Train Unloading Homework Write Up

Name: Chenhao Qian
Student ID: 010035275

The Problem

The problem is to write a simulation of a train unloading dock. Trains arrive at the station as a Poisson process on average once every 10 hours. Each train takes between 3.5 and 4.5 hours, uniformly at random, to unload. If the loading dock is busy, then trains wait in a first-come, first-served queue outside the loading dock for the currently unloading train to finish. Negligible time passes between the departure of one train; and the entry of the next train (if any) into the loading dock---unless the entering train has no crew.

When a train arrives, the remaining time of the crew is uniformly between 6 and 11 hours. Once the crew of a train has hogged out, the next crew will be called with a travel time uniformly between 2.5 and 3.5 hours.

Arrival is accepted before 72000 simulation hours.

The questions of concern are as follows:

1. Total number of trains served.
2. Average and maximum of the time-in-system over trains.
3. The percentage of time dock being busy, idle and hogged-out.
4. Time average and maximum number of trains in the queue.
5. Histogram of the number of trains hogged out various times.

Simulation Logic

I followed the textbook example to set up my logic of the simulation.

There are a total of five different events, namely arrival, exitQ, hogout, hogin and depart. The event of unload is subsumed into either arrival when a train arrives with empty queue and idle loading dock or exitQ when a train moves from the first in queue into the loading dock.

An array of time for the next occurrence of each event time is kept and a timing function that finds the event type with earliest scheduled time is used to determine the next event.

A class is defined for trains with parameters to store information and statistics.

For the termination of the simulation, an end time is set for the boundary for arrival. When the next arrival time is greater than the end time, the arrival time for the event schedule array is set to infinity. Thus all other remaining events will be completed before the actual termination of simulation. When all events are scheduled at infinity, report with statistics will be generated and the simulation if formally terminated.

The detailed logic of each event will be discussed below with code. The whole program is implemented with one .cpp file.

Main Function and Command Line Input

As required in the assignment, there should be two ways of input from the user: 2 arguments or 3 arguments provided as command line input. To achieve this, I counted the number of arguments in the command line. Given 3, counting also the program name, a Boolean variable “readfile” is set to 0 to indicate we are not reading data from files; given 4, the variable is set to 1. All other number of arguments will pop an error and terminate the program.

The remaining part of the main function is quite straight forward. An output path is created for file output. An initialization function is called to configure the starting condition of the whole simulation. The main body of the simulation is a while loop with a timing function to select the earliest event and move clock forward to the next event. It is also the timing function that recognizes the termination condition (all events scheduled at infinity) and redirects the next event to reporting. The loop ends when an report event is called.

The code for the main function is given here.

```c
int main(int argc, char *argv[])
{
    srand((unsigned int)time(NULL));
    /*check number of arguements input*/
    if (argc == 4) { //2 arguments,
        infile1 = fopen(argv[2], "r");
        infile2 = fopen(argv[3], "r");
        readfile = 1;
    }
    else if (argc == 3) {
        time_end = atof(argv[1]);
        mean_interarrival = atof(argv[2]);
        readfile = 0;
    }
    else {
        printf("\nInput not valid\n");
        exit(1);
    }

    /*Write report heading and input parameter*/
    outfile = fopen("train_output.txt", "w");
    fprintf(outfile, "Train unloading system simulation\n\n");

    /*Initialize simulation*/
    initialize();

    do {
        timing();
        update_stats();
        switch (next_event_type)
        {
        case 1: {
            arrival();
            break;
        }
        case HOGOUT:
            hogout(hogouttrain);
            break;
        case HOGIN:
            hogin(hogintrain);
            break;
        case DEPART:
            depart();
            break;
        case EXITQ:
            exitQ();
            break;
        case 6:
            report();
            break;
        }
    } while (next_event_type != 6);

    //close all files
    if (readfile) {
        fclose(infile1);
        fclose(infile2);
    }
    fclose(outfile);
    return 0;
}
```

Initialization

In the initialization stage, I first set the clock to zero. Next, I set the all state variables to proper state, such as whether a train is on dock to false. Then, I initialized all the statistics variables to zero. In addition to that, time indicators are set to infinity (practically 1.0e+30). Most importantly, the first arrival event is scheduled. This is done conditioning on the Boolean variable “readfile”. If it is true, read from the first file specified in command; otherwise, generate the time from an exponential distribution (derived from a uniform distribution).

To minimize the length of this text, only the code for scheduling first arrival will be showed here.

```c
    //Schedule first arrival
    if (readfile) { //if reading from schedule
        float firstarrival;
        fscanf(infile1, "%f", &firstarrival);
        time_next_event[1] = firstarrival;
    }
    else //if generating random value
        time_next_event[1] = currenttime + interarrival(mean_interarrival);
```

Arrival

In the arrival event, a new train (defined as a custom class) is generated. The unloading time and the remaining time of the crew on the train are again created conditioning on “readfile”. All other parameters of the train class are set to the proper initial stage. Next, the next arrival event is scheduled also conditioning on “readfile”. In addition, termination rule is being examined at this step. If we are reading from a file, the next arrival is set to infinity if it is already the end of the file. If we are generating random inter-arrival time, the next arrival is set to infinity if the current time plus the generated time is greater than the given simulation limit.

After that, I check if the dock is empty and the queue is empty. If so, the train is moved to the dock (position 0 in the queue array) and starts unloading. Otherwise, the train is moved to the end of the queue (position indicator equals the current length of the queue counting this new train). In either case, the next hog-out event for this particular train is scheduled at current time plus the remaining time of the crew. The value is recorded in the hog-out timetable at the same position as the train in the train queue. After schedule a hog-out event, a function “update_next_hog” is called to locate the earliest hog-out event in the timetable and update the event schedule.

Notice here the unload event is subsumed since it is always happening at the same time of an arrival or when a train exits queue. Thus the unloading is created as just a function that schedules the departure event of the train on dock.

The code for the arrival function is given below.

```
void arrival(void)
{
    total_arrival++; //count arrival
    crew_num++; //update crew number code
        /*Generate unloading time and remaining crew time for the arrived trai
    trainparameter train;

    if (readfile) {
        fscanf(infile1, "%f %f", &train.unloadtime, &train.remaintime);
        float nextarrival;
        if (!feof(infile1)) {
            fscanf(infile1, "%f", &nextarrival); //schedule next arrival
            time_next_event[1] = nextarrival;
        }
        else time_next_event[1] = (float) 1.0e+30; //terminate at end of file.
    }
    else {
        train.unloadtime = unloadingtime(3.5, 4.5);
        train.remaintime = crewremainingtime(6, 11);
        time_next_event[1] = currenttime + interarrival(mean_interarrival);
        if (time_next_event[1] > time_end) {
            time_next_event[1] = (float) 1.0e+30; //arrivals after end time wi
        }
    }

    /*Initialize variables for the arrived train*/
    train.trainname = total_arrival;
    train.crewcode = crew_num;
    train.arrival = currenttime;
    train.hogoutcount = 0;
    train.starthogout = (float) 1.0e+30;
    train.startwait = 0.0;
    train.timeinQ = 0.0;
    train.timeinsystem = 0.0;
    train.timeout = 0.0;
    train.trainhoggedout = 0;

    //report arrival event
    fprintf(outfile, "\nTime %.2f, train %i arrived with crew

    if (!trainondock && currentq == 0) { //if no train on doc
        //change dock status
        trainondock = 1;
        dockstartbusy = currenttime;
        dockidletime += currenttime - dockstartidle;

        train.timeinQ = 0.0; //record 0 waiting time

        trainQ[0] = train; //locate the train on dock
        hogtime[0] = currenttime + train.remaintime; //schedu
        unload();
    }
    else { //otherwise
        train.startwait = currenttime;
        currentq++;
        trainQ[currentq] = train;
        hogtime[currentq] = currenttime + train.remaintime;
        maxq = (int)max((float)currentq, (float)maxq);
    }
    update_next_hog();
}
```

Hog-out

The hog-out function has an input variable “k”, which is an indicator of the train relevant to the event that shows the position of the train in queue. In the event of hog-out, the hog-out count gets incremented while the start hog-out time is recorded and the train is set to hog-out status. Also, a new crew is called with incremented crew ID. The duration of the hog-out is also conditioned on “readfile”. To prevent the case that not enough crew is provided in the crew travel time file, the whole simulation will be terminated immediately with an error message. Such condition is not needed when generating time from a random distribution. Once the time is determined, the hog-in event of the given train is scheduled and the next hog-out of the train will be 12 hours later. Again the global hog-out and hog-in schedule will be filtered and the event schedule updated.

In addition, I checked if the train hogging out is currently on dock (unloading) by checking whether “k” is 0. If so, the departure time of the train is also pushed back by the hog-out time.

The complete code is given below.

```
void hogout(int k) {
    float hogouttime;
    trainparameter train;
    train = trainQ[k];
    train.trainhoggedout = 1;
    train.starthogout = currenttime; // record time train hogged out
    train.hogoutcount++; //count hog times
    crew_num++;
    train.crewcode = crew_num;

    //generate crew return time
    if (readfile) {
        if (!feof(infile2)) fscanf(infile2, "%f", &hogouttime);
        else {
            fprintf(outfile, "\n\nNot enough crew to come in. Simulation terminated.\n");
            report();
            fprintf(outfile, "!ERROR!\nSimulation terminated with a problem. Check input data file.\n");
            exit(0);
        }
    }
    else hogouttime = crewreturntime(2.5, 3.5);

    train.remaintime = 12 - hogouttime;
    fprintf(outfile, "\nTime %.2f, train %i hogged out, crew %i has been called and will take %.2f hours to arrive.", currenttime, train.trainname, crew_num, hogouttime);
    if (k == 0) { //in the case of server hog out
        fprintf(outfile, "(Server hogged out)");
        time_next_event[DEPART] += hogouttime; //delay depart time
        dockbusytime += currenttime - dockstartbusy; //record dock busy time
        dockstartidle = currenttime; //start recording dock idle time
    }
    hogtime[k] = currenttime + 12;
    backtime[k] = currenttime + hogouttime;
    update_next_hog();
    update_next_return();
    trainQ[k] = train;
}
```

Hog-in

The hog-in event is relatively simple. All hog-out related statistics variables are updated at this step and the next return time of the train is set back to infinity.

```
void hogin(int k) {
    fprintf(outfile, "\nTime %.2f, crew %i returned to train %i.", currenttime, trainQ[k].crewcode, trainQ[k].trainname);
    trainQ[k].timeout += currenttime - trainQ[k].starthogout; //record total hogout time for the train
    trainQ[k].trainhoggedout = 0.0;
    backtime[k] = (float) 1.0e+30;
    update_next_return();
    if (k == 0) { //in the case of server hog out
        fprintf(outfile, "(Resume unloading)");
        dockstartbusy = currenttime; //start recording dock busy time
        dockidletime += currenttime - dockstartidle; //record dock idle time
        dockhogouttime += currenttime - dockstartidle; //record dock hogout time
    }
}
```

Departure

The departure event depends on whether the queue is empty. If so, set the dock to idle.

If not, then I checked if the first train in queue is hogged out. If the train is hogged out, the exit queue event of that train is set to its hog-in time. Otherwise, exit queue is scheduled now.

Then all the statistics of the train is calculated and finalized. In the end, the departure time and hog-out and hog-in time of this train are all set to infinity.

```
void depart(void) {
    trainparameter train;
    train = trainQ[0];
    fprintf(outfile, "\nTime %f, train %d finish unload and leave dock", currenttime, train.trainname);
    /*schedule the first train in Q to exit, otherwise set dock to idle*/
    if (currentq != 0) {
        if (trainQ[1].trainhoggedout) {
            time_next_event[EXITQ] = backtime[1];
            dockbusytime += currenttime - dockstartbusy; //record dock busy time
            trainondock = 0; //change dock status to idle
            dockstartidle = currenttime; //start recording dock idle time
            fprintf(outfile, "\nTime %.2f, train %d is called to exit queue while hogged out, dock is idle", cur
        }
        else {
            time_next_event[EXITQ] = currenttime;
        }
    }
    else {
        dockbusytime += currenttime - dockstartbusy; //record dock busy time
        trainondock = 0; //no train on dock
        dockstartidle = currenttime; //start recording dock idle time
        fprintf(outfile, "; no train in queue, dock is idle.");
    }
    train.timeinsystem = currenttime - train.arrival; //record train time in system;
    total_timeinsystem += train.timeinsystem;
    max_timeinsystem = max_timeinsystem > train.timeinsystem ? max_timeinsystem : train.timeinsystem;
    total_timeinQ += train.timeinQ;
    max_timeinQ = max_timeinQ > train.timeinQ ? max_timeinQ : train.timeinQ;
    total_timeout += train.timeout;
    max_timeout = max_timeout > train.timeout ? max_timeout : train.timeout;
    addtohist(train.hogoutcount);
    hogtime[0] = (float) 1.0e+30;
    backtime[0] = (float) 1.0e+30;
    time_next_event[DEPART] = (float) 1.0e+30;
    update_next_hog();
    update_next_return();
}
```

Exit Queue

The exit queue event calculates the time the train spent in queue. Then, the train is moved to dock (position 0 of the queue) and all following trains moves one step ahead. One precaution to take is that if the train was hogged out when the previous train depart, the dock would have been set to idle. We would need to set it back to busy in such case. The code is given below.

```
void exitQ(void) {
    trainparameter train;
    train = trainQ[1];
    trainQ[1].timeinQ = currenttime - train.startwait;//calculate time in Q
    fprintf(outfile, "\nTime %.2f, train %d enter dock for unloading after waiting in Q for %.2f hc
    if (!trainondock) { //if dock is idle, set it to busy
        trainondock = 1;
        dockstartbusy = currenttime;
        dockidletime += currenttime - dockstartidle;
    }
    //move train to dock and move Q one position forward
    for (int i = 0; i <= currentq; i++) {
        trainQ[i] = trainQ[i + 1];
        hogtime[i] = hogtime[i + 1];
        backtime[i] = backtime[i + 1];
    }
    update_next_hog();
    unload();
    currentq--;
    time_next_event[EXITQ] = (float) 1.0e+30;
}
```

Timing

The timing function is simply a loop screening through the scheduled time for each event time and take the one that comes first. After the loop, if no event is found since all of them are at infinity, the end of simulation will be noticed and the report function is called to give the final statistics.

```
void timing(void) {
    int i;
    float min_next_event = (float) 1.0e+10;
    next_event_type = 0;
    for (i = 1; i <= 5; i++) {
        if (time_next_event[i] < min_next_event) {
            min_next_event = time_next_event[i];
            next_event_type = i;
        }
    }
    if (next_event_type == 0) {
        fprintf(outfile, "\nTime %.2f, event list empty. Proceed to generate report.\n\nEND OF SIMULATION\n", currenttime);
        next_event_type = 6;
        totalruntime = currenttime;
    }
    time_last_event = currenttime;
    currenttime = min_next_event;
}
```

Calculation Methods for Statistics of Interest

1. Total number of trains served

   This is simply equal to total arrival which is incremented at each arrival event.

2. Average and maximum of the time-in-system over trains

This calculated by the departure time of each train subtract its arrival time. Then at the event of departure, the train's time in system is compared to the current maximum to keep the larger. It is also added to the total time in system for all trains. This value is divided by total arrival at the end of simulation.

3. The percentage of time dock being busy, idle and hogged-out

   The dock's busy and idle time is calculated every time the status change. For example, when the dock is changed from idle to busy, the idle time for the past period is current time subtract the time when dock is switched to idle. This value is then incremented to the total idle time up to now. Meanwhile the time dock switched to busy is recorded as current time for the next calculation of busy time. At the end of simulation, the busy time and idle time are divided by the total time of the run to obtain the percentage.

   For dock hog-out time, it is only calculated when a train hogs out during service. In such case, it is the same as the duration of the hog-out of the train.

4. Time average and maximum number of trains in the queue

   The time total of trains in queue is incremented before each event after the initialization. The amount of increment equals the time interval between the coming event and the last event times the current queue size before the coming event occur. At the end of simulation, the total is divided by overall run time to obtain the time average.

   The maximum number is compared to the current queue size whenever the queue increases and keeps the larger value.

5. Histogram of the number of trains hogged out various time

   The hog-out count of each train is passed to the corresponding bin at departure.

Results

1. The total arrival / service count is roughly end time divided by the given mean arrival rate when the simulation runtime is long enough. This is logical as it follows the definition of mean arrival rate.

2. The maximum time in system for a train is about 30 hours while average is about 6 hours. For potential interest, I also calculated the maximum and average time in queue for a train, which are about 26 hours and 2 hours respectively. Thus we can see that the time in system is roughly the time in queue plus the mean unloading time (of the uniform distribution).
3. The percentage dock busy, idle and hog-out times are roughly 40%, 60% and 3%. The busy time and idle time always add up to 100% since the dock can only switch between the two states. Hog-out times of a dock are also counted as idle, so there is duplicated counting here.
4. The maximum number of trains in queue is around 6 while the time average is about 0.2. This implies that the dock is quite sufficient for the task of the setting and the queue won't get too big to cause a problem.
5. The histogram shows that the majority of trains didn't hog out and most of the remaining ones hogged out only once. The number of trains that hogged out twice or more is significantly fewer. This matches our observation in 2) and 4) as the queue size didn't get very big and trains don't typically spend much time waiting in queue.