CIS 657 LAB3 report

A Train Simulation system

Name: Jie Ren
SUID: 21037-2590
netID:jren21

CIS657 Fall 2019
Assignment Disclosure Form

Assignment #:

Name: Jie Ren

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:___Jie Ren_________________________ Date : 09/28/2019

PART1 : Time I spent on this lab

- How much time did you spend to do:
    - o Analyze the problem, determine specifications, and create your design

About 1 day of design. Including thinking about the details and discuss with others.

- o Implement the design
    - ▪ write the program

About 1.5 days of implementation

- o Test/debug the program
    - ▪ Find/fix errors
    - ▪ Create test cases, and try them out

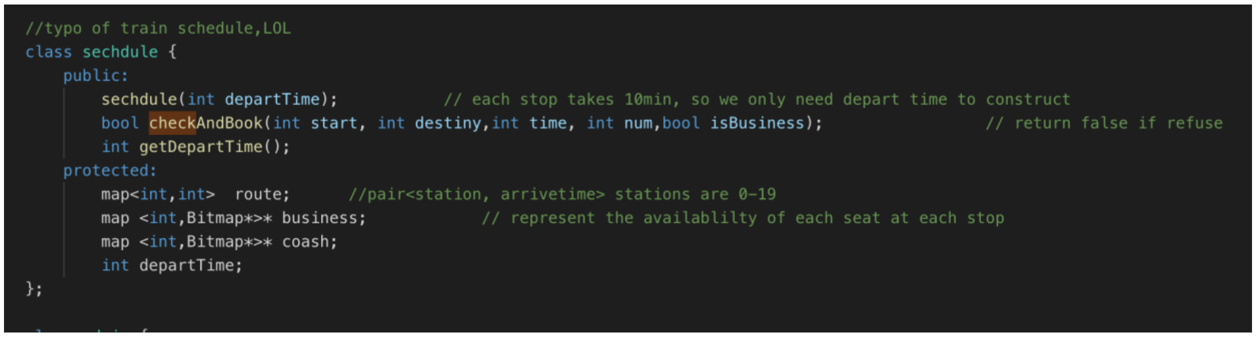
About another 1.5 days of debugging.

(all the days mentioned above is about 8-10hours full day, so the workload of this lab is still a little bit too heavy).

PART2 : brief introduction of my design

I create 2 classes for the entire simulation.

1. class sechdule{}
   This holds all the information of the train sechdule. Including the departing time and arriving time of the train as well as the seats available on that train.
   (Yes, in my design. I do not have a train class. Instead, I stroe the information of each train as sechdules in the admin center. The admin center class has a list of sechdules)

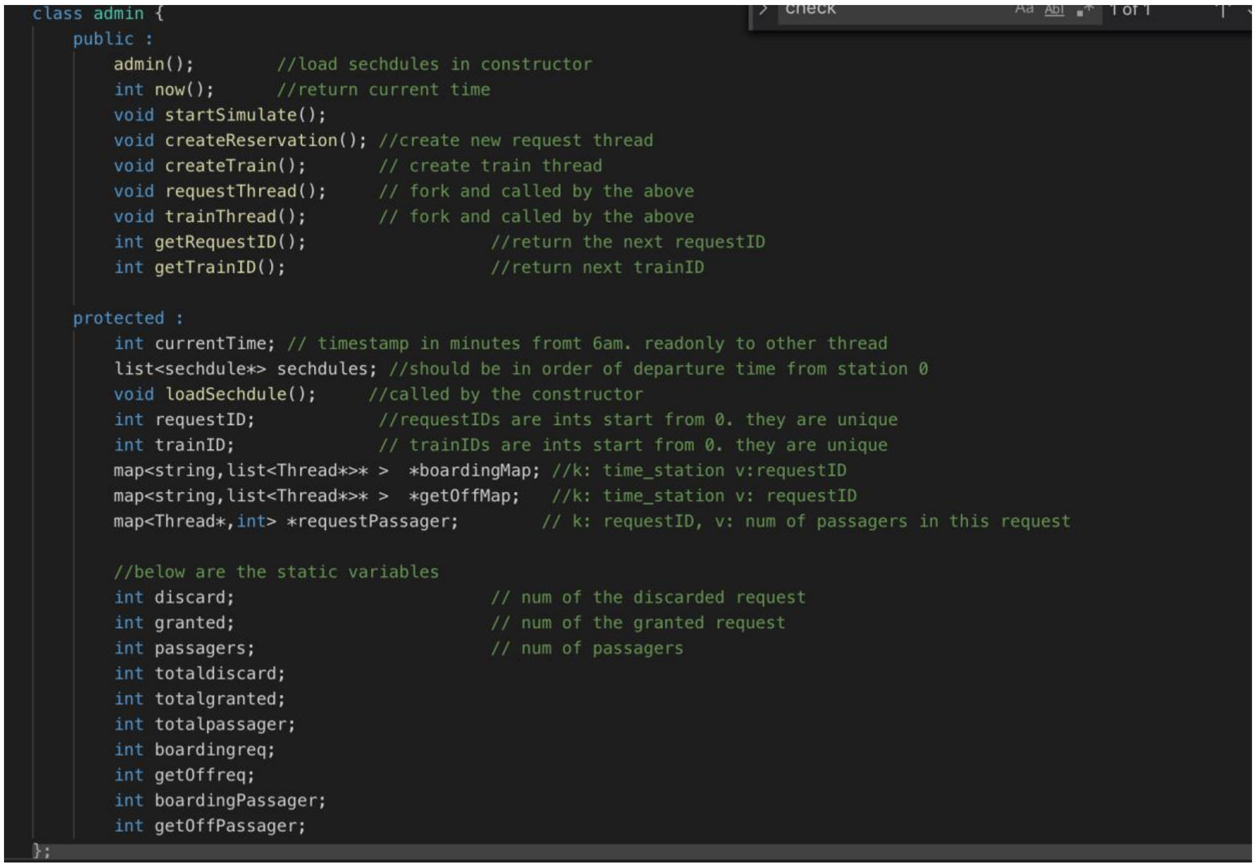


```
//typo of train schedule,LOL
class sechdule {
    public:
        sechdule(int departTime);           // each stop takes 10min, so we only need depart time to construct
        bool checkAndBook(int start, int destiny,int time, int num,bool isBusiness);                // return false if refuse
        int getDepartTime();
    protected:
        map<int,int>  route;      //pair<station, arrivetime> stations are 0-19
        map <int,Bitmap*>* business;           // represent the availablilty of each seat at each stop
        map <int,Bitmap*>* coash;
        int departTime;
};
```

2. the admin center class{}
   This is the class that serves as the admin center of a train system. It holds all the sechdules of trains, including the seats reservation information of each train. It also has member

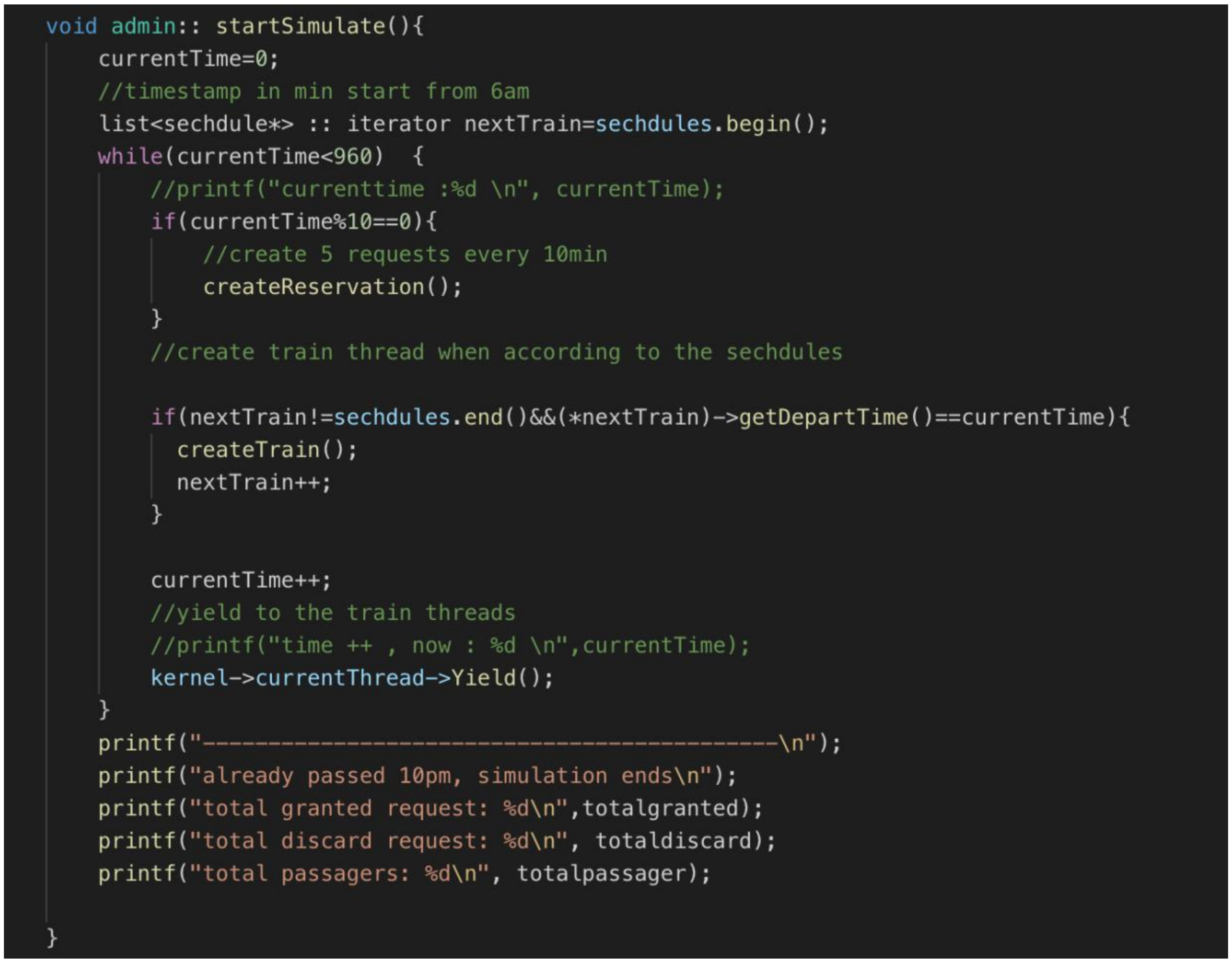
functions that simulate the action of create reservation request and running a train(get passagers on and off).

What 's more , on top of that. It has a startSimulation() function which simulate the time flow and schedule the threads in the entire process.



```
class admin {
    public :
        admin();        //load sechdules in constructor
        int now();      //return current time
        void startSimulate();
        void createReservation(); //create new request thread
        void createTrain();       // create train thread
        void requestThread();     // fork and called by the above
        void trainThread();       // fork and called by the above
        int getRequestID();                 //return the next requestID
        int getTrainID();                   //return next trainID

    protected :
        int currentTime; // timestamp in minutes fromt 6am. readonly to other thread
        list<sechdule*> sechdules; //should be in order of departure time from station 0
        void loadSechdule();     //called by the constructor
        int requestID;            //requestIDs are ints start from 0. they are unique
        int trainID;              // trainIDs are ints start from 0. they are unique
        map<string,list<Thread*>* >  *boardingMap; //k: time_station v:requestID
        map<string,list<Thread*>* >  *getOffMap;   //k: time_station v: requestID
        map<Thread*,int> *requestPassager;        // k: requestID, v: num of passagers in this request

        //below are the static variables
        int discard;                        // num of the discarded request
        int granted;                        // num of the granted request
        int passagers;                      // num of passagers
        int totaldiscard;
        int totalgranted;
        int totalpassager;
        int boardingreq;
        int getOffreq;
        int boardingPassager;
        int getOffPassager;
};
```

The figure above shows the structure of class admin();



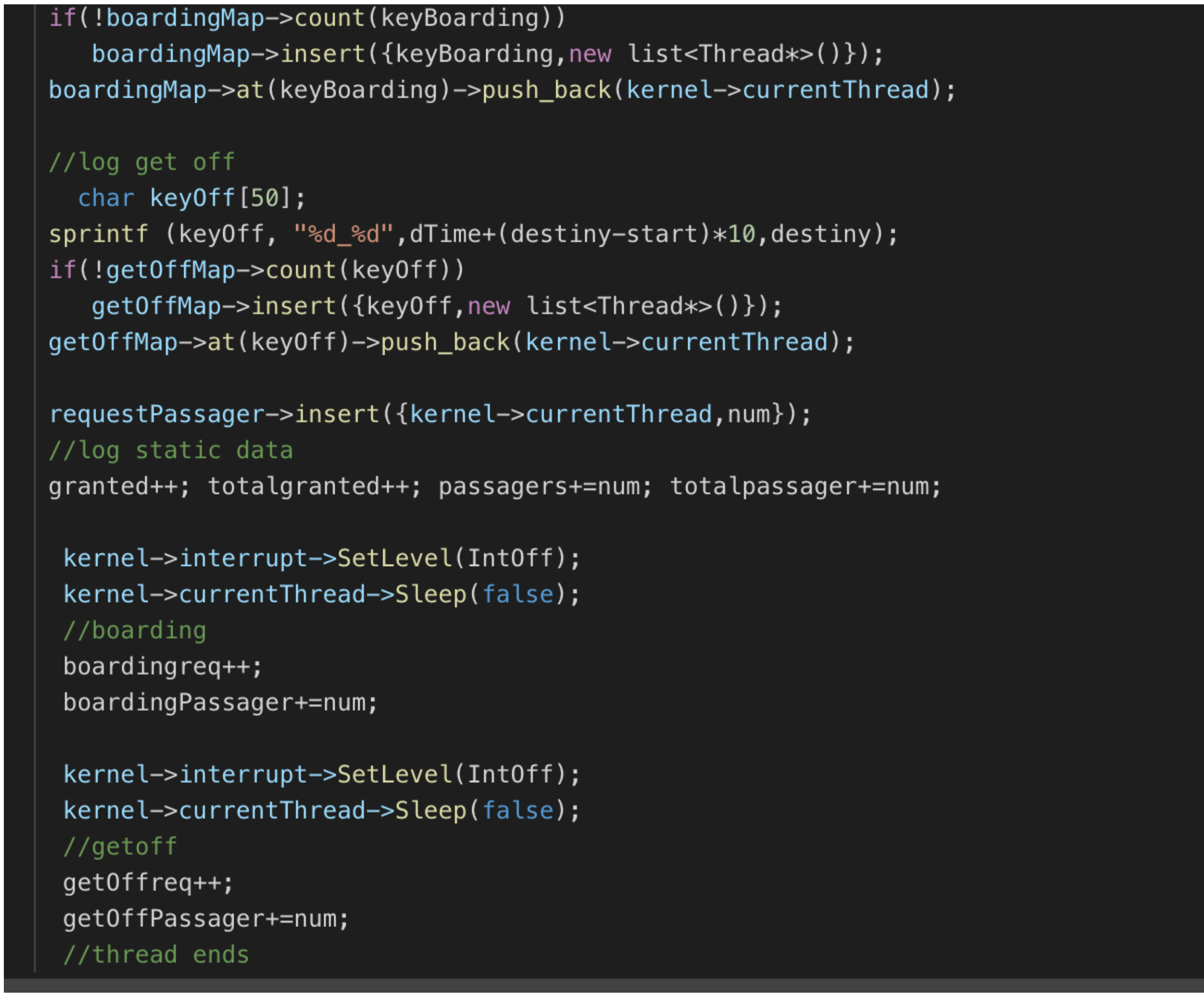
```
void admin:: startSimulate(){
    currentTime=0;
    //timestamp in min start from 6am
    list<sechdule*> :: iterator nextTrain=sechdules.begin();
    while(currentTime<960)  {
        //printf("currenttime :%d \n", currentTime);
        if(currentTime%10==0){
            //create 5 requests every 10min
            createReservation();
        }
        //create train thread when according to the sechdules

        if(nextTrain!=sechdules.end()&&(*nextTrain)->getDepartTime()==currentTime){
          createTrain();
          nextTrain++;
        }

        currentTime++;
        //yield to the train threads
        //printf("time ++ , now : %d \n",currentTime);
        kernel->currentThread->Yield();
    }
    printf("-------------------------------------------\n");
    printf("already passed 10pm, simulation ends\n");
    printf("total granted request: %d\n",totalgranted);
    printf("total discard request: %d\n", totaldiscard);
    printf("total passagers: %d\n", totalpassager);

}
```

This figure shows the main entrance of the simulation.
In the createReservation() function, it generate 5 request(with 5 forked threads) every 10 min.

If the request is denied, the thread dies at instantly.

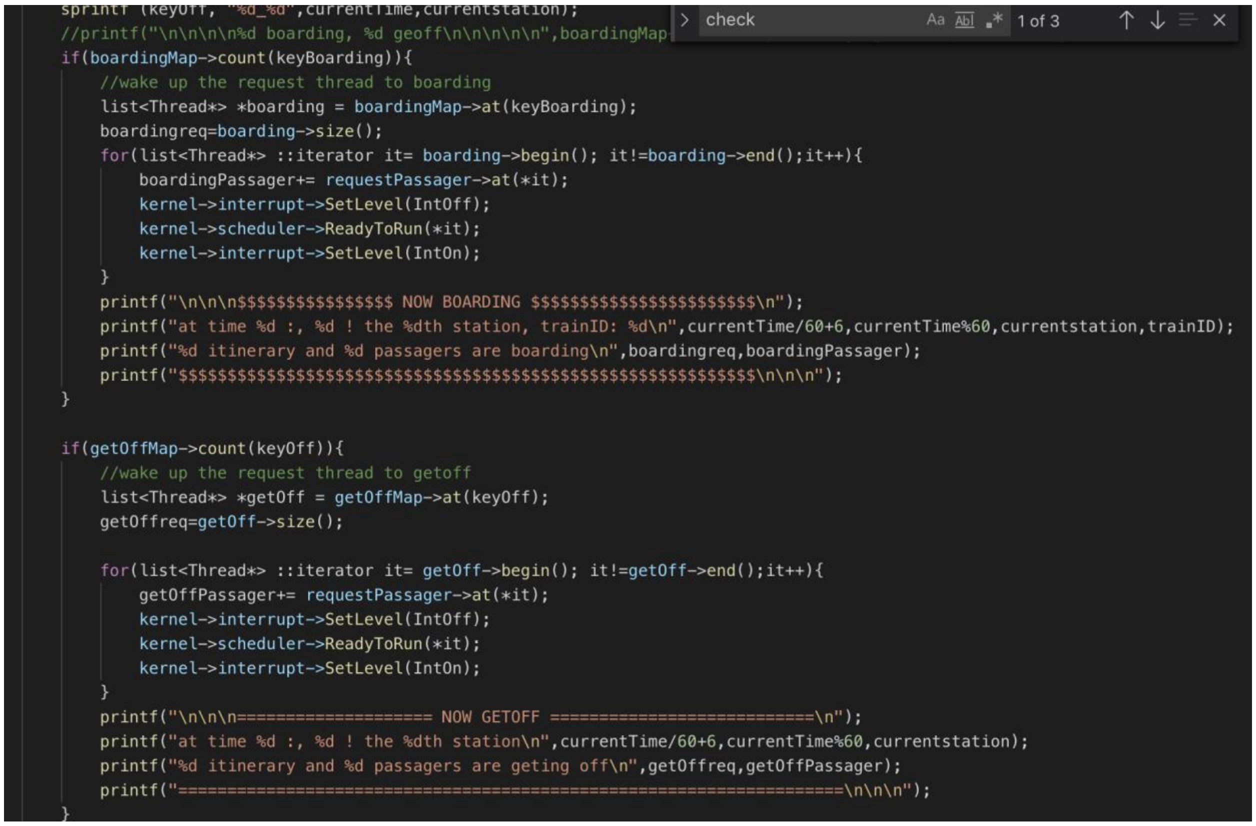
If the request is granted, it will sleep until the train thread(created by the createTrain(),according to the schedules loaded from the files) to wake it up when the passagers shold boarding or get of the train. After the passengers in the request get off the train and finish their trip. The request thread is distoried.

The logic that print out the static data is also embeded in the admin center.



```
if(!boardingMap->count(keyBoarding))
   boardingMap->insert({keyBoarding,new list<Thread*>()});
boardingMap->at(keyBoarding)->push_back(kernel->currentThread);

//log get off
  char keyOff[50];
sprintf (keyOff, "%d_%d",dTime+(destiny-start)*10,destiny);
if(!getOffMap->count(keyOff))
   getOffMap->insert({keyOff,new list<Thread*>()});
getOffMap->at(keyOff)->push_back(kernel->currentThread);

requestPassager->insert({kernel->currentThread,num});
//log static data
granted++; totalgranted++; passagers+=num; totalpassager+=num;

 kernel->interrupt->SetLevel(IntOff);
 kernel->currentThread->Sleep(false);
 //boarding
 boardingreq++;
 boardingPassager+=num;

 kernel->interrupt->SetLevel(IntOff);
 kernel->currentThread->Sleep(false);
 //getoff
 getOffreq++;
 getOffPassager+=num;
 //thread ends
```

In the request thread. It store the information of when and where to aboard and getoff to a map. And then sleep until the train thread wake it up.



```
    sprintf (keyOff, "%d_%d",currentTime,currentstation);
    //printf("\n\n\n\n%d boarding, %d geoff\n\n\n\n\n",boardingMap
    if(boardingMap->count(keyBoarding)){
        //wake up the request thread to boarding
        list<Thread*> *boarding = boardingMap->at(keyBoarding);
        boardingreq=boarding->size();
        for(list<Thread*> ::iterator it= boarding->begin(); it!=boarding->end();it++){
            boardingPassager+= requestPassager->at(*it);
            kernel->interrupt->SetLevel(IntOff);
            kernel->scheduler->ReadyToRun(*it);
            kernel->interrupt->SetLevel(IntOn);
        }
        printf("\n\n\n$$$$$$$$$$$$$$$$ NOW BOARDING $$$$$$$$$$$$$$$$$$$$$$$\n");
        printf("at time %d :, %d ! the %dth station, trainID: %d\n",currentTime/60+6,currentTime%60,currentstation,trainID);
        printf("%d itinerary and %d passagers are boarding\n",boardingreq,boardingPassager);
        printf("$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$\n\n\n");
    }

    if(getOffMap->count(keyOff)){
        //wake up the request thread to getoff
        list<Thread*> *getOff = getOffMap->at(keyOff);
        getOffreq=getOff->size();

        for(list<Thread*> ::iterator it= getOff->begin(); it!=getOff->end();it++){
            getOffPassager+= requestPassager->at(*it);
            kernel->interrupt->SetLevel(IntOff);
            kernel->scheduler->ReadyToRun(*it);
            kernel->interrupt->SetLevel(IntOn);
        }
        printf("\n\n\n==================== NOW GETOFF ==========================\n");
        printf("at time %d :, %d ! the %dth station\n",currentTime/60+6,currentTime%60,currentstation);
        printf("%d itinerary and %d passagers are geting off\n",getOffreq,getOffPassager);
        printf("===================================================================\n\n\n");
    }
```

Wake up the request thread in train threads.

PART3 : build instruction

Cd to the build.linux folder.

```
$ Cd code/build.linux
$ make depend
$ make
$ nachos -K
```