Programming Assignments #4

Group project: Priority-based Job Scheduling

Due date: 11/28/2011 Monday 23:59PM

Goals

- Understand important ADTs including priority queues and heaps
- Implement priority queues using heaps
- Understand applications of priority queues
- Learn to use STL queues and vectors

Background

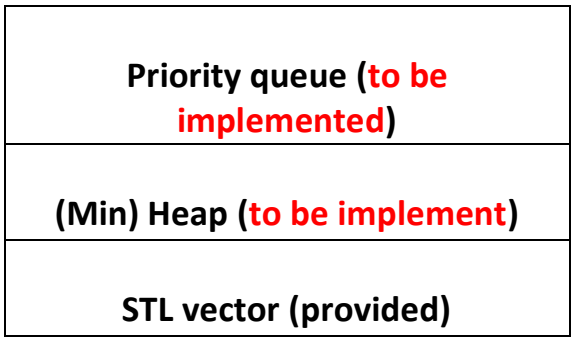
Priority queues, as an important ADT, are widely used in real world. For example, in CPU scheduling systems and disk scheduling systems (e.g., in CPSC341), different jobs (i.e., processes, threads or disk access requests) have different scheduling priorities. Some jobs need to be scheduled earlier than others. One such scheduling policy is priority-based scheduling. Using priority queues to manage jobs can enforce job scheduling based on their priorities. Another example is in network routing (e.g., in CPSC460). Network packets are routed through a path of routers to reach their destinations. Some network packets have high priority, such as the BGP routing information. These packets need to be routed first. Using priority queues can schedule packet routing based on their priorities.

Problem (30 points)

In this project, you need to implement a priority queue to schedule jobs based on their priorities. High priority jobs will be scheduled first. As discussed in class, we have various design choices to implement a priority queue, including BST, sorted list, and heap, etc. We have analyzed their pros and cons. We concluded that heap is an appropriate data structure to implement a priority queue.

That said, in this project you need to implement your priority queue using a heap. We have discussed the use of an array to implement a heap. Here, you will need to use a STL vector to implement the heap. The use of STL vector will ease your work without worrying about capacity issues due to element insertion and deletion. You priority queue is expected to handle hundreds and even thousands of jobs.

The relationships among these ADTs are shown below: Priority queue is built atop Heap and Heap is built atop STL vector.



1. File input

You will read jobs from an input file "job_info.dat" which is a text file. Each line represents a job, with multiple fields, separated by spaces. The job information includes job id, priority value, arrival time and processing time. All the fields are unsigned integer. Arrival time and processing time are in seconds. You can assume arrival times of the jobs in the input file are in non-decreasing order. Job ids are unique. Arrival times may be duplicated, which means multiple jobs may arrive at the same time. Note that a lower priority value indicates a higher priority.

Job_id	priority_value	arrival_time	processing_time
1	2	0	10
2	99	0	20
3	5	3	5

I have provided a job generator program named ***genjoblist*** for you to generate the input file.

You can download the executable by executing the command:

/home/fac/testzhuy/CPSC250/PQ_proj/download

By executing

./genjoblist

You can generate the aforementioned input file with a specified number of jobs. This generator allows for testing your priority queue with different job files.

2. Priority-based job scheduling

Job scheduling is based on their priority values: lower priority values indicate higher priority (this is true in Linux), and need to be scheduled earlier than those with higher priority values. If multiple jobs have the same priority value, break ties by job ids: a job with a lower id is scheduled first.

In order to simulate the priority-based scheduling upon continuous job arrivals, we need to use a simplified discrete-event simulator. You need to schedule jobs in your client/driver program named test.cpp.

Here is the pseudocode segment I have written for the discrete-event simulator. Discrete-event simulation is widely used in the field to study complex dynamic systems. It is OK if you do not understand the logics of the simulator now.

//test.cpp

```
int main() {
        ….
          queue<Job> Q;
          read jobs from the input file into a STL queue Q;
          PriorityQueue PQ;  //your priority queue
          unsigned  int  sys_time = 0; //system clock starting time, in seconds
          do {
                    while (!Q.empty() && the first job's arrival time <= sys_time) { //job arrived
                              remove the job X from Q;
                              insert X into PQ;
                    }
                    if (!PQ.empty()) { //PQ maintains the jobs that have arrived
                              remove the job X with the lowest priority value from PQ;
                              sys_time = sys_time + X's processing time; //emulating job scheduling
                              write this job's info to the specified output file;
                    } else
                              sys_time++; //system clock is ticking
          } while (!PQ.empty() || !Q.empty());

        ….
}
```

3. Output

Your executable must print outputs to a file named **output.txt**. Each job takes a line of text, with the following field in the order, separated by space(s):

Job id, arrival time, finish time, processing time, waiting time.

In particular, waiting time = job's scheduling time – job's arrival time.

The last second line of the output file needs to print the job with the longest waiting time during simulation. You must indicate the job id and its waiting time. The last line of the output file needs to print the average waiting time for the jobs.

4. Deliverables

- A heap (min-heap) class declaration & implementation: myheap.h/.cpp
- A priority queue class declaration & implementation: mypq.h/.cpp. It must provide the following operations:
    - void enqueue(Job x); // insert a job x into the queue

    - Job front(); // return the job with the minimum priority value
    - void dequeue(); // remove the highest-priority job from the queue
    - bool empty(); //return true if empty, false otherwise
- A client/driver program: test.cpp
- Makefile
- README that describes
    - your design
    - strengths/limitations
    - and more importantly, each team member's contributions. All team members must be a contributor in this project.

Use of STL queue and vector

We have used STL queue in the BST's programming assignment. Please refer to that assignment for details.

For STL vector, you can use it as an array without worrying about capacity. The STL vector can dynamically adjust its capacity upon element addition/removal. Here are some basic operations you may use in this project: empty(), clear(), size(), push_back(), pop_back(), etc. Please refer to http://www.cplusplus.com/reference/stl/vector/ for details. The link provides code examples of using the vector.

Assume your job struct is Job. A code segment using them is shown below:

```
#include <vector>   //head file

queue<Job>  Q;  //empty job queue
vector<Job>  V;   //empty job vector
Job x, y;
…
V.push_back(x);  //add job x as the last element in the vector
V[0] = y;   //change the first job to y
V.pop_back(); //remove the last element in the vector
…
```

Step-by-Step

As emphasized many times in class/lab, we need to solve a complex problem step by step. Here are some steps you probably need to follow:

- Play with STL vector to get familiar with its operations
- Design and implement a Heap class using STL vector. The design and implementation of the heap using arrays have been discussed in class

- Test your Heap class extensively
- Design and implement a Priority Queue class based on the Heap class
- Test your Priority Queue in test.cpp with different input files

Team policy

This is a group project. You must form a group of 2-3 members. You cannot take it as an individual project. Each member must contribute to the project.

Submission

- Deadline: **11/28/2011 Monday, 23:59PM**
- Make your files into a tar package, named hw4.tar, which includes **Makefile**, **all source files** (Do not include .o and executable files), and **README file**.

  tar –cvf hw4.tar myheap.h myheap.cpp mypq.h mypq.cpp test.cpp Makefile README

- Howto: (ONLY ONE team member is required to submit your team's project!)
  /home/fac/testzhuy/CPSC250/SubmitHW hw4 hw4.tar

Grading:

- You will receive zero if your program cannot be compiled and executed or if your submission does not include the required files.
- The underlying heap must be functional. If there is any problem for the underlying heap, you will lose 50-100% of total points, depending on the severity of problems.
- The fully functional heap will receive 10 points.
- The fully functional priority queue will receive 15 points. You will lose20-100% points, depending on the severity of problems.
- The output following the required format receives 5 points.