The input to your program will be a 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 first of the line. All input will *syntactically* correct, but you should detect and report other type of errors. There will always be exactly one blank line after each number in the input file.

Please check for the following errors, which may occur in the input file.

- The time on a command is always more than the time on an earlier command.

1. System Configuration:

C M=180 L=0.6 D=12 Q=0.1

The example above states that the system to be simulated has a main memory consisting of 180 memory units; a time excess of 0.6, which determines long jobs; and a time quantum or time slice as 0.1. It also shows that the system has 12 devices.

2. A job arrival:

A 10.0 J=1 M=50 D=4 R=0.5 P=1

The example above states that job number 1 arrives at time 10 which will require 50 units of memory and 4 devices and runs for 0.5 seconds. The job arrives with a priority of 1.

3. A display of the current system status in *Readable* format (with headings and properly assigned): This is an external event

D 11.03

The example above states that at time 11.03, 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,** waiting in the **Hold Queue,** finished at time 11.0), 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 weighted turnaround time based on jobs completed so far.

Assume that the input file has "D <infinity>" command at the end, so that you dump the final state of the system.

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 or experiences time quantum expiration. The "inner loop" of your program should calculate the time of next event, which is the minimum of the *i* and *e.* If $i = e$, then process the internal event before the next command. Notice that if this is not strictly followed, your results will not match the expected output to grade your project!

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.

Although the input data will use "real" numbers, you must convert them to type int by multiplying all the real numbers by 1000 (rounded), do your work using integers, and then convert your results back to real numbers by dividing the appropriate quantities by 1000. I want to think about why this requirement will simplify your programming considerations. At some point in class we'll discuss the reason.

You must be careful in constructing an algorithm to read the input file. Suppose the input file contains "0.2". If this read will scanf into a float with %f format, the result is "0.1999999", whereas the integer desired is "200". You must devise a way of solving this problem.

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

You must turn in the following

- A softcopy 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++. Do not use machine/compiler dependent constructs. Your code must compile in McBryde 116 lab.

- A softcopy of the program output on each test data file

- Submit the design (due on 02/02/2001) and the softcopy of the things mentioned above, to the curator http://spasm.cs.vt.edu:8080/curator/SpasmCurator

Implementation Hints

1. Implement the **Ready Queue** and **Hold Queue** as sorted linked lists. On what key are they sorted?

2. Implement the completion table as an array of size 100; the D command dumps the process table (Do not confuse this tables with the PCBs.)

3. You will be graded on the part of 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.

5. When a job's accrued time exceeds 0.6, an internal event occurs causing the process to move from **Ready Queue** to **Long Queue.**

Other Hints

If there is a completion of a job, check the hold queue before the long queue.

When a job completes, it releases memory and implicitly releases devices. Now check the **Hold Queue** and then the **Long Queue** in that order.

First job in First priority should be checked in hold 1 st.

Priority is only meaningful in the **Hold Queue, <u>NOT</u>** in the **Long Queue.**

Priority is job scheduling not process scheduling.

The constraints to move from the **Hold Queue** 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. Print an appropriate message and 'kick the job out of the system'. Do not include it in the T or W times.

If jobs have same run-time and same priority, use FIFO.

An event is generated by a process going long.

The display command does not have to list processes and jobs in any certain order.

Handle **all** internal events before external.

Do not use array of size 100 for jobs.

A display event is external.

Information Hiding is important.

Any new job entering the **Ready Queue** process 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 run.)

When a process is in a **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 loosing points.