Discrete Event Simulation implementation

Varsha Apte, CS 293

Classes defined by me:

- Simulation
- QueueingSystem
- Event
- Request

STL class templates used

- Priority queue for event list
- Queue for request buffer

Rest of the details can be learnt from comments in the code (listing follows).

Simulation.h

```
Main Simulation class. Comprises of an event list, a current
simulation clock, a queueing system that is being simulated and some
counters and metrics.

************/

#ifndef SIMULATION_H                              // avoid repeated expansion
#define SIMULATION_H

#include<queue>
#include<map>
#include "Event.h"
#include "QueueingSystem.h"

class Simulation {

 private:
  priority_queue<Event> eventList; //Eventlist implemented using STL
                                   //priority queue. This will allows
                                   //us to get the "most imminent
                                   //event" in O(1) time. I think the
                                   //"less than" operator in event is
                                   //reversed because by default STL
                                   //pq is a max-heap, whereas we want
                                   //a min-heap. Please verify this

  double simTime;                  //Simulation clock.
  QueueingSystem sys;              //The system being simulated

  double sumRespTimes;             //Intermediate counters for
                                   //performance metrics
  int customersDeparted;           //How many customers have finished service

  map<double,double> serviceTime;  //<arrival time, service time> map


 public:

  //Constructor, initializes the simulation.
  Simulation() {
    double a, s;

    //Initialize clock and all counters/accummulators

    simTime = 0.0;
    sumRespTimes=0.0;
    customersDeparted =0;

    //create the first arrival event, by reading standard input for an arrival
    //time and a service time
    cin >> a >> s;

    //a=-1 indicates end of input
    if (a != -1) {
      static Event e(0,ARRIVAL,a); //create an event with ID 0, as
                                   //this will be the first event, and
                                   //we know it is an ARRIVAL event at
                                   //time "a" as was read from
                                   //input. This needs to be "static"
                                   //because the Event just created
                                   //should live after function
                                   //exits.

      serviceTime[a] = s;          //Stores the service time 's' of
                                   //the request that arrived at time 'a'

      eventList.push(e);           //Now that event is created, throw
                                   //it into the priority queue.
```

```
    }
  }

  void run(QueueingSystem q);      // Will start the event handling

  //Event handlers
  void arrival_event_handler(Event e);
  void departure_event_handler(Event e);
};
```

QueueingSystem.h

```
/*********
QueueingSystem is the class that captures all aspects of the queueing
system being simulated. A queueing system is defined by

a buffer for queueing requests
the server state
the performance metrics associated with the system

Operations are to enqueue, dequeue and setting the metrics

************/
#ifndef QS_H                              // avoid repeated expansion
#define QS_H

#include<queue>                           //STL queue template
using namespace std;
#include "Request.h"                      //defines Request class
class QueueingSystem {

 private:
  priority_queue<Request> buffer;         //buffer is an STL queue of Requests

  bool serverBusy;                       //1 if busy, 0 if not
  double avgResponseTime;                //average time from joining the
                                         //queue till finishing service
 public:
  void enqueue(Request r);
  Request dequeue();
  Request nextReq();
  bool isBusy();
  double setAvgResponseTime(double avg){return avgResponseTime=avg;}
  QueueingSystem() {serverBusy=false;} //Constructor that initiates
                                         //empty queueing system
};
#endif
```

Request.h

```
/*************
Describes a request in a  queueing system. A request is described by
an ID, which server it is assigned to run on (in case multiple
servers), its arrival time and its service time
******************/

#ifndef REQ_H                              // avoid repeated expansion
#define REQ_H

class Request {

 private:
  long ID;                       // ID to uniquely identify the request
  int serverAssigned;           //Which server, in case of multiple servers

  double arrivalTime;
  double serviceTime;           //Time required to process the request

 public:
  //Various accessor functions
  double getArrivalTime();
  double getServiceTime();
  double getID();
  int getServerAssigned();

  //Overload the less than operator to compare Requests
  bool operator<(const Request& r) const;

  //Constructor: request should always be created with an ID, arrival
  //time, and service time
  Request(int id, double atime, double stime);
};

#endif
```

Event.h

```
/*************************
Event class describes a simulation "event". A simulation "event" is
defined by an ID, a type and a time at which the event is scheduled.
**********************/

#ifndef EVENT_H                                // avoid repeated expansion
#define EVENT_H
#include <iostream>
using namespace std;

//Enumerated type and namespace to use mnemonics for Event types
namespace EventTypeNames { enum eventType {ARRIVAL,DEPARTURE};}
using namespace EventTypeNames;

class Event {

 private:
  int ID;
  eventType type;
  double time;

 public:
  Event() {ID=0; type=ARRIVAL; time=0.0;}  //default constructor
                                           //creates an arrival event
                                           //at time 0
  Event(int id, eventType ty, double ti);
  bool operator<(const Event& e) const;   //overload "<" to compare events
  void print(ostream & outstream=cout) const; //forgot why I needed this
  double getTime() const;                    //accessor method
  int getType() const;                       //accessor method
};


#endif
```

CPP Files

main.cpp

```
/***********

Main file for starting single server queue simulation. Instantiates a
Simulation Object and a Queuing System Object, and starts the
simulation

*****************/

#include <iostream>
#include "Event.h"
#include "QueueingSystem.h"
#include "Simulation.h"
#include "declarations.h"
using namespace std;

/****
Overload << operator so that cout works to print Event objects
Needed for trace/debugging
****/

ostream & operator << (ostream& ostr, const Event& e) {
  e.print(ostr);
  return ostr;
}

//Main - just instantiates objects, starts simulation
int main() {

  Simulation qsim;  //Instantiate a simulation object
  QueueingSystem q;  //Instantiate a Queueing System Object

  qsim.run(q);  //Start Simulation

}
```

QueueingSystem.cpp

```cpp
#include "QueueingSystem.h"

void QueueingSystem::enqueue(Request r) {

  //Enqueue request in buffer, set serverBusy to true as this might be a request
  //coming to an idle server

  buffer.push(r);
  serverBusy = true;

}

Request  QueueingSystem::nextReq() {

  //Returns request at the front of the queue
  //This is the request in service
  //Does not remove the request
  Request r = buffer.top();
  return r;

}

Request  QueueingSystem::dequeue() {

  //Returns request at the front of the queue (which is the request
  // being serviced
  // Removes the request and adjusts server busy state if no more
  // requests in queue

  Request r = buffer.top();
  buffer.pop();
  if (buffer.empty() )
    serverBusy = false;
  return r;

}

bool QueueingSystem::isBusy() {

  return serverBusy;

}
```

Request.cpp

```cpp
#include "Request.h"

//Constructor with argument defaults
Request::Request(int id=0, double atime=0.0, double stime=0.0) {

  ID = id;
  serverAssigned=0;
  arrivalTime=atime;
  serviceTime=stime;
}

//Operator overload, a request r1 is "less than" a request r2 if the
//"age" of r1 in the system is lesser than the "age" of r2.
//This will happen if r1's arrival time is greater (more recent) than
//r2's arrival time

bool Request::operator<(const Request& r) const  {
  return (arrivalTime > r.arrivalTime);
}

double Request::getArrivalTime() {
  return arrivalTime;
}

double Request::getServiceTime() {
  return serviceTime;
}
```

Event.cpp

```
#include "Event.h"
#include "declarations.h"
using namespace std;

Event::Event(int id=0, eventType ty=ARRIVAL, double ti=0.0) {
  ID=id; type=ty; time=ti;
}

//Overload "less than" operator
//Less then is redefined to make it work with the STL priority queue
//which is a max-heap. We want min-heap.
//This will get the needed effect.
bool Event::operator<(const Event& e) const  {
  return (time > e.time);
}
void Event::print(ostream & outstream) const {
  outstream << "ID: " << ID << ", type=" << type << ", time=" << time << endl;
}

double Event::getTime() const { return time;}
int Event::getType() const {return type;}
```

Simulation.cpp

```
/**********
Implements the methods of the Simulation class. Simulation comprises
of the main event processing loop, and the handling of the events.
 *********/

#include "Event.h"
#include "Simulation.h"
#include "QueueingSystem.h"
#include "declarations.h"
using namespace std;

void Simulation::run(QueueingSystem q){
  Event nextEvent;

  //Process events until no more events in the event list
  while (!eventList.empty()) {
    //Pop next event
    nextEvent = eventList.top();
    eventList.pop();

    //Advance simulation clock to time of this event
    simTime = nextEvent.getTime();
    cout << "Simtime: " << simTime << " Event Type= " << nextEvent.getType() << endl;
    //Event handling switch. Call the correct event handler  based on event type.
    if (nextEvent.getType() == ARRIVAL)
      arrival_event_handler(nextEvent);
    else if (nextEvent.getType() == DEPARTURE)
      departure_event_handler(nextEvent);
  }
  //Simulation loop over, now calculate and print the  average response time
  cout << "Avg response time = " <<
    q.setAvgResponseTime(sumRespTimes/customersDeparted) << endl;
}
```

declarations.h

```
/******
File to hold random declarations that don't seem to belong anywhere else
*******/

#include "Event.h"
using namespace std;

ostream & operator << (ostream& ostr, const Event& e);
```

makefile

```
all:  qsim
qsim:  main.o Event.o Request.o  QueueingSystem.o Simulation.o
  g++ -g -o qsim main.o Event.o Request.o QueueingSystem.o Simulation.o
main.o: main.cpp
  g++ -c -g main.cpp
Event.o: Event.cpp Event.h
  g++ -c -g Event.cpp
Request.o: Request.cpp Request.h
  g++ -c -g Request.cpp
QueueingSystem.o: QueueingSystem.cpp QueueingSystem.h
  g++ -c -g QueueingSystem.cpp
Simulation.o: Simulation.cpp     Simulation.h
  g++ -c -g Simulation.cpp
```

To compile all this code, ensure all cpp and header files are in the current directory along with makefile. Then run

```
make
```

This will only compile those files that need compiling.

Server.h

This file is not being used in the current code. I don't remember exactly why I started writing it. It must have been to extend the code to multiple server. It seems to be partially done and **does not compile.**

```
//Generic server of a queueing system
#include "Request.h"

class Server {

 private:

  int ID;
  bool busy;

  double cummulativeBusyPeriod;
  double serviceFinishTime;

  Request* reqInService;

 public:
  Server(int id=0, bool b=0) {
    ID = id;
    busy=b;
    cummulativeBusyPeriod=0;
    serviceFinishTime=MAXDOUBLE;
    reqInService=NULL;
  }

  bool isBusy();
  Request & getReqInService();

  void setReqInService(Request& reqInService);
}
```