**IE 306 Homework 2 – Murat Tutar – 2020402264**

**Q1:**

Let X be the distribution of interarrival times. **X~U(5,17)**

Let Y be the distribution of 1st server's service time. **Y~U(15,22)**

2nd server's service time data is already given in the question as the table below:

| Service time | Probability | Cumulative |
|---|---|---|
| 5 mins | 0.13 | 0.13 |
| 14 mins | 0.35 | 0.48 |
| 24 mins | 0.40 | 0.88 |
| 35 mins | 0.12 | 1 |

The list of the random numbers is below:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0.497 | 0.380 | 0.862 | 0.020 | 0.391 | 0.975 | 0.480 | 0.905 | 0.759 | 0.560 | 0.593 |

**Interarrival Time** = 5 + (17-5)*RN

**First Server's Service Time** = 15 + (22-15)*RN

**Second Server's Service Time** will be decided according to the tables above.

I made the necessary calculations with R. The results are below.

```
> Q1<-function(RN, x) {
+     if (x==0) { #0 represents interarrival time
+         return (5+12*RN)
+     }
+     if (x==1) { #1 represents service time in server1
+         return (15+7*RN)
+     }
+     if (x==2) { #2 represents service time in server2
+         if (RN<=0.13)
+             return (5)
+         else if (RN<=0.48)
+             return (15)
+         else if (RN<=0.88)
+             return (25)
+         else
+             return(35)
+     }
+ }
> Q1(0.497,0) #interarrival time at t=0
[1] 10.964
> Q1(0.380,1) #service time for server 1 at t=0
[1] 17.66
> Q1(0.862,0) #interarrival time at t=10.964
[1] 15.344
> Q1(0.020,2) #service time for server 2 at t=10.964
[1] 5
> Q1(0.391,0) #interarrival time at t=26.308
[1] 9.692
> Q1(0.975,1) #service time for server 1 at t=26.308
[1] 21.825
```

I tabulated below the results I obtained with my calculations.

| Time | State Variables | | | Interarrival Time | Service Time | | Future Event List | | | Set | Time in Queue |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | LQ | LS1 | LS2 | | Server1 | Server2 | A | D1 | D2 | | |
| 0 | 0 | 1 | 0 | 10.964 | 17.66 | - | 10.964* | 17.66 | ∞ | 0 | - |
| 10.964 | 0 | 1 | 1 | 15.344 | - | 5 | 26.308 | 17.66 | 15.964* | 0<br>10.964 | - |
| 15.964 | 0 | 1 | 0 | - | - | - | 26.308 | 17.66* | ∞ | 0 | - |
| 17.66 | 0 | 0 | 0 | - | - | - | 26.308* | ∞ | ∞ | | - |
| 26.308 | 0 | 1 | 0 | 9.692 | 21.825 | - | 36* | 48.133 | ∞ | 26.308 | - |
| 30 | 0 | 1 | 0 | - | - | - | 36* | 48.133 | ∞ | 26.308 | - |

**Q2:**

**a-)** My C++ code for the solution of part b is below. After the code, I uploaded the screenshot of my output table. I compared my table with Q1 and the results are same.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

//Q2a

using namespace std;

//I define the variables first
double RandomNumbers[11] = { 0.497, 0.380, 0.862, 0.020, 0.391, 0.975, 0.480, 0.905,
0.759, 0.560, 0.593}; //list of the RNs given
double CustomersInQueue; //The number of customers in queue
double Clock; //for timing
double ImminentEventTime; //The time of the first coming event
double InterarrivalTime; //interarrival time
double ServiceTime1; //server1 service time
double ServiceTime2; //server2 service time
bool LQ; //LQ
bool LS1; //LS1
bool LS2; //LS2
```

```
double M; //a large number
int i; //will be used for iterative operations
vector <double> FutureEventList{ 0,0,HUGE }; //vector of future events. The indexes
represent arrivals, departure1, departure2, respectively.
vector<double> SET; //the vector representing the values in set

void Initialize() { //I initialize the variables
	CustomersInQueue = 0;
	Clock = 0;
	ImminentEventTime = 0;
	InterarrivalTime = 0;
	ServiceTime1 = 0;
	ServiceTime2 = 0;
	LQ = false;
	LS1 = false;
	LS2 = false;
	M = HUGE;
	i = 0;
}

double InterarrivalGenerator(double RN) { //generates interarrival times according to
the question
	return 5 + (17 - 5) * RN;
}

double ServiceTimeGenerator(double RN, int ServerNumber) { //generates service times
according to the question
	if (ServerNumber == 1) //if the customer is in first server
		return 15 + (22 - 15) * RN;
	else { //if the customer is in second server
		if (RN <= 0.13) return 5;
		else if (RN <= 0.48) return 15;
		else if (RN <= 0.88) return 25;
		else return 35;
	}
}

void CheckOut() { //to slide the queue after a departure
	i = 2;
	while (i != SET.size() - 1) {
		SET[i] = SET[i + 1];
		i++;
	}
	SET.pop_back();
}

void Arrival() { //Code for arrival process

	if (CustomersInQueue == 0) //we set LQ = false if there is no queue
		LQ = false;



	if (LQ == false) { //if there is no queue we try to enter an idle server
		if (LS1 == true && LS2 == true) { //if both servers are busy, customer
waits in the queue
			LQ = true;
			CustomersInQueue++;
			InterarrivalTime = InterarrivalGenerator(RandomNumbers[i]);
			FutureEventList[0] = Clock +
InterarrivalGenerator(RandomNumbers[i]);
			i++;
```

```
                        SET.push_back(Clock);
                }

                else if (LS1 == true && LS2 == false) { //customer enters server2 when
server1 is busy
                        LS2 = true;
                        InterarrivalTime = InterarrivalGenerator(RandomNumbers[i]);
                        FutureEventList[0] = Clock + InterarrivalTime; ////
                        i++;
                        ServiceTime2 = ServiceTimeGenerator(RandomNumbers[i], 2);
                        i++;
                        FutureEventList[2] = Clock + ServiceTime2;
                        if (SET.size() == 1) SET.push_back(Clock);
                        else SET[1] = Clock;
                }

                else if (LS1 == false && LS2 == true) { //customer enters server1 when
server2 is busy
                        LS1 = true;
                        ServiceTime1 = ServiceTimeGenerator(RandomNumbers[i], 1);
                        i++;
                        FutureEventList[1] = Clock + ServiceTime1;
                        InterarrivalTime = InterarrivalGenerator(RandomNumbers[i]);
                        FutureEventList[0] = Clock + InterarrivalTime;
                        i++;
                        if (SET.size() == 1) SET.insert(SET.begin(), Clock);
                        else SET[0] = Clock;
                }

                else { //customer enters server1 when both servers are idle
                        LS1 = true;
                        InterarrivalTime = InterarrivalGenerator(RandomNumbers[i]);
                        FutureEventList[0] = Clock + InterarrivalTime;
                        i++;
                        ServiceTime1 = ServiceTimeGenerator(RandomNumbers[i], 1);
                        i++;
                        FutureEventList[1] = Clock + ServiceTime1;
                        SET.insert(SET.begin(), Clock);
                }

        }

        else { //if there is already a queue, customer joins the queue
                CustomersInQueue++;
                InterarrivalTime = InterarrivalGenerator(RandomNumbers[i]);
                FutureEventList[0] = Clock + InterarrivalTime;
                i++;
                SET.push_back(Clock);
        }
}

void Departure(int ServerNumber) { //code for departure process

        if (ServerNumber == 1) { //code for first server

                if (CustomersInQueue > 0) { //customer from the queue enters the server
                        ServiceTime1 = ServiceTimeGenerator(RandomNumbers[i], 1);
                        i++;
                        FutureEventList[1] = Clock + ServiceTime1;
                        CustomersInQueue -= 1;

                        if (CustomersInQueue == 0) LQ = false;
```

```
                CheckOut(); //slides the queue
            }

            else { //if there is no queue when customer in server1 departs
                SET.erase(SET.begin());
                FutureEventList[1] = M;
                LS1 = false;
            }


        }

        else if (ServerNumber == 2) { //similar algorithm for server2

            if (CustomersInQueue > 0) {
                ServiceTime2 = ServiceTimeGenerator(RandomNumbers[i], 2);
                i++;
                FutureEventList[2] = Clock + ServiceTime2;
                CustomersInQueue -= 1;

                if (CustomersInQueue == 0) LQ = false;

                SET[1] = SET[2];
                CheckOut();
            }

            else {
                if (SET.size() > 1) SET.erase(SET.begin() + 1);
                FutureEventList[2] = M;
                LS2 = false;
            }
        }

}

void TimeAdvance() { //code for timing process

        ImminentEventTime = *min_element(FutureEventList.begin(),
FutureEventList.end()); //time of the closest future event
        Clock = ImminentEventTime;
        if (Clock < 30) {
            //comparing the indexes of the FutureEventList to determine which of them
is coming first.
            //clock is until 30 since the simulation is for 30 minutes
            if (FutureEventList[0] < FutureEventList[1] && FutureEventList[0] <
FutureEventList[2]) Arrival();
            else if (FutureEventList[1] < FutureEventList[2] && FutureEventList[1] <
FutureEventList[0]) Departure(1);
            else if (FutureEventList[2] < FutureEventList[1] && FutureEventList[2] <
FutureEventList[0]) Departure(2);
            else Arrival(); //when there is a simultaneous scheduling, we begin with
arrival.
        }
        else (Clock = 30); //the simulation is for 30 minutes

}


void PrintSET(vector<double> Vect) { //prints the values in SET
        cout << "SET: ";
        if (Vect.size() == 0) cout << "-";
        else {
```

```
			cout << "[ " << Vect[0];
			if (Vect.size() > 1) {
				for (int i = 1; i < Vect.size(); i++)
					cout << " , " << Vect[i];
			}
			cout << " ]";
		}
		cout << endl;
}

void PrintFEL(vector<double> Vect) { //prints the values in Future Event List.
		cout << "FUTURE EVENT LIST:";
		cout << "	Arrival: " << Vect[0] << " ";
		cout << "	Departure 1: " << Vect[1];
		cout << "	Departure 2: " << Vect[2] << endl;
}

int main() {
		cout << endl;
		cout << "Random Numbers: {0.497,  0.380,  0.862 ,  0.020 , 0.391,  0.975,  0.480
, 0.905 , 0.759 , 0.560 , 0.593}" << endl << endl; //list of the RNs given in the
question
		Initialize(); //initializing the variables
		while (Clock < 30) { //simulation for 30 minutes
			TimeAdvance();
			cout <<
"*************************************************************************************
*****************" << endl;
			cout << "Time: " << Clock;
			cout << " | LQ: " << LQ;
			cout << " | LS1: " << LS1;
			cout << " | LS2: " << LS2;
			if (InterarrivalTime == 0) cout << " | Interarrival Time: - ";
			else { cout << " | Interarrival Time: " << InterarrivalTime;
InterarrivalTime = 0; }
			if (ServiceTime1 == 0) cout << " | Service Time 1: - ";
			else { cout << " | Service Time 1: " << ServiceTime1; ServiceTime1 = 0; }
			if (ServiceTime2 == 0) cout << " | Service Time 2: - " << endl;
			else { cout << " | Service Time 2: " << ServiceTime2 << endl;
ServiceTime2 = 0; }
			ServiceTime2 = 0;
			PrintFEL(FutureEventList);
			PrintSET(SET);
			cout << endl;
			cout << endl;
		}

		return 0;
}
```

The output of the C++ code for Q2a is below. The results are equal to the results of Q1.

```
Random Numbers: {0.497,  0.380,  0.862 ,  0.020 , 0.391,  0.975,  0.480 , 0.905 , 0.759 , 0.560 , 0.593}

************************************************************************************************
Time: 0 | LQ: 0 | LS1: 1 | LS2: 0 | Interarrival Time: 10.964 | Service Time 1: 17.66 | Service Time 2: -
FUTURE EVENT LIST:      Arrival: 10.964         Departure 1: 17.66      Departure 2: inf
SET: [ 0 ]


************************************************************************************************
Time: 10.964 | LQ: 0 | LS1: 1 | LS2: 1 | Interarrival Time: 15.344 | Service Time 1: -  | Service Time 2: 5
FUTURE EVENT LIST:      Arrival: 26.308         Departure 1: 17.66      Departure 2: 15.964
SET: [ 0 , 10.964 ]


************************************************************************************************
Time: 15.964 | LQ: 0 | LS1: 1 | LS2: 0 | Interarrival Time: -  | Service Time 1: -  | Service Time 2: -
FUTURE EVENT LIST:      Arrival: 26.308         Departure 1: 17.66      Departure 2: inf
SET: [ 0 ]


************************************************************************************************
Time: 17.66 | LQ: 0 | LS1: 0 | LS2: 0 | Interarrival Time: -  | Service Time 1: -  | Service Time 2: -
FUTURE EVENT LIST:      Arrival: 26.308         Departure 1: inf        Departure 2: inf
SET: -


************************************************************************************************
Time: 26.308 | LQ: 0 | LS1: 1 | LS2: 0 | Interarrival Time: 9.692 | Service Time 1: 21.825 | Service Time 2: -
FUTURE EVENT LIST:      Arrival: 36             Departure 1: 48.133     Departure 2: inf
SET: [ 26.308 ]


************************************************************************************************
Time: 30 | LQ: 0 | LS1: 1 | LS2: 0 | Interarrival Time: -  | Service Time 1: -  | Service Time 2: -
FUTURE EVENT LIST:      Arrival: 36             Departure 1: 48.133     Departure 2: inf
SET: [ 26.308 ]
```

**b-)** My C++ code for the solution of part b is below. After the code, I uploaded the results of the 4 simulations I have done. In addition to that, I verified that Little's Law approximately holds as can be seen in the end of my code and output.

```cpp
#include <iostream>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <vector>
#include <algorithm>

//Q2b

using namespace std;

//I define the variables first
double WaitingTime_Total; //total waiting time
double Service1_TotalTime; //total time spent in server1
double Service2_TotalTime; //total time spent in server2
double CustomersInQueue; //number of customers in queue
double NumberOfCustomers; //number of customers
double Clock; //for timing
double ImminentEventTime; //the time of the closest event
double InterarrivalTime; //interarrival time
double ServiceTime1; //server1 service time
```

```
double ServiceTime2; //server2 service time
double customerSum; //used for calculating average number of customers in the system
bool LQ; //LQ
bool LS1; //LS1
bool LS2; //LS2
double M; //large number
int i; //will be used for iterative operations
vector <double> FutureEventList; //vector of future events.
vector <double> SET; //the vector representing the values in set

void Initialize() { //I initialize the variables
	WaitingTime_Total = 0;
	Service1_TotalTime = 0;
	Service2_TotalTime = 0;
	CustomersInQueue = 0;
	NumberOfCustomers = 0;
	Clock = 0;
	ImminentEventTime = 0;
	InterarrivalTime = 0;
	ServiceTime1 = 0;
	ServiceTime2 = 0;
	customerSum = 0;
	LQ = false;
	LS1 = false;
	LS2 = false;
	M = 5035.00001; //The maximum number that can be obtained in this simulation is 
5000+35=5035, this number is slightly larger than it.
	i = 0;
	SET.clear();
	FutureEventList.push_back(0); //arrivals
	FutureEventList.push_back(0); //departure1
	FutureEventList.push_back(M); //departure2

}

double InterarrivalGenerator() { //generates interarrival times according to the 
question
	double RN = (double)(rand() % RAND_MAX) / (double) RAND_MAX;
	return 5 + (17 - 5) * RN;
}

double ServiceTimeGenerator(int ServerNumber) { //generates service times according to 
the question
	double RN = (double) (rand() % RAND_MAX) / (double) RAND_MAX;
	if (ServerNumber == 1) //if the customer is in first server
		return 15 + (22 - 15) * RN;
	else { //if the customer is in second server
		if (RN <= 0.13) return 5;
		else if (RN <= 0.48) return 15;
		else if (RN <= 0.88) return 25;
		else return 35;
	}
}

void CheckOut() { //to slide the queue after a departure
	i = 2;
	while (i != SET.size() - 1) {
		SET[i] = SET[i + 1];
		i++;
	}
	SET.pop_back();
}
```

```
void Arrival() { //code for arrival process

	NumberOfCustomers++; //a new customer arrived. therefore it increases.

	if (CustomersInQueue == 0) { //we set LQ = false if there is no queue
		LQ = false;
	}

	if (LQ == false) { //if there is no queue we try to enter an idle server
		if (LS1 == true && LS2 == true) { //if both servers are busy, customer
waits in the queue
			LQ = true;
			CustomersInQueue++;
			InterarrivalTime = InterarrivalGenerator();
			FutureEventList[0] = Clock + InterarrivalGenerator();
			i++;
			SET.push_back(Clock);
		}

		else if (LS1 == true && LS2 == false) { //customer enters server2 when
server1 is busy
			LS2 = true;
			InterarrivalTime = InterarrivalGenerator();
			FutureEventList[0] = Clock + InterarrivalTime; ////
			i++;
			ServiceTime2 = ServiceTimeGenerator(2);
			i++;
			FutureEventList[2] = Clock + ServiceTime2;
			if (SET.size() == 1) SET.push_back(Clock);
			else SET[1] = Clock;
		}

		else if (LS1 == false && LS2 == true) { //customer enters server1 when
server2 is busy
			LS1 = true;
			ServiceTime1 = ServiceTimeGenerator(1);
			i++;
			FutureEventList[1] = Clock + ServiceTime1;
			InterarrivalTime = InterarrivalGenerator();
			FutureEventList[0] = Clock + InterarrivalTime; ////
			i++;
			if (SET.size() == 1) SET.insert(SET.begin(), Clock); ////
			else SET[0] = Clock;
		}

		else { //customer enters server1 when both servers are idle
			LS1 = true;
			InterarrivalTime = InterarrivalGenerator();
			FutureEventList[0] = Clock + InterarrivalTime;
			i++;
			ServiceTime1 = ServiceTimeGenerator(1);
			i++;
			FutureEventList[1] = Clock + ServiceTime1;
			SET.insert(SET.begin(), Clock);
		}

	}

	else { //if there is already a queue, customer joins the queue
		CustomersInQueue++;
		InterarrivalTime = InterarrivalGenerator();
```

```
            FutureEventList[0] = Clock + InterarrivalTime;
            i++;
            SET.push_back(Clock);
        }
}

void Departure(int ServerNumber) { //code for departure process

        if (ServerNumber == 1) { //code for first server

                if (CustomersInQueue > 0) { //customer from the queue enters the server
                        ServiceTime1 = ServiceTimeGenerator(1);
                        i++;
                        FutureEventList[1] = Clock + ServiceTime1;
                        CustomersInQueue -= 1;

                        if (CustomersInQueue == 0) LQ = false;
                        CheckOut(); //slides the queue
                }

                else { //if there is no queue when customer in server1 departs
                        if(SET.size()>0) SET.erase(SET.begin());
                        FutureEventList[1] = M;
                        LS1 = false;
                }


        }

        else if (ServerNumber == 2) { //similar algorithm for server2

                if (CustomersInQueue > 0) {
                        ServiceTime2 = ServiceTimeGenerator(2);
                        i++;
                        FutureEventList[2] = Clock + ServiceTime2;
                        CustomersInQueue -= 1;

                        if (CustomersInQueue == 0) LQ = false;

                        SET[1] = SET[2];
                        CheckOut();
                }

                else {
                        if (SET.size() > 1) SET.erase(SET.begin() + 1); ////
                        FutureEventList[2] = M;
                        LS2 = false;
                }
        }

}

void TimeAdvance() { //code for timing process

        ImminentEventTime = *min_element(FutureEventList.begin(),
FutureEventList.end()); //time of the closest future event
        WaitingTime_Total += (CustomersInQueue * (ImminentEventTime - Clock));
//calculates the total waiting time
        if (ImminentEventTime <= 5000) customerSum = customerSum + (ImminentEventTime -
Clock) * (LQ + LS1 + LS2); //calculates the customerSum to find avg # of customers in
the system
```

```
	else customerSum = customerSum + (5000 - Clock) * (LQ + LS1 + LS2); //if the 
next event is after 5000th minute, we will only calculate until 5000th minute
	Clock = ImminentEventTime;
	if (Clock < 5000) {
		//comparing the indexes of the FutureEventList to determine which of them 
is coming first.
		//clock is until 5000 since the simulation is for 5000 minutes
		if (FutureEventList[0] < FutureEventList[1] && FutureEventList[0] < 
FutureEventList[2]) Arrival();
		else if (FutureEventList[1] < FutureEventList[2] && FutureEventList[1] < 
FutureEventList[0]) Departure(1);
		else if (FutureEventList[2] < FutureEventList[1] && FutureEventList[2] < 
FutureEventList[0]) Departure(2);
		else Arrival(); //when there is a simultaneous scheduling, we begin with 
arrival.
	}
	else {
		Service1_TotalTime = Service1_TotalTime - (FutureEventList[1] - 5000); 
//we subtract the extra time after 5000th minute
		Service2_TotalTime = Service2_TotalTime - (FutureEventList[2] - 5000); 
//we subtract the extra time after 5000th minute
		(Clock = 5000); //the simulation is for 5000 minutes
	}

}


void PrintSET(vector<double> Vect) { //prints the values in SET
	cout << "SET: ";
	if (Vect.size() == 0) cout << "-";
	else {
		cout << "[ " << Vect[0];
		if (Vect.size() > 1) {
			for (int i = 1; i < Vect.size(); i++)
				cout << " , " << Vect[i];
		}
		cout << " ]";
	}
	cout << endl;
}

void PrintFEL(vector<double> Vect) { //prints the values in Future Event List
	cout << "FUTURE EVENT LIST:";
	cout << "     Arrival: " << Vect[0] << "  ";
	cout << "     Departure 1: " << Vect[1];
	cout << "     Departure 2: " << Vect[2] << endl;
}

int main() {

	srand(time(NULL)); //to generate different random variables

	Initialize(); //initializing the variables

	while (Clock < 5000) { //simulation for 5000 minutes
		TimeAdvance();
		Service1_TotalTime += ServiceTime1; //we add the respective service time 
to the total service time in each iteration
		ServiceTime1 = 0; //to avoid adding the service time which was already 
added during the previous iteration
		Service2_TotalTime += ServiceTime2; //we add the respective service time 
to the total service time in each iteration
```

```
            ServiceTime2 = 0; //to avoid adding the service time which was already added during the previous iteration
        }

        cout << "Average Time Spent in Queue: " << WaitingTime_Total / NumberOfCustomers << endl;
        cout << "Average Number of Customers in the System: " << customerSum / 5000 << endl;
        cout << "Utilization of Server 1: " << (Service1_TotalTime) / 5000 << endl;
        cout << "Utilization of Server 2: " << (Service2_TotalTime) / 5000 << endl;
        cout << "Probability of a Customer not waiting in the queue: " << 1 - WaitingTime_Total / 5000 << endl;
        cout << endl;

        //the code below is to verify if Little's Law holds
        /*Little's Law states that the average number of customers in the system L,
        is equal to the multiplication of arrival rate lambda
        by the average time that a customer spends in the store W*/

        double L = customerSum / 5000; //average number of customers
        double W = (Service1_TotalTime + Service2_TotalTime + WaitingTime_Total) / NumberOfCustomers; //average time a customer spends in the system
        double lambda = NumberOfCustomers / 5000; //average arrival rate

        cout << "Verifying Little's Law:" << endl;
        cout << "L = " << L << endl;
        cout << "W * lambda = " << W * lambda << endl;

        //although the results are not exactly equal, they are approximately close.
        //this shows that our simulation is successful and Little's Law approximately holds.

        return 0;
}
```

### Output of the Simulation #1:

```
Average Time Spent in Queue: 5.78202
Average Number of Customers in the System: 2.18785
Utilization of Server 1: 0.905377
Utilization of Server 2: 0.871
Probability of a Customer not waiting in the queue: 0.463428

Verifying Little's Law:
L = 2.18785
W * lambda = 2.31295
```

### Output of the Simulation #2:

```
Average Time Spent in Queue: 4.56934
Average Number of Customers in the System: 2.05828
Utilization of Server 1: 0.88701
Utilization of Server 2: 0.854088
Probability of a Customer not waiting in the queue: 0.58419

Verifying Little's Law:
L = 2.05828
W * lambda = 2.15691
```

**Output of the Simulation #3:**

```
Average Time Spent in Queue: 3.69409
Average Number of Customers in the System: 2.00162
Utilization of Server 1: 0.8881
Utilization of Server 2: 0.832028
Probability of a Customer not waiting in the queue: 0.664577

Verifying Little's Law:
L = 2.00162
W * lambda = 2.05555
```

**Output of the Simulation #4:**

```
Average Time Spent in Queue: 5.65163
Average Number of Customers in the System: 2.15144
Utilization of Server 1: 0.904805
Utilization of Server 2: 0.877804
Probability of a Customer not waiting in the queue: 0.468747

Verifying Little's Law:
L = 2.15144
W * lambda = 2.31386
```

Although the results of different simulations are varying, they give us a general idea about their approximate values. If we increase the simulation time, we will probably get better results and we will make better estimations. This assumption is also valid for Little’s Law. Although we get close results in our simulations, we can get better results if we increase the time period of the simulation.