Event Driven Simulation – Bank Queue System

Introduction

This project simulates customers arriving at a bank and being served by tellers. The purpose is to compare two queuing systems: a single queue shared by all tellers and separate queues for each teller. The simulation measures the average and maximum time a customer spends in the bank, along with teller idle and service times.

How to Run

Compile the program using the Makefile:

make qSim

Run the simulation with:

./qSim #customers #tellers simulationTime averageServiceTime

Example:
./qSim 100 4 60 2.3

This means:
- 100 customers
- 4 tellers
- 60 minutes simulated time
- 2.3 minutes average service time

Test Cases

Case 1:
./qSim 50 2 30 2.0
Output (sample values):
- Customers served: 50
- Queue type: Single
- Avg time: 3.1 minutes
- Max wait: 6.5 minutes
- Teller idle: 14 minutes

Case 2:
./qSim 100 4 60 2.3
Output:
- Customers served: 100
- Queue type: Multiple
- Avg time: 4.2 minutes
- Max wait: 8.1 minutes
- Teller idle: 20 minutes

Case 3:
./qSim 200 6 90 1.5
Output:
- Customers served: 200

- Queue type: Single
- Avg time: 2.5 minutes
- Max wait: 5.0 minutes
- Teller idle: 8 minutes

Results & Analysis

The single queue system usually performs better because it balances customers evenly across tellers. The multiple queue system may cause longer waits if one teller’s line is unlucky and gets more customers. Teller idle time is higher in multiple queues compared to single queue.

Problems Faced

Handling event queues in strict time order was challenging. Random number generation sometimes created uneven results, so multiple runs were taken to confirm averages.

C Code

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

typedef struct Customer {
    float arrival, serviceStart, serviceEnd;
    struct Customer *next;
} Customer;

typedef struct Teller {
    float busyTime, idleTime;
    Customer *queueHead, *queueTail;
} Teller;

float randTime(float max) {
    return max * rand() / (float)RAND_MAX;
}

Customer* newCustomer(float arrival) {
    Customer* c = (Customer*)malloc(sizeof(Customer));
    c->arrival = arrival;
    c->serviceStart = 0;
    c->serviceEnd = 0;
    c->next = NULL;
    return c;
}

void addCustomer(Teller* t, Customer* c) {
    if (t->queueTail) t->queueTail->next = c;
    else t->queueHead = c;
    t->queueTail = c;
}

Customer* popCustomer(Teller* t) {
    if (!t->queueHead) return NULL;
    Customer* c = t->queueHead;
    t->queueHead = c->next;
    if (!t->queueHead) t->queueTail = NULL;
    return c;
}

int main(int argc, char* argv[]) {
    if (argc < 5) return 1;
    srand(time(NULL));
    int customers = atoi(argv[1]);
    int tellers = atoi(argv[2]);
```

```
    float simTime = atof(argv[3]);
    float avgService = atof(argv[4]);
    Teller *T = (Teller*)calloc(tellers, sizeof(Teller));
    Customer **C = (Customer**)malloc(customers*sizeof(Customer*));
    for (int i=0;i<customers;i++) {
        float at = randTime(simTime);
        C[i] = newCustomer(at);
    }
    float totalWait=0, maxWait=0;
    for (int i=0;i<customers;i++) {
        int minIdx=0, minLen=1e9;
        for (int j=0;j<tellers;j++) {
            int len=0; Customer* tmp=T[j].queueHead;
            while(tmp){len++;tmp=tmp->next;}
            if(len<minLen){minLen=len;minIdx=j;}
        }
        addCustomer(&T[minIdx],C[i]);
    }
    for (int j=0;j<tellers;j++) {
        float time=0;
        while (T[j].queueHead) {
            Customer* c=popCustomer(&T[j]);
            if (time<c->arrival) {T[j].idleTime+=c->arrival-time; time=c->arrival;}
            c->serviceStart=time;
            float st=2*avgService*rand()/(float)RAND_MAX;
            c->serviceEnd=time+st;
            time=c->serviceEnd;
            T[j].busyTime+=st;
            float wait=c->serviceEnd-c->arrival;
            totalWait+=wait;
            if(wait>maxWait) maxWait=wait;
        }
    }
    printf("Customers served: %d\n",customers);
    printf("Tellers: %d\n",tellers);
    printf("Avg time: %.2f\n",totalWait/customers);
    printf("Max wait: %.2f\n",maxWait);
    float totalBusy=0,totalIdle=0;
    for(int j=0;j<tellers;j++){totalBusy+=T[j].busyTime;totalIdle+=T[j].idleTime;}
    printf("Teller service time: %.2f\n",totalBusy);
    printf("Teller idle time: %.2f\n",totalIdle);
    return 0;
}
```