CSE 232 Spring 2014

Programming Project 09

This assignment is worth 50 points (5.0% of the course grade) and must be **completed and turned in before 11:59 on Monday, April 7, 2014.**

Assignment Overview

This assignment will allow you to create your own classes with private data members and useful methods.

You will implement a class that simulates job processing in computer system in this project. Specifically, you will create a class to simulate a scheduler allocating cpu processing to jobs on a **round-robin** schedule. A round-robin schedule gives each job an equal amount of the cpu and is one of the simplest multiprocessing scheduling techniques.

Given a list (a file) of jobs (each job has four attributes: JobID, Arrival time, Service time, and Finish time), you need to output the jobs in the ascending order of the time they are finished, together with the average amount of time for one job to finish.

Background

In a multiprocessing system, there are two queues. The first is the **arrival queue**. Jobs (any task that requires cpu processing) are placed here when they are first created. When a job is ready to be processed, it is moved from the arrival queue to the **process queue**. At each time unit, a job is pulled from the front of the process queue, run for one time unit, and then, if the job is not completed, the job is placed back on the end of the process queue. Moving from front to back of the process queue is the "round robin" schedule.

People designing operating systems are often concerned with the average **throughput** of the system. The throughput of a job is the amount of time it is in the system: from the moment of arrival until it is finished. In operating system design, designers often concern themselves with keeping the average throughput as low as possible.

When designing a system, the scheduler is often tested by a simulation, much like you will be writing in this program. In the simulation, a list of jobs is entered where each job has a specified **arrival time** and **service time**. The arrival time is when the job is ready to be processed and the service time is how many time units the job requires to be processed. (In real life we may not know a job's service time, but many systems require a good estimate to use the scheduler). After the list of jobs is given, the program simulates the scheduler in a simple loop.

About Time

To keep things simple, everything proceeds in **time units**. When the cpu gets a job, it reduces that job's remaining processing time by 1 time unit. If a job has a service time of 10, then it takes 10 times through the cpu to finish the job.

Project Description / Specification

You need to create two classes: **Scheduler** and **Job**. **Scheduler** class will need a `scheduler.h` file and a `scheduler.cpp` file. **Job** class will need a `job.h` file and `job.cpp` file.

You need to decide what kind of private data you need for each class (note: you'll have to use vector for at least one class).

Job class needs the following member functions:

1. **Job()**. Default constructor.
2. **Job(int, int, int)**. This constructor takes three arguments which are JobID, Arrival_time and Service_time.
3. **int get_job_id()**. This method takes no arguments and returns the job id number.
4. **int get_arrival_time()**. This method takes no arguments and returns the arrival time of the job.
5. **int get_time_left().** This method returns the amount of time left to finish this job.
   a) this is the difference between the starting service time and the number of time units spent on the job so far.
6. **void update_time_left(int).** This method takes an integer argument representing the amount of time spent on a job. That is, it will update the amount of time left for the job using the argument. For example, if the job's time left is 23 time units and the argument is 10, this function will update the time left to 13 time units.
7. **void set_finish_time(int).** This method will set the finish time, the value of the simulation time when the job finished.
8. **int get_finish_time().** This method returns the time when the job finished.

Scheduler class needs the following member functions:

1. **Scheduler().** This constructor will initialize the class private data member.
2. **void load_jobs(string)**. This method takes a string argument which is the file name for the job list file. Each line in the job list file describes one job and the line is formatted with exactly 3integers to a line as follows:

   ```
   Job_ID       Arrival_time        Service_time
   ```

   This member function will open the file, read in all jobs and store them in the private data for future processing. Note: it would be very convenient if you sort the jobs according to their arrival time here in the arrival queue.
3. **void round_robin().** This method implements the round-robin scheduling scheme. You'll have to have a loop to simulate the scheduling of all jobs according to their arrival time and service time. Each iteration of the for loop is one time unit and each job processed decreases by 1 the number of time units until that job finishes. The scheme is as follows:

   ```
   long time_clock = 0;
   While jobs in processing queue:
   1. Check if any new job arrives.
      a) Add jobs from the arrival queue whose arrival_time is ≤ present
         clock_time to the end of the processing list.
   2. for each job from the front of the processing list
      a) update the amount of time left for this job (reduce by 1 time
         unit)
      b) increase the time_clock by 1 unit
      c) If this job is finished,
        i.   set the finish time for the job
       ii.   remove it from the processing list
      d) otherwise, put the job back on the end of the processing queue.
   ```

   Round robin scheduling continues **until both the process queue and arrival queue are empty**

4. **bool finished().** This method checks if ***both*** the processing queue and arrival queue are empty. If so, return true; otherwise, return false.
5. **void display().** This method will display the ordering of jobs according to their finishing time and also the average amount of time for each job to finish.
   a) the **average finish time** is the average of the difference between each job's arrival time and its finish time.
   b) The output format is as follows:

```
-----------------------------------------------------------------
        JobID    Arrival time           Finish time
        13            1                      14
         8            1                      16
        10            3                      20
         3            5                      25

The average amount of time to finish one job is 16.25 time units.
------------------------------------------------------------------
```

Deliverables
Turn in `scheduler.h`, `scheduler.cpp`, `job.h` and `job.cpp` files, using the handin program. Save a copy to your H drive.

List of Files to Download
`main.cpp, jobs.txt`

Notes and Hints:
1. Internet is a good place to find materials about "round-robin" scheduling.
2. You can declare a vector to keep track of jobs. Each row of the vector is one job and each job has four elements: Job ID, Arrival_time, Service_time, Finish_time.
3. Remember that a job can only start after it arrives. This means, the starting time, the time of the present simulation clock, should be larger than the job's arrival time.