Requirement:
This assignment required to write the bus simulation.
Write the bus simulation, as explained in the class and described in the notes. Feel free to look on the web for and then re-use any suitable code for the random number generator or for linked list algorithms. With that, 1) don't **forget** to include a reference to the source of any code you re-use and 3) remember to test all re-used code as you are the only one responsible for its performance.
The purpose of the simulation is to observe the behavior of the system, and answer the following questions:
1. Does the distance between the buses keep uniform? If not, what should be done to ensure it is uniform?
2. What is the average size of a waiting queue at each stop (and what are its maximum and minimum)? (You may provide this information on an hourly [simulation time] base.)

Plot the positions of buses as a function of time (you will need to generate periodic snapshots of the system for that). Feel free to change parameters; then observe and document the results.

Abstract:

The objective of this programming assignment is to familiarize the discreet simulation and event processing. Constructing the event structure, running mode. Using priority queue and mutual exclusion to make sure the system work well. Finally, based on the output data, analyzing the buses' distance and give some assumptions about the question. The entire system use C++ language to simulation. Using Visual Studio 2015. The figures are used matlab to generate.

1.1 System Analysis

The system has three event: person, arrival and boarding.
The person event is that generate a person comes to stop at random time. This has a requirement that the average number of coming person is 5/min. The random event should Obedience the Exponential Distribution.

The arrival event is that the bus process arrive at a stop, which has two case:
Case 1: If there is no people waiting at the current bus stop, the bus will go to the next stop.

Case 2: If there are some people waiting at the current stop, the bus event will be locked. If there is no other buses at the stop, call the boarding event. Or waiting for the bus which at the current stop.

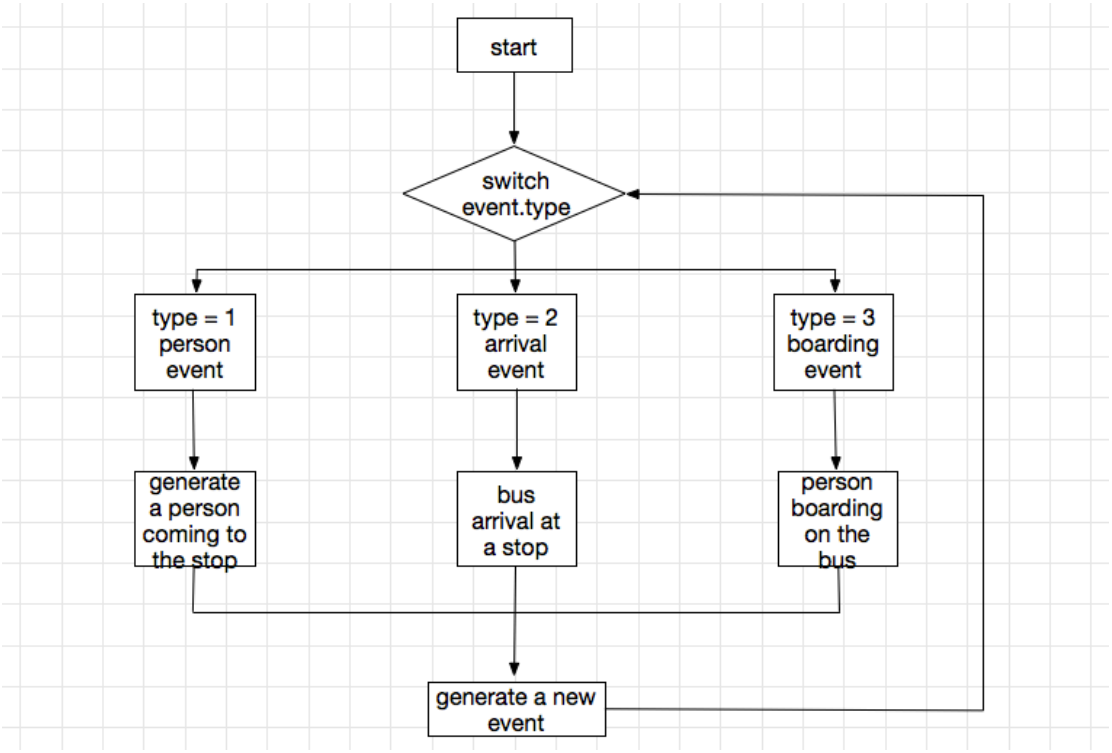
The boarding event is that people boarding on the bus. Each person boarding on the bus cost 2 seconds. The boarding event has two cases:

Case 1: If the boarding person is the last person at the current stop, the bus will go to the next stop and release the lock.
Case 2: If the boarding person is not the last person, the bus event will call boarding event again.

1.2 Data Structure

The system define a class Event, using private member variable: “time”, “stop_number”, “bus_number”, “event_count” and “event_type” to store the event current stop number, current bus number, the number of the event and event type respectively. The system uses double weight priority queue to store the events. Their first and main priority weight is time. Their second priority weight is events happen order. Make them into time ascending order. The system initializes with reading a text file that record the original parameters and will output the system log, bus information and stop information to the txt file.

1.3 Program Flow Chart



2.1 System Design

The system contains 2 header file and 3 cpp file. See table 2.1 below.

Table 2.1 System File Display

Filename	Description
Event.h	Define event variables and functions that maybe used
Event.cpp	Implement member functions of Event.h
Initial.h	Define initial functions and other functions
Initial.cpp	Implement functions that define in initial.h
Simulation.cpp	The main part of bus simulation. Implement that how the event execute.
Stop_waiting_length_hourly.txt	Record the average, max average and min average waiting people number for each stop
Stop_total_average_waiting_info.txt	Record the whole 8 hours average, max average and min average waiting people of each stop
Initial.txt	Initial data for the system
Bus_Stop_waiting_Observe.txt	Record the waiting people number of each bus stop
Bus_number_one.txt	Record bus 1 position at each event time
Bus_number_two.txt	Record bus 2 position at each event time
Bus_number_three.txt	Record bus 3 position at each event time
Bus_number_four.txt	Record bus 4 position at each event time
Bus_number_five.txt	Record bus 5 position at each event time
log.txt	Record the system log (person, arrival, boarding event)

2.2 Design Class Event

This class should contain all event attributes. According to the homework requirement, the class was designed as figure 2.1

```
class Event {  //  define the event structure. Main simulation object
private:
    int event_type; // define event type
    double time;    // define the event time
    int stop_number;    // record the current event stop number
    int bus_number;     // record the current bus number
    int event_count;    // record the current event number
public:
    Event(int type, double system_time, int stop_number, int bus_number, int event_counter);
    Event(int type, double system_time, int stop_number,int event_counter);
    int get_Type();                                 // get the event type
    double get_time();                              // get the event time
    int get_Stop_Number();                          // get the event stop number
    int get_Bus_Number();                           // get the event bus number
    double update(double increase_time);            // update the event time
    int get_Event_Count();                          // get the event number
};
```

figure 2.1 Class Event

Besides the constructor, there are 5 get functions respond the 5 private member and provide the interface to get these value to outside and another one function update provide the interface to change the event time value.

2.3 Design Initial and Other Functions

The initial function includes reading the initial file to the system event and using these data to initialize events. The other functions contain save bus information, save stop information, record sequence length, calculate the average waiting length, generate the random exponential number and compare the events priority. This header file is defined as figure 2.2.

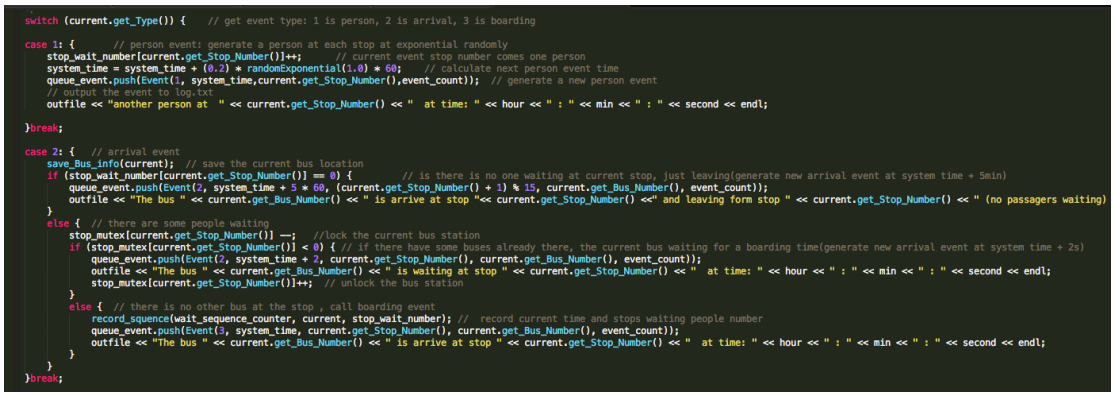
```
/* compare function: makesure the priority queue as  **
** ascending order by time, if events time are same, **
** the earlier event has higher priority            */
struct compare {
    bool operator()(Event &a, Event &b) {
        if(a.get_time() > b.get_time())
            return true;
        else {
            if (a.get_time() == b.get_time())
                return a.get_Event_Count() > b.get_Event_Count();
            else
                return false;
        }
    }
};

void init_Event(priority_queue<Event, vector<Event>, compare> &queue_event); // initial event
double randomExponential(double lambda);        // generater a exponential random number, used from http://www.cnblogs.com/yeahgis/archive/2012/07/15/2592687.html
void save_Bus_info(Event event);                // save bus info at .txt as <time , current_location>

// save stop waiting people number at .txt,include max and min waiting length
void save_Stop_info(int hourly_time, int stop_number, double wait_number);

// save 8 hours total average  , max average and min average of each stop
void save_total_average_stop_info(int stop_number, double max, double min, double total_average);

// save stop waiting people number at every event time
void save_Stop_Waiting_info(vector<unordered_map<double, int> > &wait_sequence_counter);

// record every discreet time the stop waiting people number
void record_squence(vector<unordered_map<double, int> > &wait_sequence_counter, Event event, vector<int> stop_wait_number);

// calculate the average waiting sequence and record the max and min waiting length hourly
void calculate_average(vector<unordered_map<double, int> > &wait_sequence_counter,vector<int> &min_wait_length, vector<int> max_wait_length);
```

figure 2.2 Initial.h

2.4 Design the Simulation Process

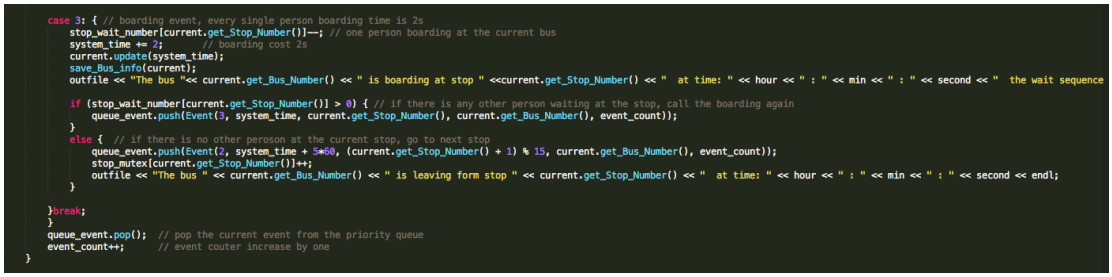
The simulation process has a switch action. Choosing the respond action based on event type. The actions have three cases: person, arrival and boarding. Execute every event will generate another event at the following time which means this what the

current event do next time. This structure is implemented as figure 2.3.1 and figure 2.3.2.



```
switch (current.get_Type()) {    // get event type: 1 is person, 2 is arrival, 3 is boarding

case 1: {       // person event: generate a person at each stop at exponential randomly
    stop_wait_number[current.get_Stop_Number()]++;      // current event stop number comes one person
    system_time = system_time + (0.2) * randomExponential(1.0) * 60;     // calculate next person event time
    queue_event.push(Event(1, system_time,current.get_Stop_Number(),event_count));  // generate a new person event
    // output the event to log.txt
    outfile << "another person at  " << current.get_Stop_Number() << "  at time: " << hour << " : " << min << " : " << second << endl;

}break;

case 2: {   // arrival event
    save_Bus_info(current);  // save the current bus location
    if (stop_wait_number[current.get_Stop_Number()] == 0) {        // is there is no one waiting at current stop, just leaving(generate new arrival event at system time + 5min)
        queue_event.push(Event(2, system_time + 5 * 60, (current.get_Stop_Number() + 1) % 15, current.get_Bus_Number(), event_count));
        outfile << "The bus " << current.get_Bus_Number() << " is arrive at stop "<< current.get_Stop_Number() <<" and leaving form stop " << current.get_Stop_Number() << " (no passagers waiting)
    }
    else {  // there are some people waiting
        stop_mutex[current.get_Stop_Number()] --;   //lock the current bus station
        if (stop_mutex[current.get_Stop_Number()] < 0) { // if there have some buses already there, the current bus waiting for a boarding time(generate new arrival event at system time + 2s)
            queue_event.push(Event(2, system_time + 2, current.get_Stop_Number(), current.get_Bus_Number(), event_count));
            outfile << "The bus " << current.get_Bus_Number() << " is waiting at stop " << current.get_Stop_Number() << "  at time: " << hour << " : " << min << " : " << second << endl;
            stop_mutex[current.get_Stop_Number()]++;  // unlock the bus station
        }
        else {  // there is no other bus at the stop , call boarding event
            record_squence(wait_sequence_counter, current, stop_wait_number); //  record current time and stops waiting people number
            queue_event.push(Event(3, system_time, current.get_Stop_Number(), current.get_Bus_Number(), event_count));
            outfile << "The bus " << current.get_Bus_Number() << " is arrive at stop " << current.get_Stop_Number() << "  at time: " << hour << " : " << min << " : " << second << endl;
        }
    }
}break;
```

figure 2.3.1 Action Case Implemented_1



```
    case 3: { // boarding event, every single person boarding time is 2s
        stop_wait_number[current.get_Stop_Number()]--; // one person boarding at the current bus
        system_time += 2;       // boarding cost 2s
        current.update(system_time);
        save_Bus_info(current);
        outfile << "The bus "<< current.get_Bus_Number() << " is boarding at stop " <<current.get_Stop_Number() << "  at time: " << hour << " : " << min << " : " << second << "  the wait sequence

        if (stop_wait_number[current.get_Stop_Number()] > 0) { // if there is any other person waiting at the stop, call the boarding again
            queue_event.push(Event(3, system_time, current.get_Stop_Number(), current.get_Bus_Number(), event_count));
        }
        else {  // if there is no other peroson at the current stop, go to next stop
            queue_event.push(Event(2, system_time + 5*60, (current.get_Stop_Number() + 1) % 15, current.get_Bus_Number(), event_count));
            stop_mutex[current.get_Stop_Number()]++;
            outfile << "The bus " << current.get_Bus_Number() << " is leaving form stop " << current.get_Stop_Number() << "  at time: " << hour << " : " << min << " : " << second << endl;
        }

    }break;
    }
    queue_event.pop();  // pop the current event from the priority queue
    event_count++;      // event couter increase by one
}
```

figure 2.3.2 Action Case Implemented_2

3 Simulation

The initial parameters are:

Bus: 5

Stops: 15

Boarding time is 2 seconds each person

The bus go to another bus stop need 5 min

Observation

At start time, 5 buses are evenly distributed in 15 stops. However, with time increasing and more and more people enter the station, the buses' distance are reduced. At later time, some buses are going after other buses. Finally, the 5 buses are gathering in one or two of the 15 stops, and keep running. Make the average waiting people number increasing. Thus, the distance between the buses is not uniform. The distance plot shows as figure 3.1. Red, green, black, yellow and blue represent 5 buses respectively. When some buses at the same stop, their plot might be covered by the latest plot, that's why at later time there are only two obviously color in the figure.

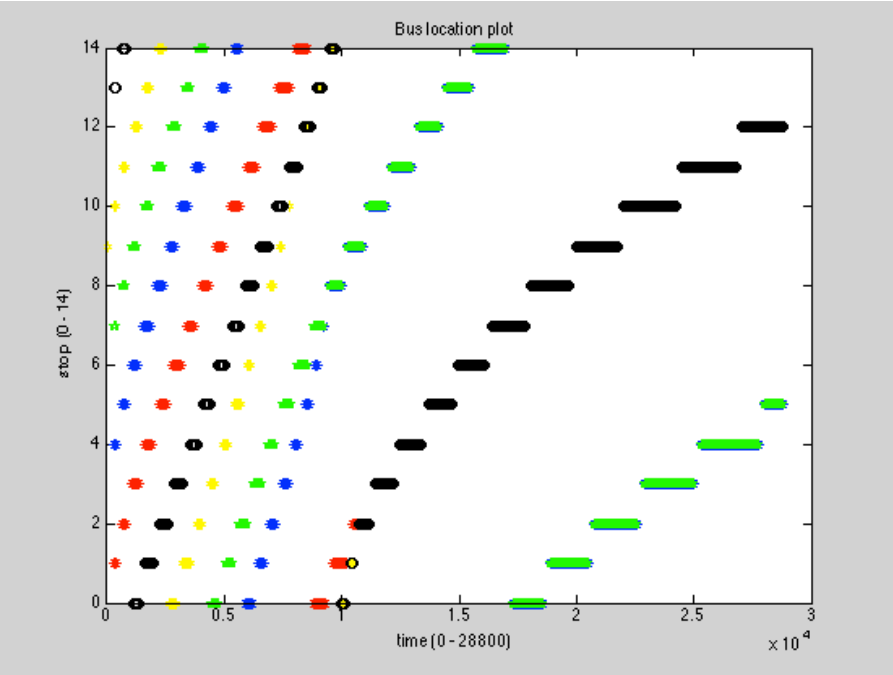


figure 3.1 Bus Location Plot

Recommendations

The way to keep the bus distance uniform is make sure all buses stop together and move together. Thus, when one of the five buses first finishes boarding, it goes to the next stop. At the same time, all other buses start to move to next stop to keep the distance uniform. Through simulation, the buses' location shows as figure 3.2. Red, green, black, yellow and blue represent 5 buses respectively.

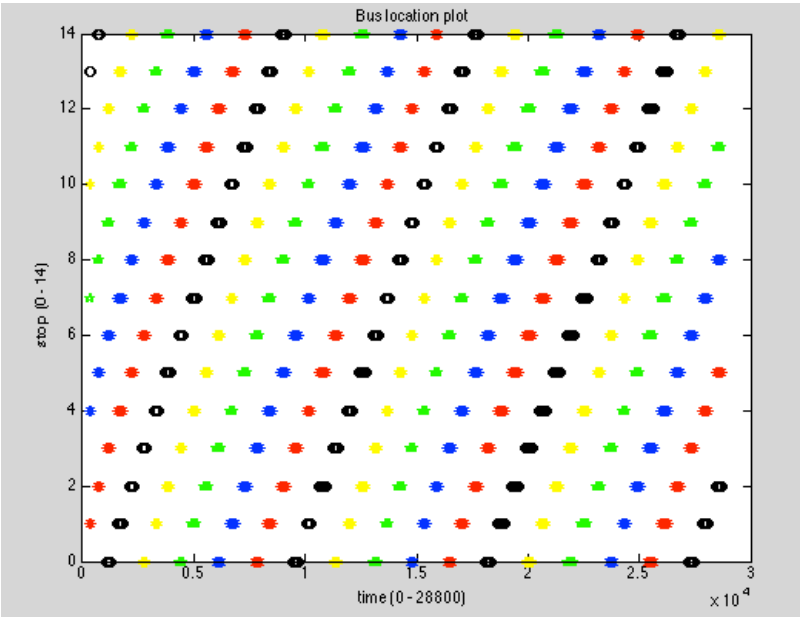


figure 3.2 Bus Location Plot

According to figure 3.2, the five color represent five buses distribute evenly in the

figure which means the buses' distance keep uniform.

Analyzing Stop Waiting Queue Size

Before use uniform method, the average waiting people size of each stop in the third hour are described in table 3.1. The rest information is in the Stop_waiting_length_hourly.txt

Table 3.1 The Bus Stop Waiting Information

Stop Number	Time/hour	Average
0	3	102.042
1	3	116.625
2	3	135.292
3	3	107.667
4	3	75.125
5	3	77.25
6	3	82.9583
7	3	83.375
8	3	86.25
9	3	104.708
10	3	89.4167
11	3	98.25
12	3	70.2917
13	3	70.2083
14	3	91.2917

The whole 8 hours' average waiting people number is described in table 3.2.

Table 3.2 8 Hours Average Waiting Number

Stop Number	Total Average	Max Average	Min Average
0	251.639	711.333	54.9375
1	234.784	508.333	52.3667
2	237.646	666	53.6875
3	271.22	712.4	60.6333
4	295.681	802	55.9333
5	340.233	994	56.75
6	305.957	909.667	55.6875
7	280.748	769.333	46.5333
8	249.142	590	51.25
9	238.844	606.25	51.8438
10	274.896	679.75	52.9688
11	269.996	779	54.8125
12	317.142	840.667	59.5938
13	323.256	979	55.3125
14	282.88	821.667	52.1875

After using uniform method, the average waiting people size of each stop in the third hour are described in table 3.3. The rest information is in the Stop_waiting_length_hourly_optimization.txt

Table 3.3 The Bus Stop Waiting Information

Stop Number	Time/hour	Average
0	3	89.2727
1	3	149.545
2	3	136.545
3	3	150.545
4	3	119.455
5	3	102.364
6	3	73
7	3	79.9091
8	3	92.6364
9	3	88.0909
10	3	110.636
11	3	118.455
12	3	113.727
13	3	72.1818
14	3	75.6364

The whole 8 hours' average waiting people number is described in table 3.2.

Table 3.4 8 Hours Average Waiting Number

Stop Number	Total Average	Max Average	Min Average
0	82.3545	91.5	65.7
1	131.861	168	88.25
2	101.726	136.545	61.8333
3	152.095	181.727	85.9167
4	140.558	184.636	78.25
5	91.8254	122.909	61.75
6	76.141	90.9	59.8333
7	92.5077	115.273	50.75
8	76.5169	92.6364	51.8333
9	108.545	151.333	68.25
10	145.739	204.364	61.1667
11	100.543	118.455	70
12	103.411	113.727	69.1667
13	70.3247	84.5455	60.25
14	88.5326	117	56.0833

Figure 3.3 shows the difference between the two methods.

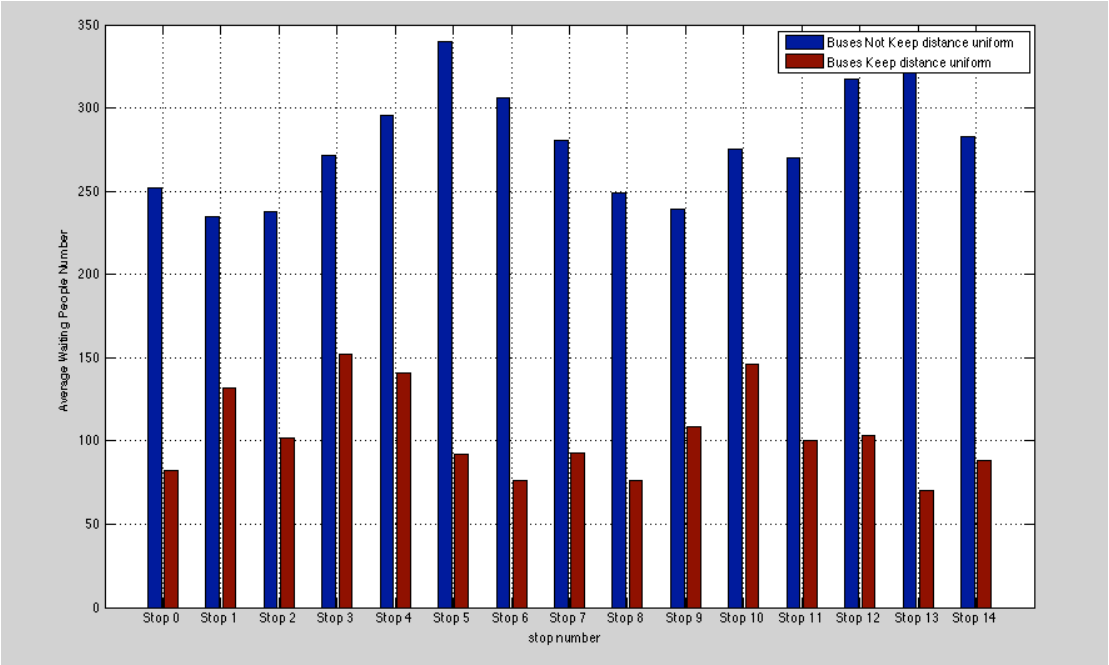


figure 3.3 Difference between two methods

Based on figure 3.3 and table 3.2, 3.4. The uniform method lead the average waiting people sequence size reduced significantly, which means making buses keep uniform distance will let the system work more efficiently.

4. Conclusions

The bus simulation runs well. We can draw the conclusion based on the waiting people queue size that the buses' distance keep uniform is better than they are not uniform to the system. The key keep the distance uniform is make sure every bus moves and stops together. Thus, the system will be more efficiency.

Reference:

The system designed idea is learned from lecture3.

The generate random exponential number function uses the code from:
http://www.cnblogs.com/yeahgis/archive/2012/07/15/2592687.html