Lab 3 Report

CIS 657 Principles of Operating Systems

Name: Xiaoqian Huang

SUID: 878174727

NetId: xhuang62

CIS657 Fall 2019
Assignment Disclosure Form

Assignment #: Lab 3

Name: Xiaoqian Huang

1. Did you consult with anyone other than instructor or TA/grader on parts of this assignment?
   If Yes, please give the details.
   No.

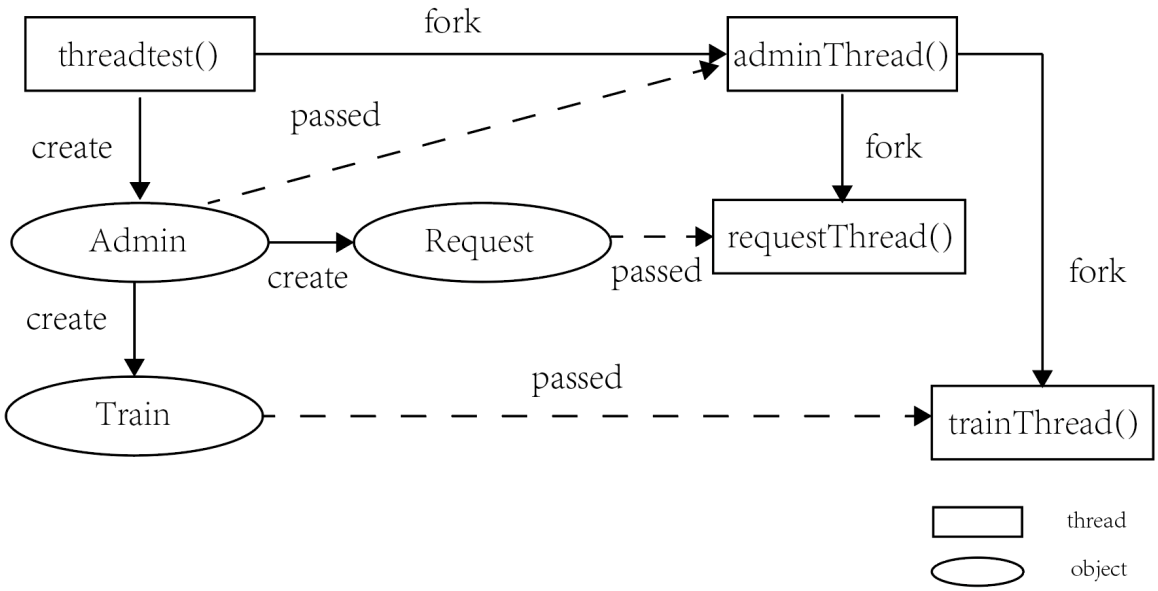
2. Did you consult an outside source such as an Internet forum or a book on parts of this assignment?
   If Yes, please give the details.
   No.

I assert that, to the best of my knowledge, the information on this sheet is true.

Signature:_____Xiaoqian Huang_____ Date : 2019/9/27

1. **Time Consuming**
   1) Analyze the problem, determine specifications, create the design:1 hour.
   2) Implement the design: 8 hours (first version of implementation).
   3) Test/debug the program: 17 hours (make the program compiled and run, fix logical errors, fix segmentation fault errors, print and analyze large amount of output data to debug).
2. **Design/Solution for each requirement**

   First of all, after I analyzed the problem carefully, I designed the structure as below:



There are 3 threads defined at threadtest.cc: adminThread, trainThread and requestThread.

- threadtest creates an Admin object and forks the adminThread() to start simulation.
- adminThread first creates 6 trains according to the stored strings, and then forks the threads for them. Then it starts simulation from 6:00 a.m. to 22:00 p.m. During this time, 5 requestThread will be forked at the beginning of every 10 minutes to process request though the Admin. Then the thread will yield and give the CPU to trainThread and reservationThread. When the CPU comes back, adminThread will record the data regarding request. At the end of every minute, all the trains will be synchronized the time. Then it will yield again to trainThread.
- trainThread runs a while loop within the simulation time and call the train to operate according to the time. Thread will yield the CPU at the end of every loop.
- reservationThread generates and acquires the processed result through Admin object and then sleep until being waked by a train. Then it will add request to onboard list when request is served and sleep again. When the service is finished, the train will wake reservationThread again and it will remove the request from onboard list then finishing the thread.

To maintain the information and process the data, I created 3 objects: Admin, Request and Train which are declared and defined at Admin.[h|cc], Request.[h|cc], Train.[h|cc] respectively.

- Admin serves as the management system for maintaining all the simulation data and operations including generating trains and requests, scheduling trains, processing requests and recording all the required data.
- Request stores all the information relating to the specific request and provides operations to get and set data.
- Train stores all the information relating to that train, and providing operations including operating trains, finding available seats, assigning seats, recording data and printing out summaries. Of this, when operating trains, the current time will be compared with the next schedule and then processing getting on and off passengers and updating schedule according to the schedule.

After modelling the above threads and objects, the simulation will run as required.

Specifically, there are some assumptions and details need some explanations.

1. For the data which stores the train information, the data format length of time is fixed, and the data format is:

   [Train Name] [Station1 ArrivalTime] .. [Station20 ArrivalTime] [Business fair, Coach fair]

   Specially, I assumed that the departure time of a train is exactly 10 minutes after the scheduled arrival time.

2. Since the stations within a region is fixed for a certain time, the id of each station is fixed and didn't store the name. It can be easily mapped in future use. So all the data structure regarding to station information, the index of them is exact the id.
3. For the seat id, I assumed that 0-19 is for business class, and 20-59 is for coach class.
4. To grant a request:

- First the train schedule of the departure station and destination station should satisfy the request.
  1) The departure time of train cannot earlier than current time.
  2) In my assumption, the departure time of a request is the latest time it can accept to leave apart. So the train should arrive the departure station before the departure time of request.
  3) The departure station time of the train should earlier than the destination station time, otherwise the direction is wrong.

  Only when the time match, the request will be attempted to find seats.
- To maintain the seat situation of each station, the train will have a Bitmap list corresponding to each station. When a request is asking for a seat, the bitmap of the first station will be examined and find the available seat, until the previous station of destination station, the availability will be verified. Only if that seat in all the stations are available, the seat is treated valid to that request. If there are enough valid seats for that request, it can be granted.

- After finding all the available trains that satisfied the above conditions, one of them will be assign to the request randomly.

For the required output:

- Request per simulation time
  - o Granted and refused

In adminThread there will be two variables to store the previous granted and refused request number. After obtaining the CPU back after yielding to reservationThread, the current granted and refused requests will be check and update.

- Train operation information per simulation time
  - o # of itinerary, # of passengers for boarding at a station of running trains at the time

This information will be stored to each station's list when the request is first granted to the train. And it will be printed out when the passengers boarding.

- o # of itinerary, # of passengers for getting off at a station at the time

This information will be stored to each station's list when the request is first granted to the train. And it will be printed out when the passengers leaving.

- Simulation summary
  - o Total # of requests

The total requests are maintained by Admin and updated when every request generated.

- o Total # of granted requests

The total granted requests are maintained by Admin and updated when every request generated.

- o Train operation summary

- Total served itinerary

Total served itinerary is maintained and updated by the Train when a request is granted.

- Total passengers

Total passengers is maintained and updated by the Train when a request is granted.

- Busiest section

I assumed that the busiest section is the time when the number of passengers on the train is max (i.e. new passengers take on and old passengers not get off yet). So I update the busiest time, busiest passenger numbers and busiest station every time the passengers are getting onboard.

Above all, the whole program to simulate the process is completely designed.

3. Implementation of the solution

In threadtest.cc

```
/* threadtest.cc
  There are 3 threads defined at threadtest.cc: adminThread, trainThread and requestThread.
 -        threadtest creates an Admin object and forks the adminThread() to start simulation.
 -        adminThread first creates 6 trains according to the stored strings, and then forks the
 threads for them. Then it starts simulation from 6:00 a.m. to 22:00 p.m. During this time,
 5 requestThread will be forked at the beginning of every 10 minutes to process request
 though the Admin. At the end of every minute, all the trains will be synchronized the time.
 -        trainThread runs a while loop within the simulation time and call the train to operate
 according to the time.
 -        reservationThread generates and acquires the processed result through Admin object, add
 request to onboard list when request is served and remove it when the service is finished.
*/

#include "kernel.h"
#include "main.h"
#include "thread.h"
#include "Admin.h"
#include "Request.h"
#include "Train.h"

void
ReservationThread(Admin *admin) {

 Request *newReq = admin->processReservation(kernel->currentThread);

 if(newReq == NULL){
  kernel->currentThread->Finish();
 }

 kernel->interrupt->SetLevel(IntOff);
 kernel->currentThread->Sleep(FALSE);

 // handling request :onboard
 admin->addOnBoard(newReq);
 kernel->currentThread->Sleep(FALSE);

 // handling request :offboard
 admin->removeOnBoard(newReq);
```

```
 kernel->currentThread->Finish();
}

void
TrainThread(Train *Train) {
 while (Train->getCurrentTime() < 22 * 60) {
  Train->operateTrain(1);
  kernel->currentThread->Yield();
 }
}

void
AdminThread(Admin *admin)
{
 int requestNum = 0; // total request
 int grantedRequestNum = 0, refusedRequestNum = 0;
 int previousGrantedNum = 0;
 int previousRefusedNum = 0;

 admin->createTrains();

 // create train threads
 List<Train*>* trainList = admin->getTrainList();
 for (int i = 0; i < trainList->NumInList(); i++) {
  Thread *trainThread = new Thread("Train Thread");

  trainThread->Fork((VoidFunctionPtr)TrainThread, (Train*)trainList->getItem(i));
  admin->addTrainThread(trainThread);
 }

 for (int hour = 6; hour < 22; hour++) {
  for (int minute = 0; minute < 60; minute++) {
   admin->setCurrentTime(hour*60 + minute); // synchronize time
   if (minute % 10 == 0) {
    printf("*****[%02d:%02d]*****\n", hour, minute);
    for (int n = 0; n < 5; n++) { // generate 5 requests
     Thread *newReq = new Thread("Reservation Thread");
     newReq->Fork((VoidFunctionPtr)ReservationThread, (Admin*)admin);
    }

    kernel->currentThread->Yield(); // yield to other threads
```

```
      cout << "# Granted Request: " << admin->getGrantedNum() - previousGrantedNum << "\n";
      cout << "# Refused Request: " << admin->getRefusedNum() - previousRefusedNum << "\n";
      previousGrantedNum = admin->getGrantedNum();
      previousRefusedNum = admin->getRefusedNum(); // update data
    }

    admin->synchronizeTrains();
    kernel->currentThread->Yield();
  }
 }

 cout << "\n-------------------------------------------------------------------------------\n\n";
 cout << "Simulation Summary: \n";
 cout << "# Request: " << admin->getRequestNum() << "\n";
 cout << "# Granted Request: " << admin->getGrantedNum() << "\n";

 cout << "\n";

 admin->summaryTrains();
 cout << "\n* Assumption: The busiest section is the time when the number of passengers on the
train is max (i.e. new passengers take on and old passengers not get off yet). \n";
}

void
ThreadTest()
{
 srand((unsigned int)time(0));
 Admin *admin = new Admin();
 Thread *t = new Thread("Admin thread");
 t->Fork((VoidFunctionPtr)AdminThread, (Admin*) admin);
}
```

In Admin.h

```
/* Admin.h
  -        Admin serves as the management system for maintaining all the simulation
   data and operations including generating trains and requests, scheduling
   trains, processing requests and recording all the required data.
*/

#pragma once

#ifndef ADMIN_H
#define ADMIN_H

#include "Request.h"
#include "Train.h"
#include "kernel.h"

#include "main.h"
#include "thread.h"
#include "../lib/utility.h"

class Admin {
public:
  Admin() { // constructor
    _grantedRequestList = new List<Request*>();
    _refusedRequestList = new List<Request*>();
    _onBoardRequestList = new List<Request*>();
    _trainThreadList = new List<Thread*>();
  }
  void simulating();
  void createTrains();
  void setCurrentTime(int time);
  Request* processReservation(Thread *resThread); // reservation thread
  int getRequestNum();
  int getGrantedNum();
  int getRefusedNum();
  int getTrainNum();
  void synchronizeTrains();
  void summaryTrains();
  List<Train*>* getTrainList();
  void addTrainThread(Thread* thread);
  void addOnBoard(Request *newReq);
```

```
  void removeOnBoard(Request *newReq);

private:
  int _translateTime(int hour, int minute);

  List<Train*>* _trainList;
  List<Request*>* _grantedRequestList;
  List<Request*>* _refusedRequestList;
  List<Request*>* _onBoardRequestList;
  List<Thread*> *_trainThreadList;
  int _requestNum = 0;
  int _timer = 0;
};

#endif // ADMIN_H
```

In Admin.cc

```
#include "Admin.h"

void Admin::simulating() {
}

int Admin::_translateTime(int hour, int minute) {
  return hour * 60 + minute;
}

void Admin::createTrains() {
  _trainList = new List<Train*>();
  List<char*> *fileContent = new List<char*>();
  // Data format: [Train Name] [Station1 ArrivalTime] .. [Station20 ArrivalTime] [Business fair, Coach fair]
  // The data format length of time is fixed.
  fileContent->Append("1 06:00 07:20 08:20 09:10 09:33 10:40 11:30 12:11 13:29 14:20 15:15 15:35 16:43 17:24 18:20 19:45 20:23 20:59 21:54 22:00 20,10");
  fileContent->Append("2 22:00 21:40 20:40 20:00 19:20 18:00 18:23 17:39 16:23 15:45 15:00 14:24 13:56 12:38 11:10 10:00 09:33 08:24 07:12 06:00 25,15");
  fileContent->Append("3 --:-- --:-- --:-- --:-- --:-- --:-- --:-- 06:00 07:20 08:20 09:10 09:33 10:40 11:30 12:11 13:29 14:20 15:15 15:35 16:43 50,20");
  fileContent->Append("4 --:-- --:-- --:-- --:-- --:-- --:-- --:-- 22:00 21:40 20:40 20:00 19:20 18:00 18:23 17:39 16:23 15:45 15:00 14:24 13:56 15,05");
  fileContent->Append("5 --:-- --:-- --:-- --:-- --:-- 06:00 07:20 08:20 09:10 09:33 10:40 11:30 12:11 13:29 14:20 15:15 15:35 16:43 17:24 18:20 20,10");
  fileContent->Append("6 --:-- --:-- --:-- --:-- --:-- --:-- --:-- --:-- --:-- --:-- 17:24 18:20 22:00 21:54 20:59 20:23 19:45 16:43 15:35 15:15 40,20");

  int num = fileContent->NumInList();
  for (int i = 0; i < num; i++) {
    char* str = fileContent->RemoveFront();
    Train *newTrain = new Train(i, str);
    _trainList->Append(newTrain);
  }
}

Request* Admin::processReservation(Thread *resThread) { //reservation thread
  _requestNum++;
  int reservationId = _requestNum;
  Request *newRequest = new Request(_requestNum, _timer);
  ListIterator<Train*> iter(_trainList);
  List<Train*> *availableList = new List<Train*>; // need to delete
```

```
 int start, end;


 while (iter.IsDone() == FALSE) {
  // Assumption: the departure time of a request is the latest time it can accept to leave apart
  if (iter.Item()->getArrivalTime(newRequest->getDepartureStation()) >= _timer &&
   iter.Item()->getArrivalTime(newRequest->getDepartureStation()) <=
newRequest->getDepartureTime() &&
   iter.Item()->getArrivalTime(newRequest->getDestinationStation()) >
iter.Item()->getArrivalTime(newRequest->getDepartureStation())) { // time match
   if (iter.Item()->testSeat(newRequest) == true) {
    availableList->Append(iter.Item());
    cout << "[" <<iter.Item()->getName() << "] ";
   }
  }
  iter.Next();
 }
 cout << availableList->NumInList() << " trains can be granted.";
 if (availableList->NumInList() > 0) {
  int selectedTrain = rand() % availableList->NumInList(); // randomly select a qualified train
  selectedTrain = availableList->getItem(selectedTrain)->getTrainId();
  newRequest->setGrantedTrainId(selectedTrain);
  _grantedRequestList->Append(newRequest);
  _trainList->getItem(selectedTrain)->addGrantedRequest(newRequest->getDepartureStation(),
newRequest->getDestinationStation(), resThread); // add the request to the train granted list
  cout << " - Request " << newRequest->getId() << " is granted to train[" <<
_trainList->getItem(newRequest->getGrantedTrainId())->getName() << "] with " <<
newRequest->getPassengerNum() << " passengers. \n";
  cout << "   Class: " << newRequest->getClass() << ", from " <<
newRequest->getDepartureStation() << " to " << newRequest->getDestinationStation() << "\n";
 }
 else {
  _refusedRequestList->Append(newRequest);
  cout << " - Request " << newRequest->getId() << " is refused with " <<
newRequest->getPassengerNum() << " passengers. \n";
  cout << "   Class: " << newRequest->getClass() << ", from " <<
newRequest->getDepartureStation() << " to " << newRequest->getDestinationStation() << "\n";
  return NULL;
 }
 // assign seat
 List<int> *seatIdList =
_trainList->getItem(newRequest->getGrantedTrainId())->assignSeat(newRequest);


 newRequest->setSeatId(seatIdList); // record the assign result in request object
```

```
  //delete availableList;
  return newRequest;
}


void Admin::addOnBoard(Request *newReq){
  _onBoardRequestList->Append(newReq);
}

void Admin::removeOnBoard(Request *newReq){
  _onBoardRequestList->Remove(newReq);
}

void Admin::setCurrentTime(int time) {
  _timer = time;
}

int Admin::getRequestNum() {
  return _requestNum;
}

int Admin::getGrantedNum() {
  return _grantedRequestList->NumInList();
}

int Admin::getRefusedNum() {
  return _refusedRequestList->NumInList();
}

int Admin::getTrainNum() {
  return _trainList->NumInList();
}

void Admin::synchronizeTrains() {
  for (int i = 0; i < _trainList->NumInList(); i++) { // synchronize time for trains
    _trainList->getItem(i)->setCurrentTime(_timer);
  }
}

void Admin::summaryTrains() {
  for (int i = 0; i < _trainList->NumInList(); i++) {
```

```
    _trainList->getItem(i)->summary(); // print summary information
  }
}

List<Train*>* Admin::getTrainList() {
  return _trainList;
}

void Admin::addTrainThread(Thread* thread) {
  _trainThreadList->Append(thread);
}
```

In Train.h

```
/* Train.h
  -        Train stores all the information relating to that train, and providing
  operations including operating trains, finding available seats, assigning
  seats, recording data and printing out summaries. Of this, when operating
  trains, the current time will be compared with the next schedule and then
  processing getting on and off passengers and updating schedule according
  to the schedule.
*/

#pragma once

#ifndef TRAIN_H
#define TRAIN_H

#include <stdlib.h>
#include <iostream>
#include <string>
#include "Request.h"
#include "thread.h"
#include "kernel.h"
#include "main.h"
#include "../lib/list.h"
#include "../lib/bitmap.h"

class Train {
public:
  Train(int id, char *data){ // constructor
    _id = id;
    _data = data;
    _setAttri();
  }

  void operateTrain(int reserved);
  List<Bitmap*>* getSeatList(); // get the list of available seat
  int getBusFare();
  int getCoachFare();
  int getArrivalTime(int stationId);
  int getBusSeatNum(int stationId);
  int getCoachSeatNum(int stationId);
```

```
  void addGrantedRequest(int departureStationId, int destinationStationId, Thread
*grantedRequest); // store the unhandled requests
  bool testSeat(Request *req); // test if there are seats available
  List<int>* assignSeat(Request *req); // assign seat
  List<int>* findSeat(Request *req);
  void setCurrentTime(int currentTime);
  int getCurrentTime();
  void summary();
  int getTrainId();
  int getName();

private:
  void _setAttri();
  bool _validate(int startStationId, int endStationId, int seatNum);
  void _checkSchedule();
  void _updateBusiestInfo();

  char *_data;
  int _id;
  int _name;
  int _route[20];
  List<Bitmap*> *_seats;
  int _nextStation;
  int _lastStation;
  int _nextArrivalTime;
  int _lastArrivalTime;
  int _BusFare, _coachFare;
  int _totalRequestNum = 0;
  List<List<Thread*>*> *_departureRequestList; // each station maintain 1 list, used to take on
  List<List<Thread*>*> *_destintationRequestList; // each station maintain 1 list, used to take off
  int _currentTime = 0;
  int _passangerInNum[20] = {0};
  int _passangerOutNum[20] = {0};
  int _currentPassangerNum = 0;
  int _busiestTime = 0;
  int _busiestStation = -1;
  int _busiestPassengerNum = 0;
};

#endif // RESERVATION_H
```

In Train.cc

```
# include "Train.h"

void Train::_setAttri() {
 char *str = _data;

 _departureRequestList = new List<List<Thread*>*>();
 _destintationRequestList = new List<List<Thread*>*>();
 _seats = new List<Bitmap*>();

 for (int i = 0; i < 20; i++) { // initialize
  List<Thread*> *temp = new List<Thread*>();
  _departureRequestList->Append(temp);
  List<Thread*> *temp2 = new List<Thread*>();
  _destintationRequestList->Append(temp2);
  Bitmap *temp3 = new Bitmap(60);
  _seats->Append(temp3);
 }

 int fare = 0;
 int name = 0;
 while (*str != ' ') {
  name = name * 10 + ((int)(*str - '0'));
  str++;
 }
 _name = name;
 std::cout << "Train " << _name << ": ";

 str++;// for space

 for (int i = 0; i < 20; i++) {
  int hour = 0;
  int minute = 0;
  if(*str != '-'){
   hour = ((int)(*str - '0')) * 10 + ((int)(*(str + 1) - '0'));
   minute = ((int)(*(str + 3) - '0')) * 10 + ((int)(*(str + 4) - '0'));
  }
  if(hour == 0 && minute == 0){
   printf("--:-- ", hour,minute);
  }else{
   printf("%02d:%02d ", hour,minute);
```

```
    }
    _route[i] = hour * 60 + minute;
    str = str + 6;
  }

  while (*str != ',') {
    fare = fare * 10 + ((int)(*str - '0'));
    str++;
  }
  _BusFare = fare;
  fare = 0;
  std::cout << "$" << _BusFare << "/";

  str++; // for ','

  while (*str != '\0') {
    fare = fare * 10 + ((int)(*str - '0'));
    str++;
  }
  _coachFare = fare;
  std::cout << "$" << _coachFare << "\n";
  _checkSchedule();

}

void Train::operateTrain(int reserved){ // run the train
  _checkSchedule();
  kernel->interrupt->SetLevel(IntOff);
  if (_currentTime == _nextArrivalTime) {
    for (int i = 0; i < _departureRequestList->getItem(_nextStation)->NumInList(); i++) { // get in,
wake up all the request
      kernel->scheduler->ReadyToRun(_departureRequestList->getItem(_nextStation)->getItem(i));
    }
    cout << "Train[" << _name << "] - ";
    cout << "departure station: " <<  _nextStation << " at time ";
    printf("%02d:%02d \n", _currentTime/60, _currentTime%60);
    cout << "# Itinerary: " << _departureRequestList->getItem(_nextStation)->NumInList() << "\n";
    cout << "# Passenger boarding: " << _passangerInNum[_nextStation] << "\n";
    _seats->getItem(_nextStation)->Print();
    _totalRequestNum += _departureRequestList->getItem(_nextStation)->NumInList();
    _currentPassangerNum += _passangerInNum[_nextStation];
```

```
    _updateBusiestInfo();
    _lastStation = _nextStation; // record current station
    _lastArrivalTime = _nextArrivalTime;
    _departureRequestList->getItem(_nextStation)->RemoveAll();
   }
   else if(_currentTime == _lastArrivalTime+10){ // time to leave, get out. The departure time is 10
minute after the arrival time
    for (int i = 0; i < _destintationRequestList->getItem(_lastStation)->NumInList(); i++) { // get in
     kernel->scheduler->ReadyToRun(_destintationRequestList->getItem(_lastStation)->getItem(i));
    }
    cout << "Train[" << _name << "] - ";
    cout << "destination station: " <<  _lastStation << " at time ";
    printf("%02d:%02d \n", _currentTime/60, _currentTime%60);
    cout << "# Passenger getting off: " << _passangerOutNum[_lastStation] << "\n";
    _seats->getItem(_lastStation)->Print();
    _currentPassangerNum -= _passangerOutNum[_lastStation];
    _destintationRequestList->getItem(_lastStation)->RemoveAll();
   }
}

List<Bitmap*>* Train::getSeatList() {
  return _seats;
}

int Train::getBusSeatNum(int stationId) {
  int num = 0;
  for (int i = 0; i < 20; i++) {
    if (_seats->getItem(stationId)->Test(i) == false) {
      num++;
    }
  }
  return num;
}

int Train::getCoachSeatNum(int stationId) {
  int num = 0;
  for (int i = 20; i < 60; i++) {
    if (_seats->getItem(stationId)->Test(i) == false) {
      num++;
    }
  }
```

```
 return num;
}

int Train::getBusFare() {
 return _BusFare;
}

int Train::getCoachFare() {
 return _coachFare;
}

int Train::getArrivalTime(int stationId) {
 return _route[stationId];
}

void Train::addGrantedRequest(int departureStationId, int destinationStationId, Thread
*grantedRequest) {
 _departureRequestList->getItem(departureStationId)->Append(grantedRequest);
 _destintationRequestList->getItem(destinationStationId)->Append(grantedRequest);
}

List<int>* Train::assignSeat(Request *req) {
 int start, end;
 List<int> *seatIdList = findSeat(req);
 if(seatIdList == NULL){
  return NULL;
 }
 List<int> *assignedSeatIdList = new List<int>;
 if (req->getDepartureStation() > req->getDestinationStation()) { // reverse direction
  start = req->getDestinationStation() + 1;
  end = req->getDepartureStation();
 }
 else {
  start = req->getDepartureStation();
  end = req->getDestinationStation() - 1;
 }

 for(int j = 0; j < req->getPassengerNum(); j++) {
  for(int i = start; i < end + 1; i++){
   _seats->getItem(i)->Mark(seatIdList->getItem(j));
  }
```

```
    assignedSeatIdList->Append(seatIdList->getItem(j));
  }

  _passangerInNum[req->getDepartureStation()] += req->getPassengerNum(); // record passenger
num
  _passangerOutNum[req->getDestinationStation()] += req->getPassengerNum();

  //delete seatIdList;
  return assignedSeatIdList;
}

bool Train::testSeat(Request *req) {
  List<int> *seatIdList = findSeat(req);
  if (seatIdList != NULL) { // available
    //delete seatIdList;
    return true;
  }
  //delete seatIdList;
  return false;
}

List<int>* Train::findSeat(Request *req) {
  int start, end;
  bool hasSeat = TRUE;
  int seatId;
  List<int> *seatIdList = new List<int>();
  int count = 0;
  if (req->getDepartureStation() > req->getDestinationStation()) { // reverse direction
    start = req->getDestinationStation() + 1;
    end = req->getDepartureStation();
  }
  else {
    start = req->getDepartureStation();
    end = req->getDestinationStation() - 1;
  }

  if (req->getClass() == 0) { // business
    for (int j = 0; j < 20; j++) {
      if (_seats->getItem(start)->Test(j) == false) { // find the available seat of the first station
        if (_validate(start + 1, end, j) == true) {
          count++;
```

```
        seatIdList->Append(j);
       }
      }
     }
    }
   else { // coach
    for (int j = 20; j < 60; j++) {
     if (_seats->getItem(start)->Test(j) == false) {
      if (_validate(start + 1, end, j) == true) {
       count++;
       seatIdList->Append(j);
      }
     }
    }
   }

   if(count >= req->getPassengerNum()){
    return seatIdList;
   }

   return NULL;
}

bool Train::_validate(int startStationId, int endStationId, int seatNum) { // validate the remain
stations
   for (int i = startStationId; i < endStationId + 1; i++) {
    if (_seats->getItem(i)->Test(seatNum) == true) {
     return false;
    }
   }
   return true;
}

void Train::setCurrentTime(int currentTime) {
   _currentTime = currentTime;
}

void Train::_checkSchedule() {
   int min = 1321;
   int stationId = -1;
   for (int i = 0; i < 20; i++) {
```

```
    int time = _route[i];
    if (time >= _currentTime) {
      if (time <= min) {
        min = time;
        stationId = i;
      }
    }
  }
  _nextArrivalTime = min;
  _nextStation = stationId;
}

void Train::_updateBusiestInfo() {
  if (_busiestPassengerNum < _currentPassangerNum) {
    _busiestTime = _currentTime;
    _busiestStation = _nextStation;
    _busiestPassengerNum = _currentPassangerNum;
  }
}

void Train::summary() {
  cout << "Train " << _name << " Summary: \n";
  cout << "# Total served itinerary: " << _totalRequestNum << "\n";
  int pNum = 0;
  for (int i = 0; i < 20; i++) {
    pNum += _passangerInNum[i];
  }
  cout << "# Total passengers: " << pNum << "\n";
  cout << "Busiest Section: ";
  printf("%02d:%02d",_busiestTime / 60, _busiestTime % 60);
  cout << " at station #" << _busiestStation << " with " << _busiestPassengerNum << " passengers on
the train.\n";
}


int Train::getCurrentTime(){
  return _currentTime;
}

int Train::getTrainId(){
  return _id;
```

```
}

int Train::getName(){
  return _name;
}
```

In Request.h

```
/* Request.h
  -        Request stores all the information relating to the specific request
  and provides operations to get and set data.
*/

#pragma once

#ifndef REQUEST_H
#define REQUEST_H

#include <stdlib.h>
#include "../lib/list.h"

class Request {
public:
 Request(int id, int currentTime) { // constructor
  _id = id;
  _generateRequest(currentTime);
 }

 int getId();
 int getDepartureStation();
 int getDestinationStation();
 int getClass();
 int getPassengerNum();
 void setGrantedTrainId(int id);
 void setSeatId(List<int> *_seatIdList);
 int getGrantedTrainId();
 int getDepartureTime();
 int printInfo();

private:
 void _generateRequest(int currentTime);

 int _id;
 int _departureStation = -1; // 0-19
 int _destinationStation = -1;
 int _class = -1; // 0 for business, 1 for coach
 int _passenger_Num = 0; // 1-8
 int _departureTime = 0;
```

```
  int _grantedTrainId = -1; // when -1, the request is not granted
  List<int> *_seatIdList;
};

#endif // REQUEST_H
```

In Request.h

```
#include "Request.h"

void Request::_generateRequest(int currentTime) {
 _departureStation = rand() % 20; // 0-19
 _destinationStation = rand() % 20;
 while (_destinationStation == _departureStation) {
  _destinationStation = rand() % 20;
 }
 _class = rand() % 2;
 _passenger_Num = rand() % 8 + 1;
 _departureTime = rand() % (60 * 22 - currentTime) + currentTime; // no later than current time
 cout << "New Request: from " << _departureStation << " to " << _destinationStation << " pNum: "
  << _passenger_Num << " class: " << _class << " departureTime: ";
  printf("%02d:%02d \n", _departureTime/60, _departureTime%60);
}

int Request::getId() {
 return _id;
}

int Request::getDepartureStation() {
 return _departureStation;
}

int Request::getDestinationStation() {
 return _destinationStation;
}

int Request::getClass() {
 return _class;
}

int Request::getPassengerNum() {
 return _passenger_Num;
}

int Request::getDepartureTime(){
 return _departureTime;
}
```

```
void Request::setGrantedTrainId(int id) {
  _grantedTrainId = id;
}

void Request::setSeatId(List<int> *seatIdList) {
  _seatIdList = seatIdList;
}

int Request::getGrantedTrainId() {
  return _grantedTrainId;
}

int Request::printInfo(){
  cout << "@Request " << _id << "from " << _departureStation << " to " << _destinationStation <<
  "with " << _passenger_Num << " passengers assigned to train " << _grantedTrainId << "\n";
}
```

Additional code added:

List.h

```
T getItem(int index); // get the nth item in the list
```

List.cc

```
//-----------------------------------------------------------------------
// List::getItem
//      Get the nth item i the list.
//-----------------------------------------------------------------------

template <class T>
T
List<T>::getItem(int index) {
  ListIterator<T> *iterator = new ListIterator<T>(this);
  for (int i = 0; i < this->numInList; i++) {
    if (index == i) {
      return iterator->Item();
    }
    iterator->Next();
  }
  delete iterator;
  return NULL;
}
```

4. Testing

1. Method to run my tests.
   1) Run:
      [result mode] ./**nachos -K** => only required result

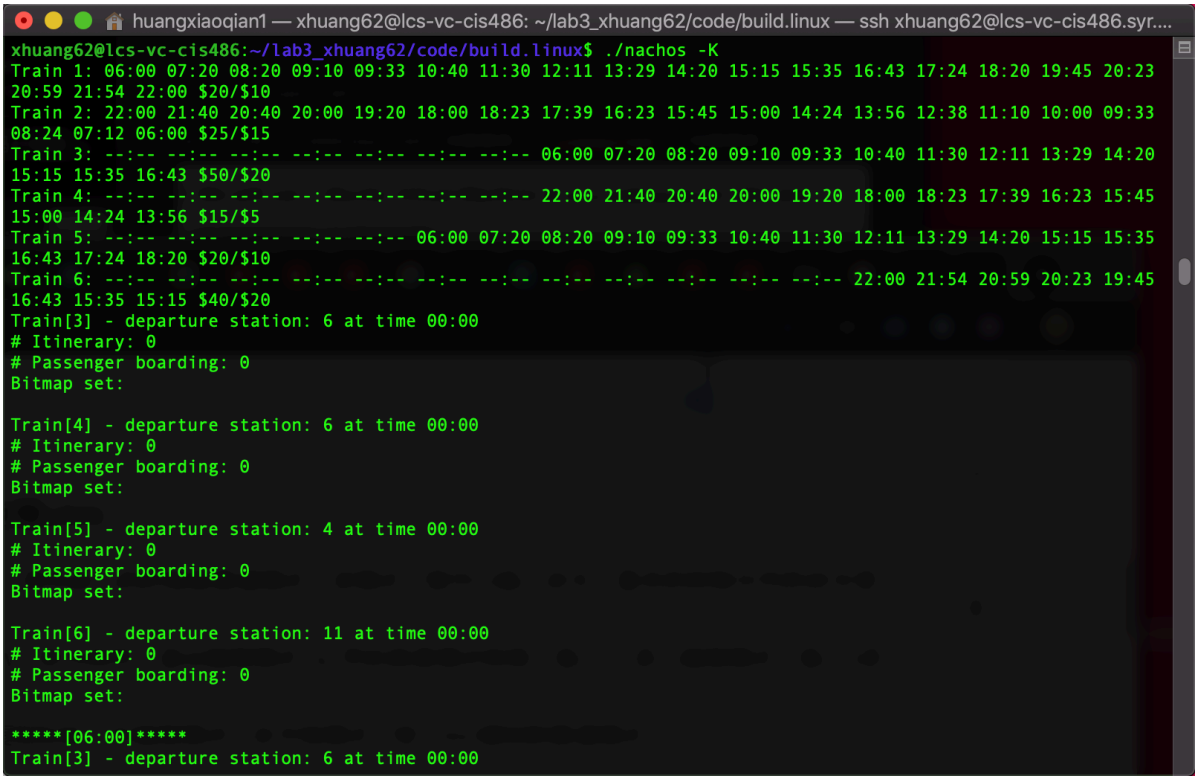
   2) The data of train schedule in the test is:

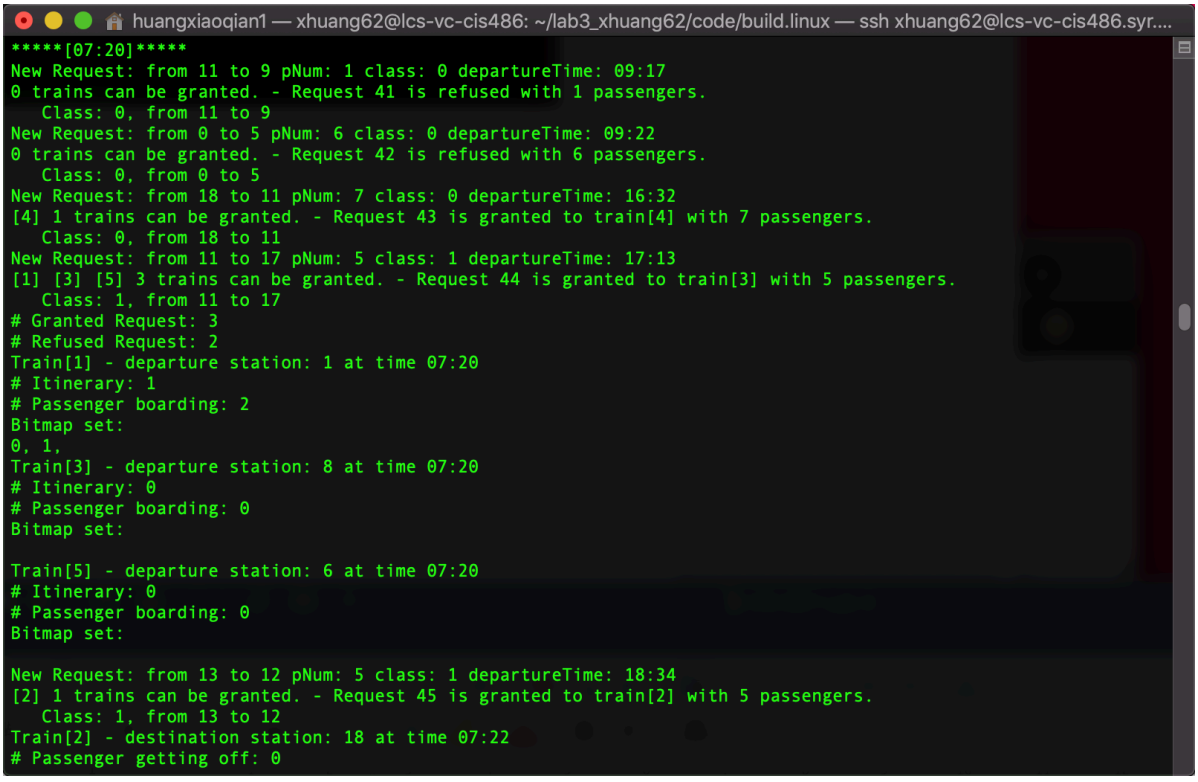
Train 1: 06:00 07:20 08:20 09:10 09:33 10:40 11:30 12:11 13:29 14:20 15:15 15:35 16:43 17:24 18:20 19:45 20:23 20:59 21:54 22:00 $20/$10
Train 2: 22:00 21:40 20:40 20:00 19:20 18:00 18:23 17:39 16:23 15:45 15:00 14:24 13:56 12:38 11:10 10:00 09:33 08:24 07:12 06:00 $25/$15
Train 3: --:-- --:-- --:-- --:-- --:-- --:-- --:-- 06:00 07:20 08:20 09:10 09:33 10:40 11:30 12:11 13:29 14:20 15:15 15:35 16:43 $50/$20
Train 4: --:-- --:-- --:-- --:-- --:-- --:-- --:-- 22:00 21:40 20:40 20:00 19:20 18:00 18:23 17:39 16:23 15:45 15:00 14:24 13:56 $15/$5
Train 5: --:-- --:-- --:-- --:-- --:-- 06:00 07:20 08:20 09:10 09:33 10:40 11:30 12:11 13:29 14:20 15:15 15:35 16:43 17:24 18:20 $20/$10
Train 6: --:-- --:-- --:-- --:-- --:-- --:-- --:-- --:-- --:-- --:-- --:-- --:-- --:-- 22:00 21:54 20:59 20:23 19:45 16:43 15:35 15:15 $40/$20

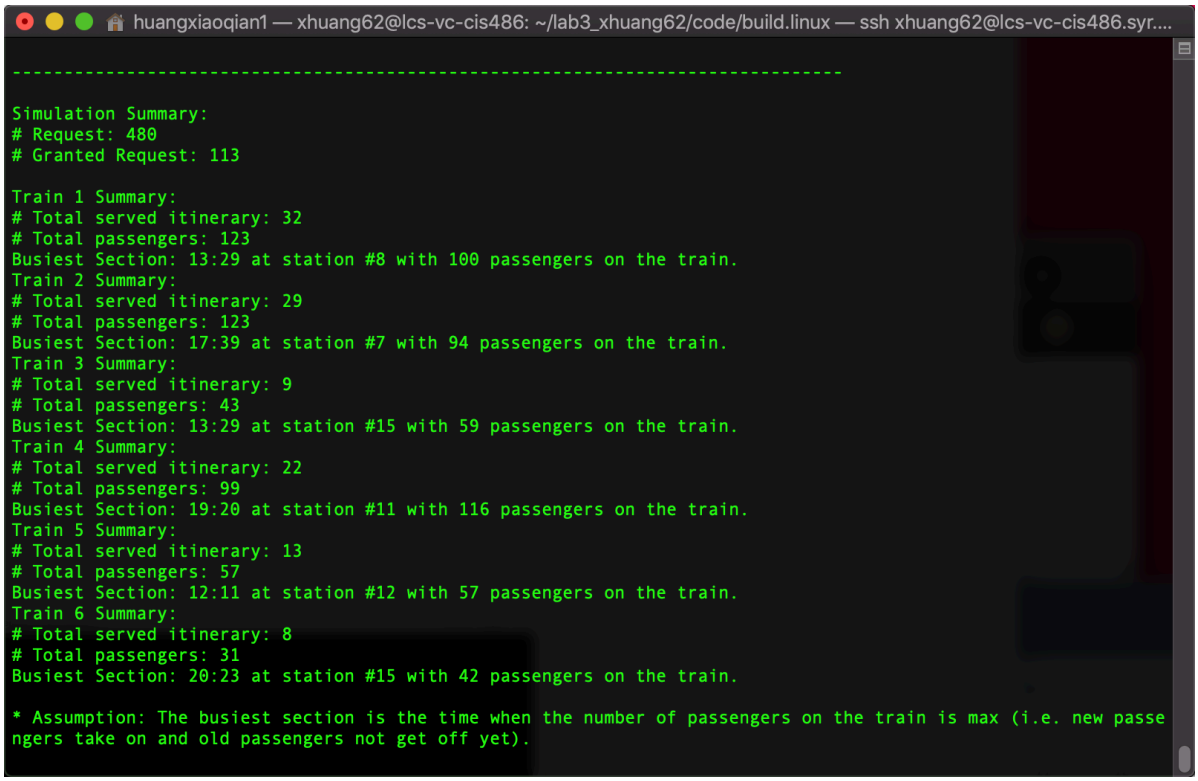
2. When verifying the simulation process, I printed the following information to prove the correctness of my simulation:
   1) The information of new generated request
      New Request: from 13 to 10 pNum: 2 class: 1 departureTime: 19:58
   2) Whether the request can be granted, which train(s) is available for this request and which train is finally assigned to it.
      [4] 1 trains can be granted. - Request 346 is granted to train[4] with 2 passengers. Class: 1, from 13 to 10
      * [4] means the train with name "4" is available for this request
   3) Granted request number and refuse request number (required output)
      # Granted Request: 2
      # Refused Request: 3
   4) If the simulating time for train schedule (arrive and depart) is achieved, the information of train will be printed out.
      Train[1] - destination station: 13 at time 17:34
      # Passenger getting off: 12
      Bitmap set:
      5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 35, 36, 37, 38, 42, 51, 52, 53, 54, 57,
      * Specially, the bitmap seat is also print out, meaning the seat number that is occupied. (0-19 is business, 20-59 is coach).
   5) When the simulation finished, all the required output is printed.

   According to the result, it's easy to prove that my simulation is run as expected.

5. Output Snapshots






huangxiaoqian1 — xhuang62@lcs-vc-cis486: ~/lab3_xhuang62/code/build.linux — ssh xhuang62@lcs-vc-cis486.syr....

Simulation Summary:
Request: 480
Granted Request: 113
Train 1 Summary:
Total served itinerary: 32
Total passengers: 123
Busiest Section: 13:29 at station #8 with 100 passengers on the train.
Train 2 Summary:
Total served itinerary: 29
Total passengers: 123
Busiest Section: 17:39 at station #7 with 94 passengers on the train.
Train 3 Summary:
Total served itinerary: 9
Total passengers: 43
Busiest Section: 13:29 at station #15 with 59 passengers on the train.
Train 4 Summary:
Total served itinerary: 22
Total passengers: 99
Busiest Section: 19:20 at station #11 with 116 passengers on the train.
Train 5 Summary:
Total served itinerary: 13
Total passengers: 57
Busiest Section: 12:11 at station #12 with 57 passengers on the train.
Train 6 Summary:
Total served itinerary: 8
Total passengers: 31
Busiest Section: 20:23 at station #15 with 42 passengers on the train.
* Assumption: The busiest section is the time when the number of passengers on the train is max (i.e. new passe
ngers take on and old passengers not get off yet).