COMPENG 4DK4 LAB1

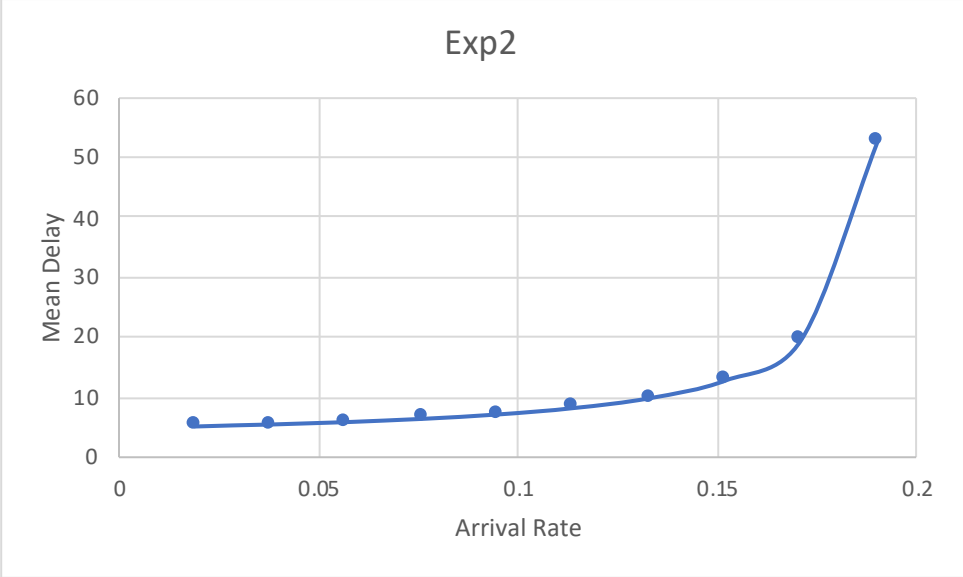
Richard Qiu – 400318681 – Group 120

Experiment

2.

Service Time	Arrival Rate	Mean Delay
5	0.019	5.262567
5	0.038	5.586691
5	0.057	5.996825
5	0.076	6.53254
5	0.095	7.262067
5	0.114	8.313856
5	0.133	9.963165
5	0.152	12.919958
5	0.171	19.748152
5	0.19	52.581758

Plot:



Experiment:
By making ten loops of the arrival rate with each increment of 0.19 and for each arrival loop, increment 9 random seeds with 10e6 and use the student number 400318681 as the last random seed to simulate this experiment. Code is paste in the appendix.

Justification:
As we obtained from the simulation plot, the mean delay axis intercept time at low arrival rate values (below 0.1 arrival rate) has a value around 6.0. As the arrival rate starts increasing and approaching the maximum allowed value 0.2, we can obtain the plot is rising pretty fast when it

overs 0.16 arrival rate and approaches the mean delay time around 60 when it approaches the right-hand inequality.

This shape of the mean delay curve can reflect how the single server queue system handle customers under different unexpected arrival rates by using the Poisson process simulation where the service time is constant (5) here. It can help better simulate the real-word single-serve queuing system when the service time is fixed, in this case, the arrival rate of customers is simulated, and we can understand better how the system will handle under different arrival rates. From this curve, we can find for this system, to allow customers have a low mean delay, the ideal arrival rate should lower than 0.16.

3. When the product of arrival rate and service time is greater than 1, set the arrival rate to 0.201, the following experiments are shown below.

```
Customers served = 10000 (Total arrived = 10057)
Arrival rate is: 0.201000
Utilization = 0.994428
Fraction served = 0.994332
Mean number in system = 41.943639
Mean delay = 210.893212
```

```
Customers served = 50000 (Total arrived = 50340)
Arrival rate is: 0.201000
Utilization = 0.997357
Fraction served = 0.993246
Mean number in system = 132.901952
Mean delay = 666.270394
```

```
Customers served = 100000 (Total arrived = 100965)
Arrival rate is: 0.201000
Utilization = 0.998677
Fraction served = 0.990442
Mean number in system = 327.468627
Mean delay = 1639.512227
```

```
Customers served = 300000 (Total arrived = 301855)
Arrival rate is: 0.201000
Utilization = 0.999559
Fraction served = 0.993855
Mean number in system = 986.015400
Mean delay = 4932.254059
```

When the arrival rate is slightly greater than 1/5=0.2, as we can see from those pictures, the fraction served is no longer equal to 1 which means there are unserved customers in the system and that can also be proved from the total arrived customers is larger than the customers served.

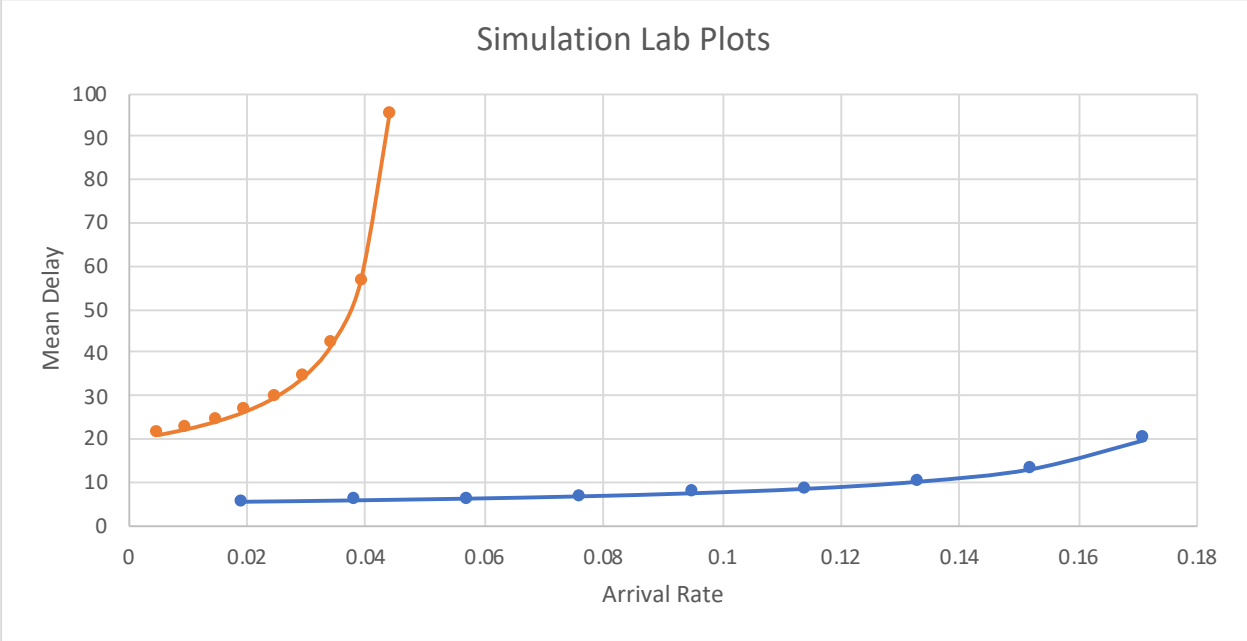
And as we increased the run length of customers, we can see the unserved customers is increased as well. Moreover, the mean number in system and mean delay are both increasing. Why this happen is due to as we increased the total customers, the system don’t have enough time to serve

each customer, so when the arrival rate and service time is not changing, a portion of total customers will always be unserved which that is why the Expression 1 is necessary.

What's more, we can calculate the mean customer interarrival time by the formula 1/ARRIVAL_RATE which is equal to 1/0.201 = 4.975s. And we know the service time for every customer is 5s. So, for every 5s, a portion of 0.025s customers of the total will always not be served which explain why always a portion of customers is arrived but not be served.

4.

Service Time	Arrival Rate	Mean Delay	Analytic Result
20	0.0049	21.087031	21.0864745
20	0.0098	22.438918	22.43781095
20	0.0147	24.165591	24.16430595
20	0.0196	26.44854	26.44736842
20	0.0245	29.608348	29.60784314
20	0.0294	34.271262	34.27184466
20	0.0343	41.849959	41.84713376
20	0.0392	56.311356	56.2962963
20	0.0441	94.798235	94.74576271



As we can see from the above plot, the organe line is where the service time is equal to 20 while arrival rate is within a suitable range. Compared to the mean delay in exp2, we can see an increase of the mean delay happens when it approaches the right-hand equality (0.05 in this case), this is due to because the increase of service time, now the customers have to wait four times longer than before when the serve is full. But overall, the shape of both curves are sharing the same similarities when approach the inequality point.

5.

$\bar{d}_{M/D/1} = \frac{\bar{X}(2-\rho)}{2(1-\rho)}$, by applying this formula where $\rho = \lambda\overline{X}$, the analytic result for parts2 and 4 are shown below.

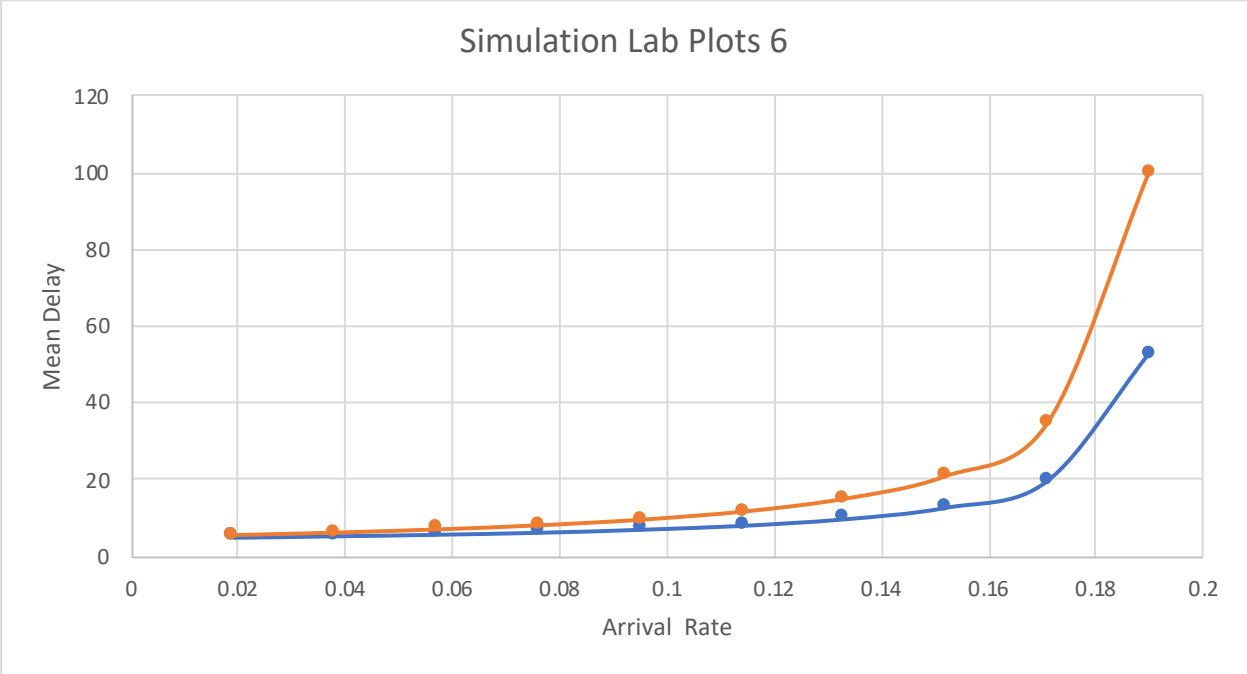
Mean delay(parts2)	Analytic Result	Mean delay(parts4)	Analytic Result
5.262567	5.26243094	21.087031	21.0864745
5.586691	5.58641975	22.438918	22.4378109
5.996825	5.9965035	24.165591	24.1643059
6.53254	6.53225806	26.44854	26.4473684
7.262067	7.26190476	29.608348	29.6078431
8.313856	8.31395349	34.271262	34.2718447
9.963165	9.96268657	41.849959	41.8471338
12.919958	12.9166667	56.311356	56.2962963
19.748152	19.7413793	94.798235	94.7457627
52.581758	52.5	508.523258	510

From the tables, we can compare the simulation and analytic results are very close to each other, which can further prove this formula is correct.

6.

By replacing the fixed service time with an exponential generate service time using the same service value 5, we can obtain the following results.

Arrival Rate	Mean Delay	Analytic Result
0.019	5.524538	5.524861878
0.038	6.172102	6.172839506
0.057	6.992284	6.993006993
0.076	8.06353	8.064516129
0.095	9.52195	9.523809524
0.114	11.626566	11.62790698
0.133	14.923195	14.92537313
0.152	20.82566	20.83333333
0.171	34.490801	34.48275862
0.19	99.89674	100



Where the orange line is the simulation of the experiment 6 using exponential distributed service time and the blue line is the fixed service time in experiment 2.

The reason why the mean delay of the MM1 system is higher than MD1 system when approaching the inequality is because for the MM1 system, when the service time is exponential distributed which means it is stochastic and variable, customers can experience a really different amount of service times. And when the arrival rate of customer is large, that stochastic service time will cause more and more customers to wait due to it's behaviors, the unequal distributed service time can cause a small amount of customers being served very fast but the rest amount of customers being served slow and they have to wait for the queue. And with the increase of arrival rates, the amount of long service time to customers will cause more and more delay. And in contract, with a fixed service time in the MD1 system, the resources of each customers is equally distributed, so they are will experience the same wait time in this case which it will use the resources more efficiently when the arrival rate is approaching the inequality.

The mean delay expression is derived below, and we can calculate the analytic results from this formula. After comparing, we can prove this formula is correct.

$\bar{X}$: mean service time , μ : service rate

Exp distribution: $f(x) = \mu e^{-\mu x}$ (for $x > 0$)

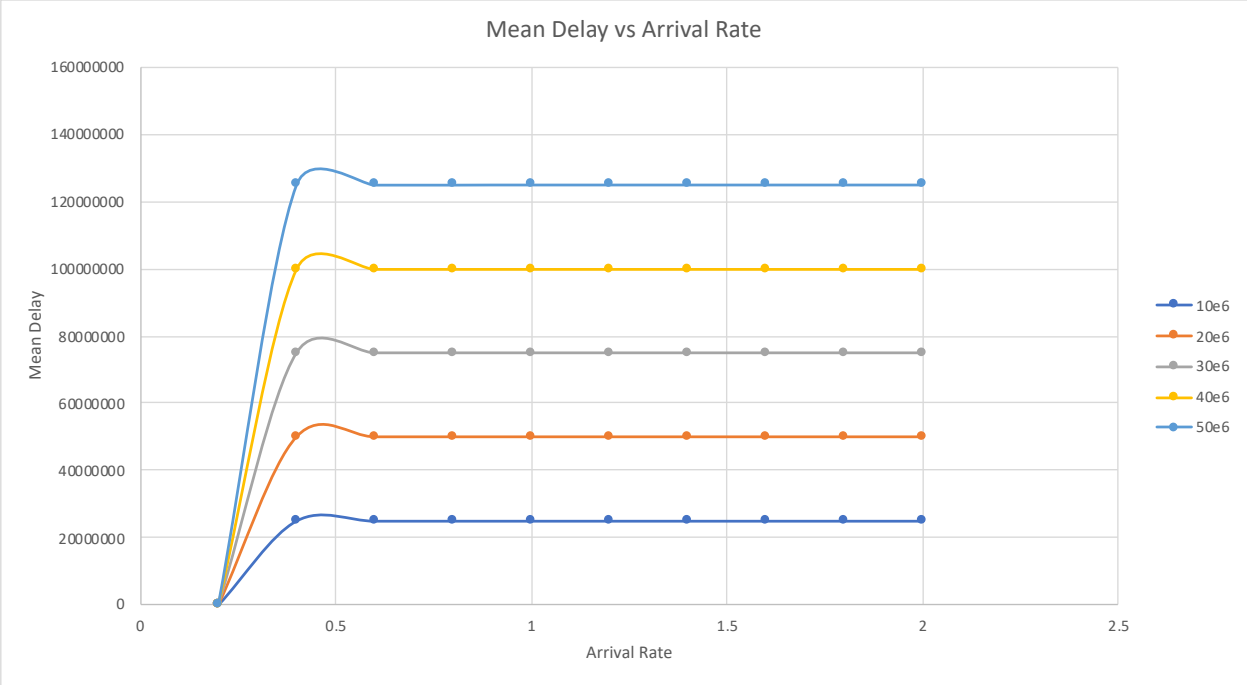
$\bar{X} = \frac{1}{\mu}$ and $\sigma_X = \frac{1}{\mu^2}$, $\bar{X^2} = \frac{2}{\mu^2}$, $\rho = \lambda \bar{X}$. $\lambda = \frac{\rho}{\bar{X}} = \rho\mu$

(2) $\bar{d}_{MG1} = \bar{X} + \frac{\lambda \bar{X^2}}{2(1-\rho)} = \frac{1}{\mu} + \frac{\rho\mu \frac{2}{\mu^2}}{2(1-\rho)} = \frac{2(1-\rho)}{2(1-\rho)\mu} + \frac{2\rho}{2(1-\rho)\mu}$

$= \frac{2-2\rho+2\rho}{2(1-\rho)\mu} = \frac{1}{(1-\rho)\mu}$

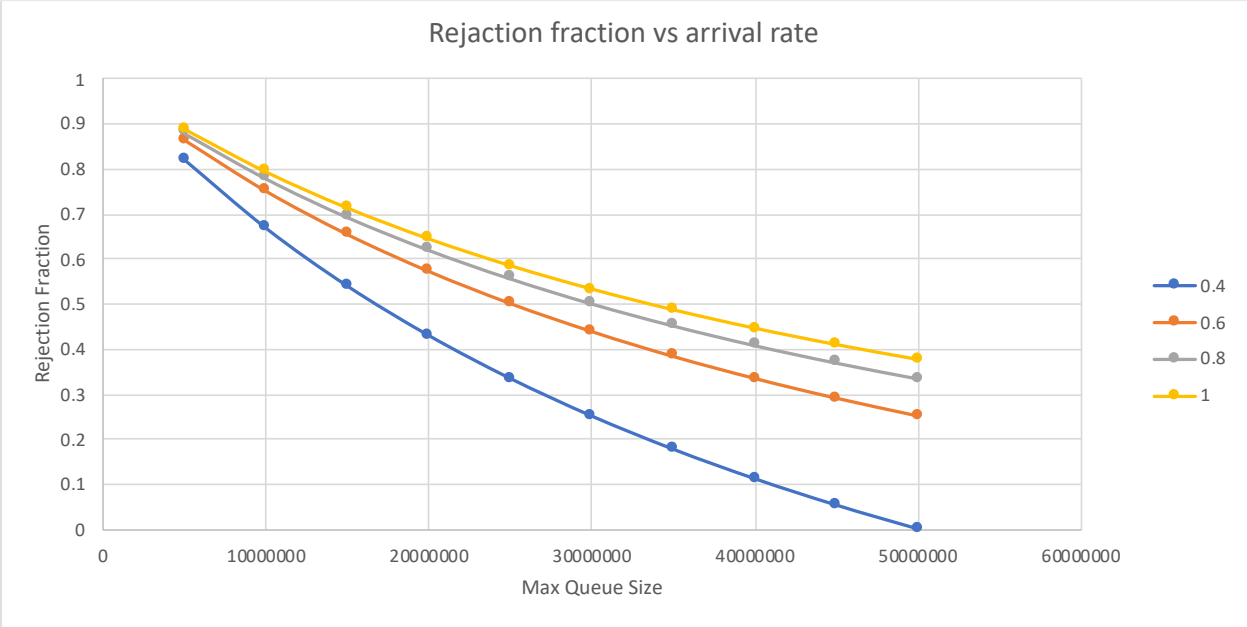
$= \frac{1}{\mu - \rho\mu} = \frac{1}{\mu - \lambda\bar{X}\cdot\frac{1}{\bar{X}}} = \frac{1}{\mu-\lambda}$

7.



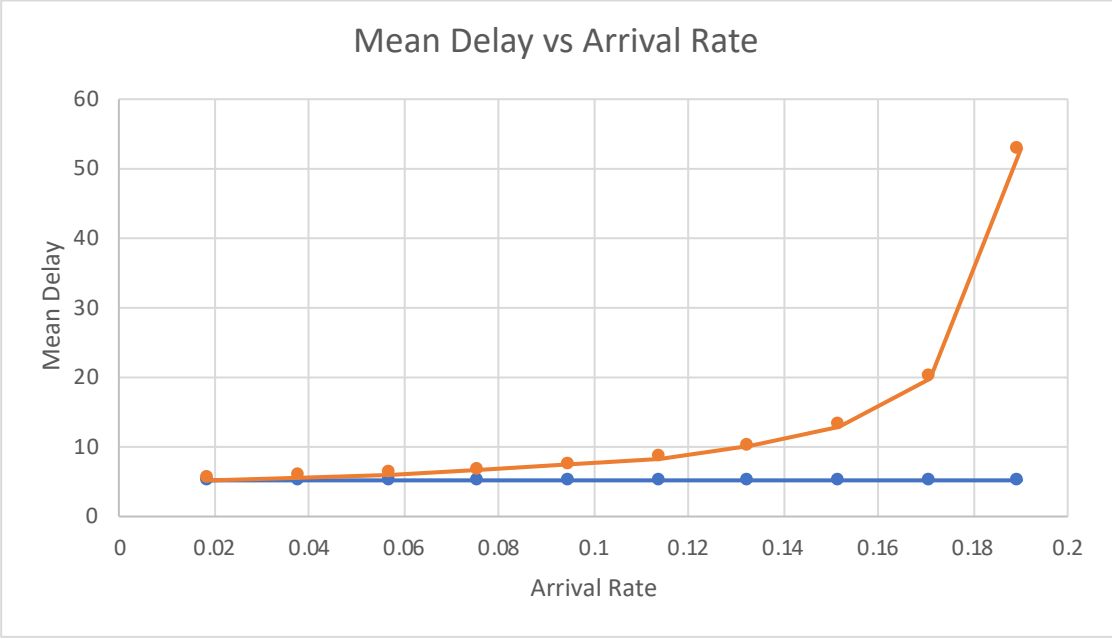
As we can find from the above simulation plot, we simulate five 5 plots of the MD1 system by limiting their MAX QUEUE SIZE. From the plot, we can see as we increase the size of the max queue, the mean delay versus arrival rate starts increasing and the mean delay will stop when reach the customers in the system is reaching the max queue size. And it is reasonable to

conclude from the plot when more customers are allowed in the system, the customer will experience a longer delay than the smaller max queue size.



The above plot is the rejected fraction vs different arrival rates under different max queue size. As we can obtained from the graph, when the max queue size of the system starts increasing, the rejection fraction will start decrease which more and more customers are allowed to enter the system. And when the max queue become infinite (the same size of the total served customer, the rejection fraction will become zero which it will allow as many as customers to come in.

8.



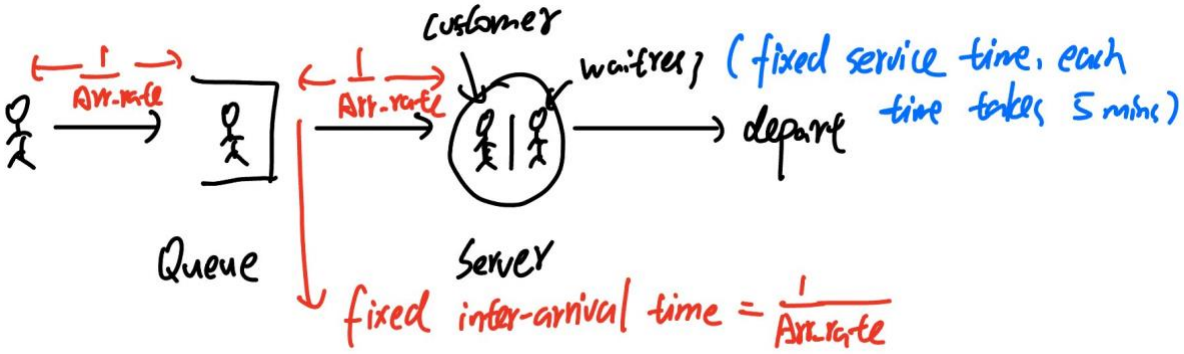
By modifying the code instead of a Poisson process arrival, now change it to a fixed time arrival with a rate of 1/Arrival rate, the above plot can be obtained where the blue line is when we change the passion process to fixed.

The reason why the shape of the curve is so different is because now we can predict the arrival process of each customer which is a fixed value so as long as we meet the condition where the products of arrival rate and service time smaller than 1, the mean delay will always be the same as the service time.

But at the same time, if the condition is not met where in this case the arrival rate is 0.201. The mean delay of customers will change and increase a lot. And it will increase more if more and more total customers being served. And a portion of customers will stay in the queue when the expected served customers is met.

```
Customers served = 10000 (Total arrived = 10051)
Utilization = 1.000000
Fraction served = 0.994926
Mean number in system = 26.000000
Mean delay = 130.000000
Hit Enter to finish ...
```

By drawing an example of why this happens, we can imagine for each customer, the waitress will take 5minutes to serve each one. And the inter-arrival time between each customer is equal to the value of 1/Arrival_rate, so to meet the condition, the max allowed arrival_rate cannot be larger than 0.2 which is exactly 5minutes inter-arrival time of each customer and utilization of the server will equal to 1 when that happens. And when the arrival_rate is smaller than 0.2, the inter-arrival time will be larger than 5, which means there is a short gap between every arrived customer. So in conclusion, the only time the customers spent before departure is the service time with the waitress which is always 5 minutes when the condition is met.



Appendix:

Exp2. Service time=5,

```
#include <stdio.h>
#include "simlib.h"
#include <time.h>
/**************************************************************************/

/*
 * Simulation Parameters
 */

// #define RANDOM_SEED 5259140
#define NUMBER_TO_SERVE 50e6

#define SERVICE_TIME 5
#define ARRIVAL_RATE 0.01

#define BLIP_RATE 10000

/**************************************************************************/

/*
 * main() uses various simulation parameters and creates a clock variable to
 * simulate real time. A loop repeatedly determines if the next event to occur
 * is a customer arrival or customer departure. In either case the state of the
 * system is updated and statistics are collected before the next
 * iteration. When it finally reaches NUMBER_TO_SERVE customers, the program
 * outputs some statistics such as mean delay.
 */

int main()
{
  //write your own code
  //
  // double arrival_rate = ARRIVAL_RATE;
  /* Set the seed of the random number generator. */
```

```
double result_1[10];
double result_2[10];


double arrival_rate[10] = {0};
double arr_increment = 0.019;
arrival_rate[0] = 0.019;


for(int j=0;j<10;j++){
 if(j!=0){
  arrival_rate[j] = arrival_rate[j-1]+arr_increment;
 }
 //for each arrival_rate value
 double random_seed[10];
 double sum = 0;
 random_seed[9] = 400318681;
 double increment = 10e6;
 for(int i=0;i<10;i++){
  //each different random_seed to make sure each simulation is independent
  if(i!=9){
   //use time Null to generate ten different number seeds
   random_seed[i] = random_seed[0] + increment;
   increment += 10e6;
  }
  printf("%f",random_seed[i]);
  random_generator_initialize(random_seed[i]);

  double clock = 0; /* Clock keeps track of simulation time. */

  /* System state variables. */
  int number_in_system = 0;
  double next_arrival_time = 0;
  double next_departure_time = 0;

  /* Data collection variables. */
  long int total_served = 0;
  long int total_arrived = 0;
```

```
double total_busy_time = 0;
double integral_of_n = 0;
double last_event_time = 0;

/* Process customers until we are finished. */
while (total_served < NUMBER_TO_SERVE) {

 /* Test if the next event is a customer arrival or departure. */
if(number_in_system == 0 || next_arrival_time < next_departure_time) {

 /*
  * A new arrival is occurring.
  */

 clock = next_arrival_time;
 next_arrival_time = clock + exponential_generator((double) 1/arrival_rate[j]);

 /* Update our statistics. */
 integral_of_n += number_in_system * (clock - last_event_time);
 last_event_time = clock;

 number_in_system++;
 total_arrived++;

 /* If this customer has arrived to an empty system, start its
service right away. */
 if(number_in_system == 1) next_departure_time = clock + SERVICE_TIME;

} else {

 /*
  * A customer departure is occuring.
  */

 clock = next_departure_time;
```

```
    /* Update our statistics. */
    integral_of_n += number_in_system * (clock - last_event_time);
    last_event_time = clock;

    number_in_system--;
    total_served++;
    total_busy_time += SERVICE_TIME;

    /*
     * If there are other customers waiting, start one in service
     * right away.
     */

    if(number_in_system > 0) next_departure_time = clock + SERVICE_TIME;

    /*
     * Every so often, print an activity message to show we are active.
     */

    if (total_served % BLIP_RATE == 0)
      printf("Customers served = %ld (Total arrived = %ld)\r",
        total_served, total_arrived);
  }

  }

  /* Output final results. */

  printf("\nUtilization = %f\n", total_busy_time/clock);
  printf("Fraction served = %f\n", (double) total_served/total_arrived);
  printf("Mean number in system = %f\n", integral_of_n/clock);
  printf("Mean delay = %f\n", integral_of_n/total_served);
  printf("Arrival rate = %f\n",arrival_rate[j]);
  sum += integral_of_n/total_served;
}

double average_mean_delay = sum/10;
```

```
 printf("The average mean delay of the arrival rate %f is %f\n",arrival_rate[j],average_mean_delay);
 result_1[j] = arrival_rate[j];
 result_2[j] = average_mean_delay;
 }
 for(int i=0;i<10;i++){
   printf("The average mean delay of the arrival rate %f is %f\n",result_1[i],result_2[i]);

 }
 return 0;
}
```

6.

```
/*
 *
 * Simulation of Single Server Queueing System
 *
 * Copyright (C) 2014 Terence D. Todd Hamilton, Ontario, CANADA,
 * todd@mcmaster.ca
 *
 * This program is free software; you can redistribute it and/or modify it under
 * the terms of the GNU General Public License as published by the Free Software
 * Foundation; either version 3 of the License, or (at your option) any later
 * version.
 *
 * This program is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
 * FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
 * details.
 *
 * You should have received a copy of the GNU General Public License along with
 * this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 */

/*************************************************************************/
```

```c
#include <stdio.h>
#include "simlib.h"

/**************************************************************************/

/*
 * Simulation Parameters
 */

#define RANDOM_SEED 400318681
#define NUMBER_TO_SERVE 50e6

#define SERVICE_TIME 5
#define ARRIVAL_RATE 0.2
#define MAX_QUEUE_SIZE 10e6
#define BLIP_RATE 10000

/**************************************************************************/

/*
 * main() uses various simulation parameters and creates a clock variable to
 * simulate real time. A loop repeatedly determines if the next event to occur
 * is a customer arrival or customer departure. In either case the state of the
 * system is updated and statistics are collected before the next
 * iteration. When it finally reaches NUMBER_TO_SERVE customers, the program
 * outputs some statistics such as mean delay.
 */

int main()
{
  double result1[50];
  double result2[50];
  double result3[50];
  int i=0;
  for(double arrival_rate = 0.2;arrival_rate<=1;arrival_rate+=0.2){
    for(long int queue_size=5e6;queue_size<=70e6;queue_size+=5e6){
```

```
double clock = 0; /* Clock keeps track of simulation time. */
/* System state variables. */
int number_in_system = 0;
double next_arrival_time = 0;
double next_departure_time = 0;

/* Data collection variables. */
long int total_served = 0;
long int total_arrived = 0;

double total_busy_time = 0;
double integral_of_n = 0;
double last_event_time = 0;

long int total_reject =0;
/* Set the seed of the random number generator. */
random_generator_initialize(RANDOM_SEED);

/* Process customers until we are finished. */
while (total_served+total_reject < NUMBER_TO_SERVE) {
        /* Test if the next event is a customer arrival or departure. */
   if(number_in_system == 0 || next_arrival_time < next_departure_time) {
     if(number_in_system >= queue_size){
       //it won't serve
       total_reject++;
       // number_in_system--;
       total_arrived++;
     }else{
       /*
        * A new arrival is occurring.
        */

       clock = next_arrival_time;
       next_arrival_time = clock + exponential_generator((double) 1/arrival_rate);

       /* Update our statistics. */
       integral_of_n += number_in_system * (clock - last_event_time);
```

```
    last_event_time = clock;

    number_in_system++;
    total_arrived++;
   }


   /* If this customer has arrived to an empty system, start its
  service right away. */
   if(number_in_system == 1) next_departure_time = clock + SERVICE_TIME;

  } else {

   /*
    * A customer departure is occuring.
    */

   clock = next_departure_time;

   /* Update our statistics. */
   integral_of_n += number_in_system * (clock - last_event_time);
   last_event_time = clock;

   number_in_system--;
   total_served++;
   total_busy_time += SERVICE_TIME;

   /*
    * If there are other customers waiting, start one in service
    * right away.
    */

   if(number_in_system > 0) next_departure_time = clock + SERVICE_TIME;

   /*
    * Every so often, print an activity message to show we are active.
    */
```

```
        if (total_served % BLIP_RATE == 0)
          printf("Customers served = %ld (Total arrived = %ld)\r",
            total_served, total_arrived);
       }
     }


   /* Output final results. */
   printf("\nArrival Rate is=%f\n",arrival_rate);
   printf("Utilization = %f\n", total_busy_time/clock);
   printf("Fraction served = %f\n", (double) total_served/total_arrived);
   printf("Mean number in system = %f\n", integral_of_n/clock);
   printf("Mean delay = %f\n", integral_of_n/total_served);
   printf("Total reject %ld\n",total_reject);
   printf("The rejection probability %f\n",(double) total_reject/total_arrived);

   result1[i] = integral_of_n/total_served;
   result2[i] = arrival_rate;
   result3[i] = (double) total_reject/total_arrived;
   i++;
  }


}

for(int i=0;i<50;i++){
  if(i%10 == 0){
   printf("The arrival rate is: balabl \n");
  }
  printf("The rejection fra is %f ,The arrival rate is %f\n",result3[i],result2[i]);



}
```

```
 return 0;

}
```

7.

```
/*
 *
 * Simulation of Single Server Queueing System
 *
 * Copyright (C) 2014 Terence D. Todd Hamilton, Ontario, CANADA,
 * todd@mcmaster.ca
 *
 * This program is free software; you can redistribute it and/or modify it under
 * the terms of the GNU General Public License as published by the Free Software
 * Foundation; either version 3 of the License, or (at your option) any later
 * version.
 *
 * This program is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
 * FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
 * details.
 *
 * You should have received a copy of the GNU General Public License along with
 * this program.  If not, see <http://www.gnu.org/licenses/>.
```

```
 *
 */

/*************************************************************************/

#include <stdio.h>
#include "simlib.h"

/*************************************************************************/

/*
 * Simulation Parameters
 */

#define RANDOM_SEED 400318681
#define NUMBER_TO_SERVE 50e6

#define SERVICE_TIME 5
#define ARRIVAL_RATE 0.2
#define MAX_QUEUE_SIZE 10e6
#define BLIP_RATE 10000

/*************************************************************************/

/*
 * main() uses various simulation parameters and creates a clock variable to
 * simulate real time. A loop repeatedly determines if the next event to occur
 * is a customer arrival or customer departure. In either case the state of the
 * system is updated and statistics are collected before the next
 * iteration. When it finally reaches NUMBER_TO_SERVE customers, the program
 * outputs some statistics such as mean delay.
 */

int main()
{
  double result1[50];
  double result2[50];
```

```
double result3[50];
int i=0;
for(long int queue_size=10e6;queue_size<=50e6;queue_size+=10e6){
 for(double arrival_rate = 0.2;arrival_rate<=2;arrival_rate+=0.2){
   double clock = 0; /* Clock keeps track of simulation time. */
   /* System state variables. */
   int number_in_system = 0;
   double next_arrival_time = 0;
   double next_departure_time = 0;

   /* Data collection variables. */
   long int total_served = 0;
   long int total_arrived = 0;

   double total_busy_time = 0;
   double integral_of_n = 0;
   double last_event_time = 0;

   long int total_reject =0;
   /* Set the seed of the random number generator. */
   random_generator_initialize(RANDOM_SEED);

   /* Process customers until we are finished. */
   while (total_served+total_reject < NUMBER_TO_SERVE) {
          /* Test if the next event is a customer arrival or departure. */
     if(number_in_system == 0 || next_arrival_time < next_departure_time) {
       if(number_in_system >= queue_size){
         //it won't serve
         total_reject++;
         // number_in_system--;
         total_arrived++;
       }else{
         /*
          * A new arrival is occurring.
          */

         clock = next_arrival_time;
```

```
    next_arrival_time = clock + exponential_generator((double) 1/arrival_rate);

    /* Update our statistics. */
    integral_of_n += number_in_system * (clock - last_event_time);
    last_event_time = clock;

    number_in_system++;
    total_arrived++;
   }


  /* If this customer has arrived to an empty system, start its
 service right away. */
  if(number_in_system == 1) next_departure_time = clock + SERVICE_TIME;

 } else {

  /*
   * A customer departure is occuring.
   */

  clock = next_departure_time;

  /* Update our statistics. */
  integral_of_n += number_in_system * (clock - last_event_time);
  last_event_time = clock;

  number_in_system--;
  total_served++;
  total_busy_time += SERVICE_TIME;

  /*
   * If there are other customers waiting, start one in service
   * right away.
   */

  if(number_in_system > 0) next_departure_time = clock + SERVICE_TIME;
```

```
        /*
         * Every so often, print an activity message to show we are active.
         */

        if (total_served % BLIP_RATE == 0)
          printf("Customers served = %ld (Total arrived = %ld)\r",
            total_served, total_arrived);
       }
      }


    /* Output final results. */
    printf("\nArrival Rate is=%f\n",arrival_rate);
    printf("Utilization = %f\n", total_busy_time/clock);
    printf("Fraction served = %f\n", (double) total_served/total_arrived);
    printf("Mean number in system = %f\n", integral_of_n/clock);
    printf("Mean delay = %f\n", integral_of_n/total_served);
    printf("Total reject %ld\n",total_reject);
    printf("The rejection probability %f\n",(double) total_reject/total_arrived);

    result1[i] = integral_of_n/total_served;
    result2[i] = arrival_rate;
    result3[i] = (double) total_reject/total_arrived;
    i++;
   }


 }

 for(int i=0;i<50;i++){
   if(i%10 == 0){
    printf("The max queue size is: balabl \n");
   }
   printf("The rejection fra is %f ,The arrival rate is %f and the mean delay
is %f\n",result3[i],result2[i],result1[i]);
```

```
 }


 return 0;
}
```