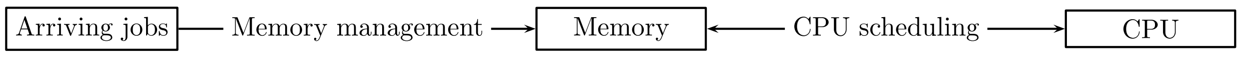
Project 1: Virtualization

Instructor: Dr. Shengquan Wang

Due Time:
Part 1: 6PM, 10/24/2012
Part 2: 6PM, 11/05/2012

You are to design, implement, and then experiment with a program to simulate the memory management method and the CPU scheduler. The following figure shows the overall structure of the simulated system.



1 Memory Management

Memory will be allocated in Memory Units – all of the same size (this sounds like paging, but it isn't. IBMs OS/MVT allocates memory in 2K blocks). Each process will need a specific number of memory units, and a specific amount of CPU burst time.

Each job's memory must be CONTIGUOUS block (this is where we differ from paging). If a large enough block is not available, the next job waits (is not loaded) until enough other jobs complete to free up a large enough contiguous block. You will need to implement three dynamic memory allocation algorithms, "next-fit" and "worst-fit", for locating the block of memory to use for loading a new process.

The behavior of all the jobs your simulator will handle is described in the data file. Associated with every job is a unique process id (pid), the arrival time for the given job, a service time (how long a job has until completion, i.e. how long the jobs "real work" will take), and an address space size. The simulated memory has 256 memory units. The pid is between 0 and 69. The address space size is between 5 and 200 memory units. The service time is between 1 to 100 clock ticks. The data file format is:

```
pid ArrivalTime ServiceTime AddressSpaceSize
```

For example, the data file

```
0 0 10 20
1 5 15 50
```

describes a simulation that has two jobs. The first has a pid of 0, an arrival time of zero (when the simulation starts), needs to do 10 clock ticks of work, and needs a memory size of 20 units. The second job has a pid of 1, arrives at clock tick 5, needs to do 15 clock ticks of work, and needs a memory of 50 units.

2 CPU Scheduler

The simulated system is a based on the simplified MLFQ scheduling algorithm as shown in the example and quiz in class. The three queues are Q_1, Q_2, and Q_3 with time quantum size as 8, 16, and 32. In this system, a process has only one CPU burst and no I/O burst. Priority boost is ignored too. The scheduling procedure follows the policy discussed in class.

3 The Simulated System

The system will have the following behavior. You must have a way to keep time intervals (clock ticks) during the execution of the system. A simple approach to this problem might be to use a loop and for each iteration, increment the clock (initially zero). Your main loop will something like the following:

(1) Check if there is a new job arrival. If yes, put it to the waiting list and start memory scheduler.

(2) If a new process loaded, update it to the ready list.

(3) Run the MLFQ scheduler to select a ready process to run for a certain time period, update the ready list, and so on.

(4) keep track of statistics.

(5) If the job currently running complete, free the allocated memory units, remove it from the ready list, and start the memory scheduler so that any pending process for memory could have the opportunity to be admitted; otherwise, choose next job in the ready list, and go on until all processes are done.

For this simulation, jobs will do no "real" work; i.e. your simulation will implement what happens when the CPU invokes the operating system process scheduler. You can think of this project as a function call inside the operating system. A function call that updates all job states and sets up the next job to run, so that when the function call completes, the CPU has next job and can begin to run it.

4 Output Specification

You will display a memory map every time a new job is loaded or finished. Each process will be represented by a unique character among

```
a-z, A-Z, 0-9, @, #, $, %, &, *, +,  ?
```

Thus, you have 70 unique one-character process IDs to use. As such, your simulation should process up to 70 job. Along with each memory map, give a percentage of unused memory (hole percent). Hole percent is the fraction of the empty memory over the overall memory. Your output should look something like this:

```
000  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbb
064  bbbbbbbbbbbbbbbbbbbbcccccccccccccccccccccccccccccccddddddddddddd
128  ddddddddddddddddddddddddddd____________________fffffffffffffgggg
192  gggggggggggghhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh_____________
Hole percent: 12.89

000  iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii______________bbbbbbbbbbbbbbbbb
064  bbbbbbbbbbbbbbbbbbbbcccccccccccccccccccccccccccccccddddddddddddd
128  dddddddddddddddddddddddddddkkkkkk______________ffffffffffffggggg
192  ggggggggggggjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj_____________
Hole percent: 16.02
```

In addition, you need to display the states of all processes whenever context switch happens and when all processes complete. Your output should look something like this:

```
The Process States at Time xxxx
---------+-------+--------+--------+----------
Job#     |Arrival|Running |Waiting |Completion
         |Time   |Duration|Duration|Time
---------+-------+--------+--------+----------
```

```
pid0     |xxx    |xxx     |xxx     |xxx
pid1     |xxx    |xxx     |xxx     |xxx
...       ...     ...      ...      ...
pidN     |xxx    |xxx     |xxx     |xxx
---------+-------+--------+--------+----------
```

When all processes are done, the following information should be print out:

```
Total simulated time units: xxx
Total number of jobs: xxx
Average hole percent: xx.xx
Average waiting Time: xx.xx
```

5 Coding and Testing

It's recommended to use C/C++ for the programming language. It will be fine too if you use any other languages, but in this case you need to demo it to us.

Your executable file should be named as `virtualization`. Your simulator should support 2 command line arguments, one for the choice of algorithm (1: next-fit; 2: worst-fit), and one for input data file name. So the program will be run as follows:

```
prompt> virtualization 1 datafile1.txt
```

6 Submission

There are two parts of submission:

- Part 1: Form a team if you want to work with another teammate. Write down a tackle strategy for this project on how to build the simulator and what language you will use and task assignment for each team member if you work as a team. It is like when you work on a project you need to sit down to think or discuss how to do it. That is what I need for Part 1. It should be written in a ".docx" format, and submitted to Project P1-1 on VLT.

- Part 2: Write a detailed report with the screenshots of the detailed testing results. Each screenshot should include your username and the current time, which show that you did it by yourself. It should be written in a ".docx" format, zipped together with the source code, and submitted to Project P1-2 on VLT.