|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Mean interarrival time ( minutes) | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| Mean service time minutes | 0.300 | 0.400 | 0.500 | 0.600 | 0.700 |
| Mean time an item spends in the system | 0.3/1-0.3 = 0.429 | 0.4/1-0.4=0.667 | 0.5/1-0.5=1 | 1.5 | 2.333 |
| Mean no of items waiting to be served | 0.3\*0.3/1-0.3=0.129 | 0.4\*0.4/1-0.4=0.267 | 0.5\*0.5/1-0.5=0.5 | 0.9 | 1.633 |
| Mean waiting time (includes items that have to wait and items with waiting time=0) | 0.3\*0.3/1-0.3=0.129 | 0.267 | 0.5\*0.5/1-0.5=0.5 | 0.9 | 1.633 |
| Average delay in queue | 0.129 | 0.267 | 0.500 | 0.900 | 1.633 |
| Average number in queue | 0.129 | 0.267 | 0.500 | 0.900 | 1.633 |

**Assignment #2**

Analytical results: for M/M/1 for n=20000 customers

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Mean interarrival time ( minutes) | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| Mean service time minutes | 0.300 | 0.400 | 0.500 | 0.600 | 0.700 |
| Mean time an item spends in the system | 0.3/1-0.3 = 0.429 | 0.4/1-0.4=0.667 | 0.5/1-0.5=1 | 1.5 | 2.333 |
| Mean no of items waiting to be served | 0.3\*0.3/1-0.3=0.129 | 0.4\*0.4/1-0.4=0.267 | 0.5\*0.5/1-0.5=0.5 | 0.9 | 1.633 |
| Mean waiting time (includes items that have to wait and items with waiting time=0) | 0.3\*0.3/1-0.3=0.129 | 0.267 | 0.5\*0.5/1-0.5=0.5 | 0.9 | 1.633 |
| Average delay in queue | 0.129 | 0.260 | 0.475 | 0.833 | 1.443 |
| Average number in queue | 0.129 | 0.260 | 0.472 | 0.832 | 1.428 |

Simulation results:

**C program for M/M/1:**

/\* External definitions for single-server queueing system. \*/

#include <stdio.h>

#include <math.h>

#include<stdlib.h>

#include "lcgrand.h"

/\* #include "lcgrand.h" /\* Header file for random-number generator. \*/

#define Q\_LIMIT 100 /\* Limit on queue length. \*/

#define BUSY 1 /\* Mnemonics for server's being busy \*/

#define IDLE 0 /\* and idle. \*/

int next\_event\_type, num\_custs\_delayed, num\_delays\_required, num\_events,

num\_in\_q, server\_status;

float area\_num\_in\_q, area\_server\_status, mean\_interarrival, mean\_service,

sim\_time, time\_arrival[Q\_LIMIT + 1], time\_last\_event,

time\_next\_event[3],

total\_of\_delays;

FILE \*infile, \*outfile;

void initialize(void);

void timing(void);

void arrive(void);

void depart(void);

void report(void);

void update\_time\_avg\_stats(void);

float expon(float mean);

int main() /\* Main function. \*/

{

/\* Open input and output files. \*/

infile = fopen("mm1.in", "r");

outfile = fopen("mm1.out", "w");

/\* Specify the number of events for the timing function. \*/

num\_events = 2;

/\* Read input parameters. \*/

fscanf(infile, "%f %f %d", &mean\_interarrival, &mean\_service,

&num\_delays\_required);

/\* Write report heading and input parameters. \*/

fprintf(outfile, "Single-server queueing system\n\n");

fprintf(outfile, "Mean interarrival time%11.3f minutes\n\n",

mean\_interarrival);

fprintf(outfile, "Mean service time%16.3f minutes\n\n", mean\_service);

fprintf(outfile, "Number of customers%14d\n\n", num\_delays\_required);

/\* Initialize the simulation. \*/

initialize();

/\* Run the simulation while more delays are still needed. \*/

while (num\_custs\_delayed < num\_delays\_required) {

/\* Determine the next event. \*/

timing();

/\* Update time-average statistical accumulators. \*/

update\_time\_avg\_stats();

/\* Invoke the appropriate event function. \*/

switch (next\_event\_type) {

case 1:

arrive();

break;

case 2:

depart();

break;

}

}

/\* Invoke the report generator and end the simulation. \*/

report();

fclose(infile);

fclose(outfile);

return 0;

}

void initialize(void) /\* Initialization function. \*/

{

/\* Initialize the simulation clock. \*/

sim\_time = 0.0;

/\* Initialize the state variables. \*/

server\_status = IDLE;

num\_in\_q = 0;

time\_last\_event = 0.0;

/\* Initialize the statistical counters. \*/

num\_custs\_delayed = 0;

total\_of\_delays = 0.0;

area\_num\_in\_q = 0.0;

area\_server\_status = 0.0;

/\* Initialize event list. Since no customers are present, the

departure

(service completion) event is eliminated from consideration. \*/

time\_next\_event[1] = sim\_time + expon(mean\_interarrival);

time\_next\_event[2] = 1.0e+30;

}

void timing(void) /\* Timing function. \*/

{

int i;

float min\_time\_next\_event = 1.0e+29;

next\_event\_type = 0;

/\* Determine the event type of the next event to occur. \*/

for (i = 1; i <= num\_events; ++i){

if (time\_next\_event[i] < min\_time\_next\_event) {

min\_time\_next\_event = time\_next\_event[i];

next\_event\_type = i;

}

}

/\* Check to see whether the event list is empty. \*/

if (next\_event\_type == 0) {

/\* The event list is empty, so stop the simulation. \*/

fprintf(outfile, "\nEvent list empty at time %f", sim\_time);

exit(1);

}

/\* The event list is not empty, so advance the simulation clock. \*/

sim\_time = min\_time\_next\_event;

}

void arrive(void) /\* Arrival event function. \*/

{

float delay;

/\* Schedule next arrival. \*/

time\_next\_event[1] = sim\_time + expon( mean\_interarrival);

/\* Check to see whether server is busy. \*/

if (server\_status == BUSY) {

/\* Server is busy, so increment number of customers in queue. \*/

++num\_in\_q;

/\* Check to see whether an overflow condition exists. \*/

if (num\_in\_q > Q\_LIMIT) {

/\* The queue has overflowed, so stop the simulation. \*/

fprintf(outfile, "\nOverflow of the array time\_arrival at");

fprintf(outfile, " time %f", sim\_time);

exit(2);

}

/\* There is still room in the queue, so store the time of arrival

of the

arriving customer at the (new) end of time\_arrival. \*/

time\_arrival[num\_in\_q] = sim\_time;

}

else {

/\* Server is idle, so arriving customer has a delay of zero. (The

following two statements are for program clarity and do not

affect

the results of the simulation.) \*/

delay = 0.0;

total\_of\_delays += delay;

/\* Increment the number of customers delayed, and make server

busy. \*/

++num\_custs\_delayed;

server\_status = BUSY;

/\* Schedule a departure (service completion). \*/

time\_next\_event[2] = sim\_time + expon(mean\_service);

}

}

void depart(void) /\* Departure event function. \*/

{

int i;

float delay;

/\* Check to see whether the queue is empty. \*/

if (num\_in\_q == 0) {

/\* The queue is empty so make the server idle and eliminate the

departure (service completion) event from consideration. \*/

server\_status = IDLE;

time\_next\_event[2] = 1.0e+30;

}

else {

/\* The queue is nonempty, so decrement the number of customers in

queue. \*/

--num\_in\_q;

/\* Compute the delay of the customer who is beginning service and

update

the total delay accumulator. \*/

delay = sim\_time - time\_arrival[1];

total\_of\_delays += delay;

/\* Increment the number of customers delayed, and schedule

departure. \*/

++num\_custs\_delayed;

time\_next\_event[2] = sim\_time + expon(mean\_service);

/\* Move each customer in queue (if any) up one place. \*/

for (i = 1; i <= num\_in\_q; ++i)

time\_arrival[i] = time\_arrival[i + 1];

}

}

void report(void) /\* Report generator function. \*/

{

/\* Compute and write estimates of desired measures of performance. \*/

fprintf(outfile, "\n\nAverage delay in queue%11.3f minutes\n\n",

total\_of\_delays / num\_custs\_delayed);

fprintf(outfile, "Average number in queue%10.3f\n\n",

area\_num\_in\_q / sim\_time);

fprintf(outfile, "Server utilization%15.3f\n\n",

area\_server\_status / sim\_time);

fprintf(outfile, "Time simulation ended%12.3f minutes", sim\_time);

}

void update\_time\_avg\_stats(void) /\* Update area accumulators for timeaverage

statistics. \*/

{

float time\_since\_last\_event;

/\* Compute time since last event, and update last-event-time marker.

\*/

time\_since\_last\_event = sim\_time - time\_last\_event;

time\_last\_event = sim\_time;

/\* Update area under number-in-queue function. \*/

area\_num\_in\_q += num\_in\_q \* time\_since\_last\_event;

/\* Update area under server-busy indicator function. \*/

area\_server\_status += server\_status \* time\_since\_last\_event;

}

float expon(float mean) /\* Exponential variate generation function. \*/

{

/\* Return an exponential random variate with mean "mean". \*/

return -mean \* logf(lcgrand(1));

}

/\* End of Code \*/

Plot using MATLAB:

x = [0.3,0.4,0.5,0.6,0.7];

y1 = [0.129,0.260,0.475,0.833,1.443];

y2 = [0.129,0.267,0.5,0.9,1.633];

plot(x,y1,x,y2)

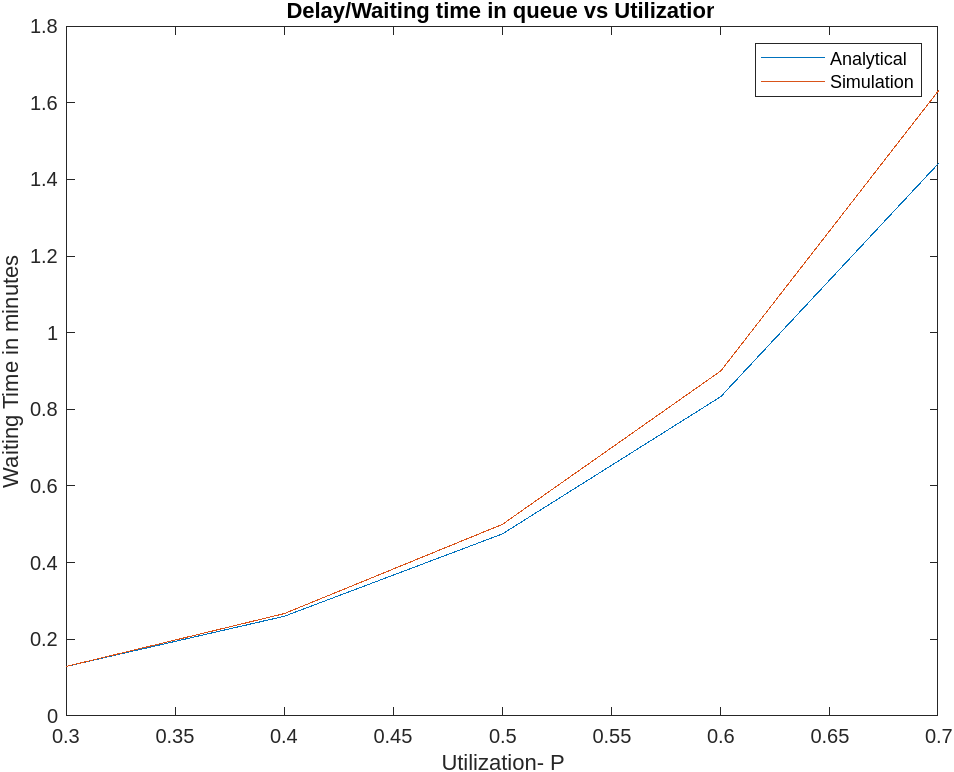
legend({'Analytical','Simulation'},'Location','northeast')

title('Delay/Waiting time in queue vs Utilization')

xlabel('Utilization- P')

ylabel('Waiting Time in minutes')

Figure:



The above figure shows the results for the measures both analytic and from simulation as a function of ρ(utilization) = λ/μ where λ is arrival rate(= inverse of mean interarrival time) and μ is service rate (inverse of mean service time)

\*\*\*

**Assignment #3**

Analytical results: for M/D/1, n=20000 customers

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Mean interarrival time ( minutes) | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| Mean service time minutes | 0.300 | 0.400 | 0.500 | 0.600 | 0.700 |
| Mean time an item spends in the system | 0.3\*(2-0.3) / 2\*(1-0.3) = 0.364 | 0.4\*(2-0.4) /2\*(1-0.4) = 0.533 | 0.5\*(2-0.5) /2(1-0.5) = 0.75 | 0.6\*(2-0.6) / 2(1-0.6) = 1.05 | 0.7\*(2-0.7) /2(1-0.7) = 1.517 |
| Mean no of items waiting to be served | 0.3\*0.3/2(1-0.3) =0.064 | 0.4\*0.4/2(1-0.4) = 0.133 | 0.5\*0.5/2(1-0.5) = 0.25 | 0.45 | 1.817 |
| Mean waiting time (includes items that have to wait and items with waiting time=0) | 0.3\*0.3/2(1-0.3) =0.064 | 0.133 | 0.5\*0.5/2(1-0.5) = 0.25 | 0.45 | 1.817 |
| Average delay in queue | 0.064 | 0.133 | 0.250 | 0.450 | 1.817 |
| Average number in queue | 0.064 | 0.133 | 0.250 | 0.450 | 1.817 |

Simulation results:

For M/D/1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Mean interarrival time ( minutes) | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| Mean service time minutes | 0.300 | 0.400 | 0.500 | 0.600 | 0.700 |
| Mean time an item spends in the system | 0.3\*(2-0.3) / 2\*(1-0.3) = 0.364 | 0.4\*(2-0.4) /2\*(1-0.4) = 0.533 | 0.5\*(2-0.5) /2(1-0.5) = 0.75 | 0.6\*(2-0.6) / 2(1-0.6) = 1.05 | 0.7\*(2-0.7) /2(1-0.7) = 1.517 |
| Mean no of items waiting to be served | 0.3\*0.3/2(1-0.3) =0.064 | 0.4\*0.4/2(1-0.4) = 0.133 | 0.5\*0.5/2(1-0.5) = 0.25 | 0.45 | 0.817 |
| Mean waiting time (includes items that have to wait and items with waiting time=0) | 0.3\*0.3/2(1-0.3) =0.064 | 0.133 | 0.5\*0.5/2(1-0.5) = 0.25 | 0.45 | 0.817 |
| Average delay in queue | 0.064 | 0.132 | 0.245 | 0.440 | 0.806 |
| Average number in queue | 0.065 | 0.133 | 0.247 | 0.444 | 0.812 |

**C Program for M/D/1:**

/\* External definitions for single-server queueing system. \*/

#include <stdio.h>

#include <math.h>

#include<stdlib.h>

#include "lcgrand.h"

/\* #include "lcgrand.h" /\* Header file for random-number generator. \*/

#define Q\_LIMIT 100 /\* Limit on queue length. \*/

#define BUSY 1 /\* Mnemonics for server's being busy \*/

#define IDLE 0 /\* and idle. \*/

int next\_event\_type, num\_custs\_delayed, num\_delays\_required, num\_events,

num\_in\_q, server\_status;

float area\_num\_in\_q, area\_server\_status, mean\_interarrival, mean\_service,

sim\_time, time\_arrival[Q\_LIMIT + 1], time\_last\_event,

time\_next\_event[3],

total\_of\_delays;

FILE \*infile, \*outfile;

void initialize(void);

void timing(void);

void arrive(void);

void depart(void);

void report(void);

void update\_time\_avg\_stats(void);

float expon(float mean);

int main() /\* Main function. \*/

{

/\* Open input and output files. \*/

infile = fopen("md1.in", "r");

outfile = fopen("md1.out", "w");

/\* Specify the number of events for the timing function. \*/

num\_events = 2;

/\* Read input parameters. \*/

fscanf(infile, "%f %f %d", &mean\_interarrival, &mean\_service,

&num\_delays\_required);

/\* Write report heading and input parameters. \*/

fprintf(outfile, "Single-server queueing system\n\n");

fprintf(outfile, "Mean interarrival time%11.3f minutes\n\n",

mean\_interarrival);

fprintf(outfile, "Mean service time%16.3f minutes\n\n", mean\_service);

fprintf(outfile, "Number of customers%14d\n\n", num\_delays\_required);

/\* Initialize the simulation. \*/

initialize();

/\* Run the simulation while more delays are still needed. \*/

while (num\_custs\_delayed < num\_delays\_required) {

/\* Determine the next event. \*/

timing();

/\* Update time-average statistical accumulators. \*/

update\_time\_avg\_stats();

/\* Invoke the appropriate event function. \*/

switch (next\_event\_type) {

case 1:

arrive();

break;

case 2:

depart();

break;

}

}

/\* Invoke the report generator and end the simulation. \*/

report();

fclose(infile);

fclose(outfile);

return 0;

}

void initialize(void) /\* Initialization function. \*/

{

/\* Initialize the simulation clock. \*/

sim\_time = 0.0;

/\* Initialize the state variables. \*/

server\_status = IDLE;

num\_in\_q = 0;

time\_last\_event = 0.0;

/\* Initialize the statistical counters. \*/

num\_custs\_delayed = 0;

total\_of\_delays = 0.0;

area\_num\_in\_q = 0.0;

area\_server\_status = 0.0;

/\* Initialize event list. Since no customers are present, the

departure

(service completion) event is eliminated from consideration. \*/

time\_next\_event[1] = sim\_time + expon(mean\_interarrival);

time\_next\_event[2] = 1.0e+30;

}

void timing(void) /\* Timing function. \*/

{

int i;

float min\_time\_next\_event = 1.0e+29;

next\_event\_type = 0;

/\* Determine the event type of the next event to occur. \*/

for (i = 1; i <= num\_events; ++i){

if (time\_next\_event[i] < min\_time\_next\_event) {

min\_time\_next\_event = time\_next\_event[i];

next\_event\_type = i;

}

}

/\* Check to see whether the event list is empty. \*/

if (next\_event\_type == 0) {

/\* The event list is empty, so stop the simulation. \*/

fprintf(outfile, "\nEvent list empty at time %f", sim\_time);

exit(1);

}

/\* The event list is not empty, so advance the simulation clock. \*/

sim\_time = min\_time\_next\_event;

}

void arrive(void) /\* Arrival event function. \*/

{

float delay;

/\* Schedule next arrival. \*/

time\_next\_event[1] = sim\_time + expon( mean\_interarrival);

/\* Check to see whether server is busy. \*/

if (server\_status == BUSY) {

/\* Server is busy, so increment number of customers in queue. \*/

++num\_in\_q;

/\* Check to see whether an overflow condition exists. \*/

if (num\_in\_q > Q\_LIMIT) {

/\* The queue has overflowed, so stop the simulation. \*/

fprintf(outfile, "\nOverflow of the array time\_arrival at");

fprintf(outfile, " time %f", sim\_time);

exit(2);

}

/\* There is still room in the queue, so store the time of arrival

of the

arriving customer at the (new) end of time\_arrival. \*/

time\_arrival[num\_in\_q] = sim\_time;

}

else {

/\* Server is idle, so arriving customer has a delay of zero. (The

following two statements are for program clarity and do not

affect

the results of the simulation.) \*/

delay = 0.0;

total\_of\_delays += delay;

/\* Increment the number of customers delayed, and make server

busy. \*/

++num\_custs\_delayed;

server\_status = BUSY;

/\* Schedule a departure (service completion). \*/

time\_next\_event[2] = sim\_time + mean\_service; /\*changes here\*/

}

}

void depart(void) /\* Departure event function. \*/

{

int i;

float delay;

/\* Check to see whether the queue is empty. \*/

if (num\_in\_q == 0) {

/\* The queue is empty so make the server idle and eliminate the

departure (service completion) event from consideration. \*/

server\_status = IDLE;

time\_next\_event[2] = 1.0e+30;

}

else {

/\* The queue is nonempty, so decrement the number of customers in

queue. \*/

--num\_in\_q;

/\* Compute the delay of the customer who is beginning service and

update

the total delay accumulator. \*/

delay = sim\_time - time\_arrival[1];

total\_of\_delays += delay;

/\* Increment the number of customers delayed, and schedule

departure. \*/

++num\_custs\_delayed;

time\_next\_event[2] = sim\_time + mean\_service; /\* changes here \*/

/\* Move each customer in queue (if any) up one place. \*/

for (i = 1; i <= num\_in\_q; ++i)

time\_arrival[i] = time\_arrival[i + 1];

}

}

void report(void) /\* Report generator function. \*/

{

/\* Compute and write estimates of desired measures of performance. \*/

fprintf(outfile, "\n\nAverage delay in queue%11.3f minutes\n\n",

total\_of\_delays / num\_custs\_delayed);

fprintf(outfile, "Average number in queue%10.3f\n\n",

area\_num\_in\_q / sim\_time);

fprintf(outfile, "Server utilization%15.3f\n\n",

area\_server\_status / sim\_time);

fprintf(outfile, "Time simulation ended%12.3f minutes", sim\_time);

}

void update\_time\_avg\_stats(void) /\* Update area accumulators for timeaverage

statistics. \*/

{

float time\_since\_last\_event;

/\* Compute time since last event, and update last-event-time marker.

\*/

time\_since\_last\_event = sim\_time - time\_last\_event;

time\_last\_event = sim\_time;

/\* Update area under number-in-queue function. \*/

area\_num\_in\_q += num\_in\_q \* time\_since\_last\_event;

/\* Update area under server-busy indicator function. \*/

area\_server\_status += server\_status \* time\_since\_last\_event;

}

float expon(float mean) /\* Exponential variate generation function. \*/

{

/\* Return an exponential random variate with mean "mean". \*/

return -mean \* logf(lcgrand(1));

}

/\* End of Code \*/

Plot using MATLAB:

x = [0.3,0.4,0.5,0.6,0.7];

y1 = [0.064,0.133,0.25,0.45,0.817];

y2 = [0.064,0.132,0.245,0.44,0.806];

plot(x,y1,x,y2)

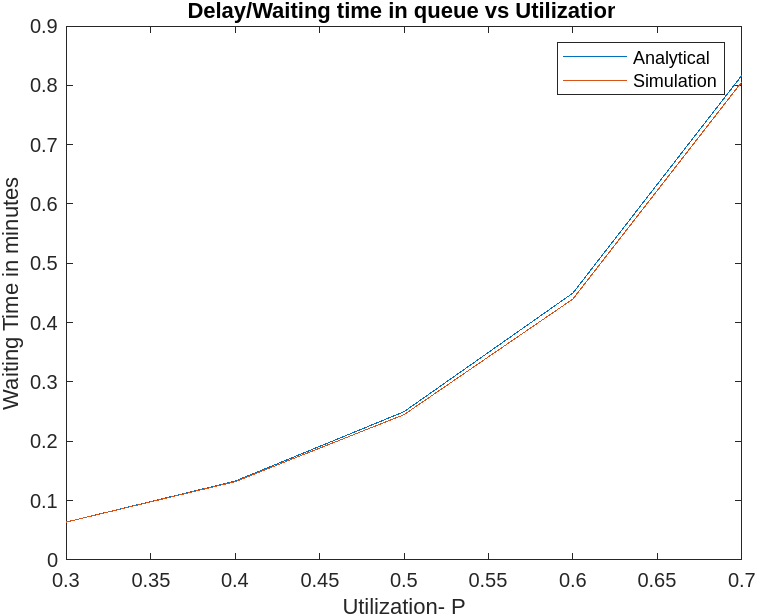
legend({'Analytical','Simulation'},'Location','northeast')

title('Delay/Waiting time in queue vs Utilization')

xlabel('Utilization- P')

ylabel('Waiting Time in minutes')

Figure:



The above figure shows the results for the measures both analytic and from simulation as a function of ρ(utilization) = λ/μ where λ is arrival rate(= inverse of mean interarrival time) and μ is service rate (inverse of mean service time)

\*\*\*