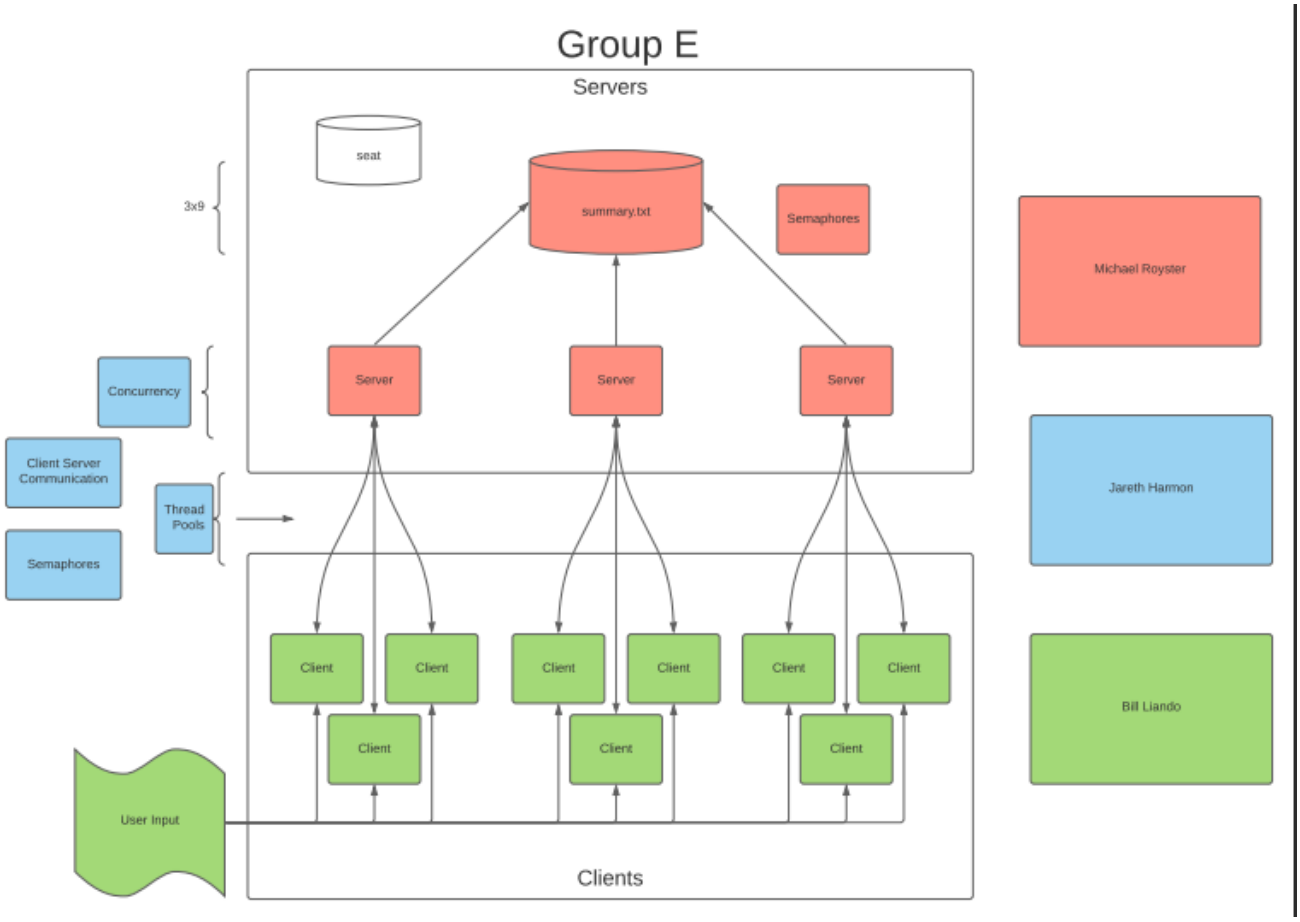
Group E Final Report

Michael Royster | Jareth Harmon | Bill Liando



Michael Royster

Job function:

Handle the Server operations for the menu and manage semaphores for critical sections dealing with reading and writing.

Server operations for the following have been implemented:

1. Make a reservation
2. Inquiry about the ticket
3. Modify the reservation
4. Cancel the reservation

All backend functions work to maintain and query the summary.txt file. Many helper functions were created to assist the server with checking information in the summary.txt file. When necessary

semaphores are added to solve the reader/writer problem. Many processes/threads can read at the same time, but only one can write

Code:

Makefile:

```
main:
    gcc -c -o server.o Server.c -pthread -lrt
    gcc -c -o backend.o Backend.c -pthread -lrt
    gcc -c -o main.o Main.c
    gcc -o main main.o server.o backend.o -pthread -lrt

client:
    gcc -c client Client.c

clean:
    rm *.o

remove:
    rm /dev/shm/sem.file_write
    rm /dev/shm/sem.file_read

test:
    gcc -c -o backend.o Backend.c -pthread -lrt
    gcc -c -o test.o Test.c -pthread -lrt
    gcc -o test test.o backend.o -pthread -lrt
```

Backend.h

```
// Author: Michael Royster
#ifndef BACKEND_H
#define BACKEND_H

#include "Reservation.h"

void    init_sync(sem_t *file_write, sem_t *file_read, int shm_fd, int
*ptrReaders);
```

```
void    desync(sem_t *file_write, sem_t *file_read, int shm_fd, int
*ptrReaders);
void    ServerX(char name);
void    get_date(char* date, char* tomorrow);
int     file_exists(char *filename);
void    make_reservation(char server, Reservation* reservation, int
numberTravelers);
void    add_travelers(char server, Reservation* reservation, int
numberNewTravelers, char* ticket);
void    remove_traveler(char* ticket, char* name);
void    inquiry(char* ticket, Reservation* info, int* numTravelers);
void    update_train_seats(char* ticket,char* name, char* seat, char server);
// modify
void    cancel_reservation(char* ticket);
//void    available_seats(int date, char* options, int* numberAvailable);
void    available_seats(char *date, char* options, int* numberAvailable);
void    receipt(Reservation *reservations, int numberTravelers, char server,
char* receipt);
//int     check_seat(int date, char *seat);
int     check_seat(char *date, char *seat);
void    testX(char name);
void    get_num_travelers(char* ticket, int* numTravelers);
void    get_travel_date(char* ticket, char* travelDate);

#endif
```

Reservation.h:

```
#ifndef RESERVATION_H
#define RESERVATION_H

// make a struct for Reservation

typedef struct Reservation{
    char customerName[64];
    char dob[32];
    char gender[12];
    int govID;
    char travelDate[32];
```

```
    int numberTravelers;
    char seat[4];
} Reservation;

#endif
```

Backend.c:

```
// Author: Michael Royster
// Email: micaher@okstate.edu

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <dirent.h>
#include <semaphore.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "Reservation.h"
#include "Backend.h"

#define MAX_SEATS 27
#define BUFFER_SIZE 2048

// Initialize semaphores and shared memory space
void init_sync(sem_t *file_write, sem_t *file_read, int shm_fd, int
*ptrReaders){
    // Create semaphores
    file_write = sem_open("/file_write", O_CREAT, 0666, 1);
    file_read = sem_open("/file_read", O_CREAT, 0666, 1);

    // Create shared memory object
    shm_fd = shm_open("readers", O_CREAT | O_RDWR, 0666);
    ftruncate(shm_fd, sizeof(int));
    ptrReaders = mmap(0, sizeof(int), PROT_WRITE, MAP_SHARED, shm_fd, 0);
```

```
    *ptrReaders = 0;
}

// Clean up semaphores and shared memory space
void desync(sem_t *file_write, sem_t *file_read, int shm_fd, int *ptrReaders){
    // unmap, close and delete shared memory object
    munmap(ptrReaders, sizeof(int));
    close(shm_fd);
    shm_unlink("readers");

    // Unlink and close semaphores
    sem_unlink("/file_write");
    sem_close(file_write);
    sem_unlink("/file_read");
    sem_close(file_read);
}

// Testing purposes only
void ServerX(char name){
    char server_name = name; // this should later be a parameter of Server(char
name)

    // Reservation *info = (Reservation*)malloc(sizeof(Reservation)*27);
    // inquiry("4172021-2", info);
    // printf("%c\n", server_name);
    // for (int i =0 ;i < 3; i++){
    //     printf("%s\n",(info+i)->customerName);
    // }
    // free(info);

    Reservation reservation1 = {"Katy", "1-1-1900", "M", 12345, "4172021", 2,
"D2"};
    Reservation reservation2 = {"Joey", "1-1-2122", "M", 45768, "4772021", 2,
"A3"};

    // Reservation reservation3 = {"Steve", "1-1-2122", "M", 45768, "4172021",
2, "A3"};
```

```
    // Reservation reservation4 = {"Johny", "1-1-2122", "M", 45768, "4172021",
2, "A4"};
    Reservation *reservations = (Reservation*)malloc(sizeof(Reservation) * 4);
    // *reservations = reservation3;
    // *(reservations+1) = reservation4;

    // Reservation *reservations = (Reservation*)malloc(sizeof(Reservation) *
2);
    *reservations = reservation1;
    *(reservations+1) = reservation2;
    // add_travelers(server_name, reservations, 2, "4172021-1");
    // inquiry("4172021-1", reservations);
    // printf("1: %s \n2: %s\n3: %s\n4: %s\n", reservations->customerName,
(reservations+1)->customerName, (reservations+2)->customerName,
(reservations+3)->customerName);
    // make_reservation(server_name, reservations, 2);
    // update_train_seats("4172021-3", "Sean", "B3", server_name);
    // remove_traveler("4172021-1", "Joey");
    // cancel_reservation("4172021-4");
    // char arr[112];
    // available_seats(4172021, arr);
    // printf("%s\n", arr);
    // check_seat(4172021, "A3");
    // receipt(reservations, 2, name);
}

// Get date for today and tomorrow
void get_date(char* today, char* tomorrow){
    time_t t = time(NULL);
    struct tm tm = *localtime(&t);
    snprintf(today, sizeof(char)*10, "%d-%d-%d", tm.tm_year + 1900, tm.tm_mon +
1, tm.tm_mday);
    snprintf(tomorrow, sizeof(char)*10, "%d-%d-%d", tm.tm_year + 1900,
tm.tm_mon + 1, tm.tm_mday+1);
}

// return 1 if filename is in directory
int file_exists(char *filename){
```

```
    DIR *d;
    struct dirent *dir;
    d = opendir(".");
    if (d){
        while ((dir =readdir(d)) != NULL){
            if (strcmp(filename, dir->d_name) == 0){
                closedir(d);
                return 1;
            }
        }
    }
    closedir(d);
    return 0;
}

// Has a critical section - WRITE, threadsafe X
// Updated 4/23, Now takes an array of passengers and the number of passengers
// Ticket number is based on the travelDate of the first reservation
void make_reservation(char server, Reservation* reservation, int
numberTravelers){
    //time_t t = time(NULL);
    //struct tm tm = *localtime(&t);
   // char today[9];
   // snprintf(today, sizeof(char) * 8, "%d", tm.tm_mday);
    char date[32];
    strcpy(date, reservation->travelDate);
   // int date = atoi(reservation->travelDate);
    // char filename[24];
    // sprintf(filename, "%d.txt", date);
    char* filename = "summary.txt";
    char buffer[BUFFER_SIZE];
    int tick = 1;
    FILE *file;

    sem_t *file_write = sem_open("/file_write", O_CREAT, 0666, 0);
    sem_wait(file_write);

    if (file_exists(filename)){
```

```
        file = fopen(filename, "r");
        int tmp = 0;
        char *rest = "";
        while(fgets(buffer, sizeof(buffer), file)){
            char *tok = strtok_r(buffer, " \t", &rest);

            if (atoi(tok) > tmp) tmp = atoi(tok);

            tick++;
            memset(buffer, 0, sizeof(buffer));
            memset(tok, 0, sizeof(tok));
        }
        tick = tmp + 1;
        fclose(file);
    }

    char ticket[24];
    sprintf(ticket, "%d", tick);
    //sprintf(ticket, "%s-%d", date, tick);
    //sprintf(ticket, "%d-%d", date, tick);
    for (int j = 0; j < numberTravelers; j++)
        strcpy((reservation+j)->ticket_number, ticket);
    file = fopen(filename, "a");
    char buffer_out[BUFFER_SIZE];
    for (int i = 0; i < numberTravelers; i++){
        sprintf(buffer_out, "%s\t%c\t%s\t%s\t%s\t%d\t%s\t%s\tOG\n", ticket,
server, (reservation+i)->customerName, (reservation+i)->dob,
(reservation+i)->gender, (reservation+i)->govID, (reservation+i)->travelDate,
(reservation+i)->seat);
        fprintf(file, "%s", buffer_out);
        puts((reservation+i)->dob);
    }
    fclose(file);
    sem_post(file_write);

   // receipt(reservation, numberTravelers, server, "");
}
```

```
// Has critical section - Write, threadsafe X
// Adds travelers with the given ticket number
void add_travelers(char server, Reservation* reservation, int
numberNewTravelers, char* ticket){
    // time_t t = time(NULL);
    // struct tm tm = *localtime(&t);
    // char today[9];
    // snprintf(today, sizeof(char) * 8, "%d%d%d", tm.tm_mon + 1, tm.tm_mday,
tm.tm_year + 1900);
    int date = atoi(reservation->travelDate);
    // char filename[24];
    // sprintf(filename, "%d.txt", date);
    char* filename = "summary.txt";
    char buffer[BUFFER_SIZE];
    FILE *file;

    sem_t *file_write = sem_open("/file_write", O_CREAT, 0666, 0);
    sem_wait(file_write);

    for (int j = 0; j < numberNewTravelers; j++)
        strcpy((reservation+j)->ticket_number, ticket);
    file = fopen(filename, "a");
    char buffer_out[BUFFER_SIZE];
    for (int i = 0; i < numberNewTravelers; i++){
        sprintf(buffer_out, "%s\t%c\t%s\t%s\t%s\t%d\t%s\t%s\tOG\n", ticket,
server, (reservation+i)->customerName, (reservation+i)->dob,
(reservation+i)->gender, (reservation+i)->govID, (reservation+i)->travelDate,
(reservation+i)->seat);
        fprintf(file, "%s", buffer_out);
    }
    fclose(file);
    sem_post(file_write);
}

// Has critical section - READ, threadsafe X
// Updated 4/22, Now takes ticket number and array of reservations and places
all
```

```c
// reservations that match the ticket number are placed into the array (in
place in memory)
void inquiry(char *ticket, Reservation* info, int* numTravelers){
    // find ticket
    char date[9];
    for (int i = 0; i < 7; i++) *(date+i) = *(ticket+i);
    // char filename[16];
    // sprintf(filename, "%s.txt", date);
    char* filename = "summary.txt";

    sem_t *file_write = sem_open("/file_write", O_CREAT, 0666, 0);
    sem_t *file_read = sem_open("/file_read", O_CREAT, 0666, 0);

    // first reader
    sem_wait(file_read);
    int shm_fd = shm_open("readers", O_RDWR, 0666);
    int *reader_count;
    reader_count = mmap(0, sizeof(int), PROT_WRITE, MAP_SHARED, shm_fd, 0);
    *reader_count = *reader_count + 1;

    if (*reader_count == 1){
        sem_wait(file_write);
    }
    sem_post(file_read);
    Reservation resy;
    int count = 0;
    if (file_exists(filename)){
        char buffer[512];
        char temp[512];
        char *rest = buffer;
        char *token;
        int flag = 0;
        FILE *file = fopen(filename, "r");
        fgets(buffer, sizeof(buffer), file);
        while(!feof(file)){
            strcpy(temp, buffer);
            token = strtok_r(temp, "\t", &rest);  //Ticket
            if (strcmp(ticket, token) == 0){
```

```
                flag = 1;
                strcpy(resy.ticket_number, ticket);
                token = strtok_r(NULL, "\t", &rest); // Server
                token = strtok_r(NULL, "\t", &rest); // Name
                strcpy(resy.customerName, token);
                token = strtok_r(NULL, "\t", &rest); // dob
                strcpy(resy.dob, token);
                token = strtok_r(NULL, "\t", &rest); // gender
                strcpy(resy.gender, token);
                token = strtok_r(NULL, "\t", &rest); // govid
                resy.govID = atoi(token);
                token = strtok_r(NULL, "\t", &rest); // traveldate
                strcpy(resy.travelDate, token);
                token = strtok_r(NULL, "\t", &rest); // seat
                strcpy(resy.seat, token);

                *(info + count) = resy;
                count++;
            }
            fgets(buffer, sizeof(buffer), file);
        }
        if (!flag) {
            printf("Ticket not found.\n");
        }
        fclose(file);
    }else {
        printf("Ticket not found!");
    }

    *numTravelers = count;

    // last reader
    sem_wait(file_read);
    *reader_count = *reader_count - 1;
    if (*reader_count == 0){
        sem_post(file_write);
    }
    sem_post(file_read);
```

```
    munmap(reader_count, sizeof(int));
    close(shm_fd);
}

// Has critical section - WRITE, threadsafe X
// Updated 4/23, Now requires both ticket number and name
void update_train_seats(char* ticket, char *name, char* seat, char server){
    // create filename from date
    char date[9];
    for (int i = 0; i < 7; i++) *(date+i) = *(ticket+i);
    // char filename[16];
    // sprintf(filename, "%s.txt", date);
    char* filename = "summary.txt";

    char buffer_in[MAX_SEATS][BUFFER_SIZE];
    char line[BUFFER_SIZE];
    char temp[BUFFER_SIZE];
    char *rest = temp;
    char buffer_out[BUFFER_SIZE];
    char *token;
    int count = 0;
    int flag = 0;
    FILE *file;

    sem_t *file_write = sem_open("/file_write", O_CREAT, 0666, 0);
    sem_wait(file_write);

    if (file_exists(filename)){
        file = fopen(filename, "r");
        fgets(line, sizeof(line), file);
        while(!feof(file)){
            strcpy(temp, line);
            token = strtok_r(temp, "\t", &rest);
            if (strcmp(token, ticket) == 0){
                strcpy(buffer_out, token);
                strcat(buffer_out, "\t");
                token = strtok_r(NULL, "\t", &rest);
```

```
                strcat(buffer_out, (char[2]){(char) server, '\0'}); // casting
char server name as string for strcat
                strcat(buffer_out, "\t");
                token = strtok_r(NULL, "\t", &rest);
                if (strcmp(token, name) == 0){
                    flag = 1;
                    strcat(buffer_out, token);
                    strcat(buffer_out, "\t");
                    for (int i = 0; i < 4; i++){
                        token = strtok_r(NULL, "\t", &rest);
                        strcat(buffer_out, token);
                        strcat(buffer_out, "\t");
                    }
                    strcat(buffer_out, seat);
                    strcat(buffer_out, "\tMD");
                    strcat(buffer_out, "\n");
                    strcpy(buffer_in[count], buffer_out);
                } else{
                    strcpy(buffer_in[count], line);
                }
            }else{
                strcpy(buffer_in[count], line);
            }
            fgets(line, sizeof(buffer_in), file);
            count++;
        }
        fclose(file);
        if (!flag) {
            printf("Ticket not found!\n");
        }
        file = fopen(filename, "w");
        for (int i = 0; i < count; i++){
            fprintf(file, "%s", buffer_in[i]);
        }
        fclose(file);
    }else{
        printf("Ticket not found!\n");
    }
```

```
    sem_post(file_write);
}

// Has critical section - WRITE, threadsafe X
// Updated 4/25, cancel by ticket number only
void cancel_reservation(char* ticket){
    // create filename from date
    char date[9];
    for (int i = 0; i < 7; i++) *(date+i) = *(ticket+i);
    // char filename[16];
    // sprintf(filename, "%s.txt", date);
    char* filename = "summary.txt";

    char buffer_in[MAX_SEATS][BUFFER_SIZE];
    char line[BUFFER_SIZE];
    char temp[BUFFER_SIZE];
    char *rest = temp;
    char *token;
    int count = 0;
    int flag = 0;
    FILE *file;

    sem_t *file_write = sem_open("/file_write", O_CREAT, 0666, 0);
    sem_wait(file_write);

    if (file_exists(filename)){
        file = fopen(filename, "r");
        fgets(line, sizeof(line), file);
        while(!feof(file)){
            strcpy(temp, line);
            token = strtok_r(temp, "\t", &rest);
            if (strcmp(token, ticket) == 0){
                flag = 1;
                count--;
            }else{
                strcpy(buffer_in[count], line);
            }
            fgets(line, sizeof(buffer_in), file);
```

```
            count++;
        }
        if (!flag) printf("Ticket not found!\n");
        fclose(file);

        file = fopen(filename, "w");
        for (int i = 0; i < count; i++){
            fprintf(file, "%s", buffer_in[i]);
        }
        fclose(file);
    }else{
        printf("Ticket not found!\n");
    }
    sem_post(file_write);
}

// Has critical section - WRITE, threadsafe X
// Cancel by ticket number and name
void remove_traveler(char* ticket, char* name){
    // create filename from date
    char date[9];
    for (int i = 0; i < 7; i++) *(date+i) = *(ticket+i);
    // char filename[16];
    // sprintf(filename, "%s.txt", date);
    char* filename = "summary.txt";

    char buffer_in[MAX_SEATS][BUFFER_SIZE];
    char line[BUFFER_SIZE];
    char temp[BUFFER_SIZE];
    char *rest = temp;
    char *token;
    int count = 0;
    int flag = 0;
    FILE *file;

    sem_t *file_write = sem_open("/file_write", O_CREAT, 0666, 0);
    sem_wait(file_write);
```

```
    if (file_exists(filename)){
        file = fopen(filename, "r");
        fgets(line, sizeof(line), file);
        while(!feof(file)){
            strcpy(temp, line);
            token = strtok_r(temp, "\t", &rest);
            if (strcmp(token, ticket) == 0){
                token = strtok_r(NULL, "\t", &rest);
                token = strtok_r(NULL, "\t", &rest);
                if (strcmp(token, name) == 0){
                    flag = 1;
                    count--;
                }else{
                    strcpy(buffer_in[count], line);
                }
            }else{
                strcpy(buffer_in[count], line);
            }
            fgets(line, sizeof(buffer_in), file);
            count++;
        }
        if (!flag) printf("Ticket not found!\n");
        fclose(file);

        file = fopen(filename, "w");
        for (int i = 0; i < count; i++){
            fprintf(file, "%s", buffer_in[i]);
        }
        fclose(file);
    }else{
        printf("Ticket not found!\n");
    }
    sem_post(file_write);
}

// Has critical section - READ, threadafe X
// Puts a string of the available seats into options delimited by spaces
//void available_seats(int date, char* options, int* numberAvailable){
```

```c
void available_seats(char *date, char* options, int* numberAvailable){
    char all_seats[3][9][4];
    char x[4];
    char first = 'A';
    int num = 1;
    for (int i = 0; i < 3; i ++){
        for (int j = 0; j < 9; j++){
            sprintf(x, "%c%d", first + i, num + j);
            strcpy(all_seats[i][j], x);
        }
    }
    printf("\n");
    char taken[MAX_SEATS][3];
    char buffer[BUFFER_SIZE];
    char temp[BUFFER_SIZE];
    char *rest = temp;
    char available[100];
    char *token;
    int count = 0;
    // char filename[16];
    // sprintf(filename, "%d.txt", date);
    char* filename = "summary.txt";
    FILE *file;

    sem_t *file_write = sem_open("/file_write", O_CREAT, 0666, 0);
    sem_t *file_read = sem_open("/file_read", O_CREAT, 0666, 0);

    // first reader
    sem_wait(file_read);
    int shm_fd = shm_open("readers", O_RDWR, 0666);
    int *reader_count;
    reader_count = mmap(0, sizeof(int), PROT_WRITE, MAP_SHARED, shm_fd, 0);
    *reader_count = *reader_count + 1;

    if (*reader_count == 1){
        sem_wait(file_write);
    }
    sem_post(file_read);
```

```
if (file_exists(filename)){
    file = fopen(filename, "r");
    fgets(buffer, sizeof(buffer), file);
    strcpy(temp, buffer);
    while(!feof(file)){
        token = strtok_r(temp, "\t", &rest);
        for (int i = 0; i < 6; i++) token = strtok_r(NULL, "\t", &rest);
        if (!strcmp(token, date)){
        //if(atoi(token) == date){
            token = strtok_r(NULL, "\t", &rest);
            strcpy(taken[count], token);
        }
        fgets(buffer, sizeof(buffer), file);
        strcpy(temp, buffer);
        count++;
    }
}else{
    printf("Ticket not found!\n");
}
int availSeats = 27;
for (int i = 0; i < 3; i++){
    for (int j = 0; j < 9; j++){
        for (int k = 0; k < count; k++){
            if (strcmp(all_seats[i][j], taken[k]) == 0){
                strcpy(all_seats[i][j], "XX");
                availSeats--;
            }
        }
    }
}
char row[38];
strcpy(options, "");
for (int i = 0; i < 3; i ++){
    for (int j = 0; j < 9; j++){
        if (strcmp(all_seats[i][j], "XX") != 0){
            strcat(options, all_seats[i][j]);
            strcat(options, " ");
```

```
            }
            strcat(options, "\t");
        }
        strcat(options, "\n");
    }
    *numberAvailable = availSeats;
    // last reader
    sem_wait(file_read);
    *reader_count = *reader_count - 1;
    if (*reader_count == 0){
        sem_post(file_write);
    }
    sem_post(file_read);
    munmap(reader_count, sizeof(int));
    close(shm_fd);
}

// Has critical section by calling available_seats, threadsafe X
// Returns 1 if seat is available, and returns 0 if seat is taken
//int check_seat(int date, char *seat){
int check_seat(char *date, char *seat){
    char avail[112];
    int* temp;
    available_seats(date, avail, temp);
    char* s = strstr(avail, seat);
    if (s != NULL){
        printf("Found seat %s\n", seat);
        return 1;
    }else{
        printf("Seat %s not found\n", seat);
        return 0;
    }
}

void get_travel_date(char* ticket, char* travelDate){
    char* filename = "summary.txt";

    sem_t *file_write = sem_open("/file_write", O_CREAT, 0666, 0);
```

```c
    sem_t *file_read = sem_open("/file_read", O_CREAT, 0666, 0);

    // first reader
    sem_wait(file_read);
    int shm_fd = shm_open("readers", O_RDWR, 0666);
    int *reader_count;
    reader_count = mmap(0, sizeof(int), PROT_WRITE, MAP_SHARED, shm_fd, 0);
    *reader_count = *reader_count + 1;

    if (*reader_count == 1){
        sem_wait(file_write);
    }
    sem_post(file_read);
    int count = 0;
    if (file_exists(filename)){
        char buffer[512];
        char temp[512];
        char *rest = buffer;
        char *token;
        int flag = 0;
        FILE *file = fopen(filename, "r");
        fgets(buffer, sizeof(buffer), file);
        while(!feof(file)){
            strcpy(temp, buffer);
            token = strtok_r(temp, "\t", &rest);  //Ticket
            if (strcmp(ticket, token) == 0){
                flag = 1;
                token = strtok_r(NULL, "\t", &rest); // Server
                token = strtok_r(NULL, "\t", &rest); // Name
                token = strtok_r(NULL, "\t", &rest); // dob
                token = strtok_r(NULL, "\t", &rest); // gender
                token = strtok_r(NULL, "\t", &rest); // govid
                token = strtok_r(NULL, "\t", &rest); // traveldate
                strcpy(travelDate, token);
                break;
                token = strtok_r(NULL, "\t", &rest); // seat
                count++;
            }
```

```
            fgets(buffer, sizeof(buffer), file);
        }
        if (!flag) {
            printf("Ticket not found.\n");
        }
        fclose(file);
    }else {
        printf("Ticket not found!");
    }

    // last reader
    sem_wait(file_read);
    *reader_count = *reader_count - 1;
    if (*reader_count == 0){
        sem_post(file_write);
    }
    sem_post(file_read);
    munmap(reader_count, sizeof(int));
    close(shm_fd);
}

// Has critical section - READ
void get_num_travelers(char* ticket, int* numTravelers){
    char* filename = "summary.txt";

    sem_t *file_write = sem_open("/file_write", O_CREAT, 0666, 0);
    sem_t *file_read = sem_open("/file_read", O_CREAT, 0666, 0);

    // first reader
    sem_wait(file_read);
    int shm_fd = shm_open("readers", O_RDWR, 0666);
    int *reader_count;
    reader_count = mmap(0, sizeof(int), PROT_WRITE, MAP_SHARED, shm_fd, 0);
    *reader_count = *reader_count + 1;

    if (*reader_count == 1){
        sem_wait(file_write);
    }
```

```
    sem_post(file_read);
    int count = 0;
    if (file_exists(filename)){
        char buffer[512];
        char temp[512];
        char *rest = buffer;
        char *token;
        int flag = 0;
        FILE *file = fopen(filename, "r");
        fgets(buffer, sizeof(buffer), file);
        while(!feof(file)){
            strcpy(temp, buffer);
            token = strtok_r(temp, "\t", &rest);  //Ticket
            if (strcmp(ticket, token) == 0){
                flag = 1;
                count++;
            }
            fgets(buffer, sizeof(buffer), file);
        }
        if (!flag) {
            printf("Ticket not found.\n");
        }
        fclose(file);
    }else {
        printf("Ticket not found!");
    }
    *numTravelers = count;

    // last reader
    sem_wait(file_read);
    *reader_count = *reader_count - 1;
    if (*reader_count == 0){
        sem_post(file_write);
    }
    sem_post(file_read);
    munmap(reader_count, sizeof(int));
    close(shm_fd);
}
```

```c
// Create receipt X
void receipt(Reservation* reservations, int numberTravelers, char server, char*
receipt){
    char buffer[BUFFER_SIZE];
    char buffer_out[1024] = "";

    char filename[40];
    sprintf(filename, "Receipt-%s.txt", reservations->ticket_number);

    FILE *file = fopen(filename, "w");
    fprintf(file, "%s%s\n", "Receipt for ticket number: ",
reservations->ticket_number);
    fprintf(file, "%s%c\n\n", "Server: ", server);
    for (int i = 0; i < numberTravelers; i++){
        sprintf(buffer, "%s\t%s\t%s\t%s\t%d\t%s\t%s\n",
(reservations+i)->ticket_number, (reservations+i)->customerName,
(reservations+i)->dob, (reservations+i)->gender, (reservations+i)->govID,
(reservations+i)->travelDate, (reservations+i)->seat);
        fprintf(file, "%s", buffer);
    }
    fclose(file);
    file = fopen(filename, "r");
    char rec[1024] = "";
    while(!feof(file)){
        memset(buffer, 0, sizeof(buffer));
        fgets(buffer, BUFFER_SIZE, file);
        strcat(rec, buffer);
    }
    //for (int i = 0; i < 1024 || rec[i] == '\0'; i++){
    for (int i = 0; i < BUFFER_SIZE-24 || rec[i] == '\0'; i++){
        *(receipt+i) = *(rec+i);
    }
    fclose(file);


}
```

```c
// testing purposes only
void testX(char name){
    sem_t *file_write = sem_open("/file_write", O_CREAT, 0666, 0);
    sem_t *file_read = sem_open("/file_read", O_CREAT, 0666, 0);

    // first reader
    sem_wait(file_read);
    int shm_fd = shm_open("readers", O_RDWR, 0666);
    int *reader_count;
    reader_count = mmap(0, sizeof(int), PROT_WRITE, MAP_SHARED, shm_fd, 0);
    *reader_count = *reader_count + 1;
    printf("%c entering: render_count: %d\n", name, *reader_count);

    if (*reader_count == 1){
        sem_wait(file_write);
        printf("%c LOCKED WRITE\n", name);
    }
    sem_post(file_read);

    printf("Reader %c  is in.\n", name);
    sleep(4);

    // last reader
    sem_wait(file_read);
    *reader_count = *reader_count - 1;
    printf("%c exiting: render_count: %d\n", name, *reader_count);
    if (*reader_count == 0){
        sem_post(file_write);
        printf("%c UNLOCKED WRITE\n",name);
    }
    sem_post(file_read);

    printf("Reader %c  successfully exited\n", name);
    munmap(reader_count, sizeof(int));
    close(shm_fd);

}
```

Bill Liando

Job Function:

1. Handle the message passing on the client side of the reservation system, working with Jareth to output messages to the user from the server.

Client operations for the following have been implemented:

1. Client side message passing added to Jareth's client program

What is left to do:

1. There is nothing left to do.

Code Limitations:

1. There are no limitations.

client.c:

```
//Bill Liando
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/wait.h>
#include <arpa/inet.h>   // client
#include <sys/socket.h>
#include <netinet/in.h> // server
#include <pthread.h>
#include <semaphore.h>

#define IP "127.0.0.1"
#define PORT 8019
#define BUFFER_SIZE 2048
#define SERVER_COUNT 5

int connect_to_server()
{
        int sock, i;
```

```
	struct sockaddr_in address;
	socklen_t addrlen;

	if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
	{
		puts("Failed to create socket");
		return -1;
	}


	while (1)
	{
		for (i = 0; i < SERVER_COUNT; i++)
		{
			address.sin_family = AF_INET;
			address.sin_port = htons(PORT + i);
			char temp[20];
			sprintf(temp, "port %d", PORT+i);
			puts(temp);
			if (inet_pton(AF_INET, IP, &address.sin_addr) <= 0)
			{
				puts("Invalid address");
				return -1;
			}

			// TRY to connect to server i
			addrlen = sizeof(address);
			if (connect(sock, (struct sockaddr *)&address, addrlen) < 0) // failed to
connect to server i
			{
				char temp[120];
				sprintf(temp, "Failed to connect to server %d", i + 1);
				puts(temp);
			}
			else // connected to server i
			{
				//puts("Connection successful");
				return sock;
			}
		}
		puts("All servers full, waiting 5 seconds and retrying");
		sleep(5);
	}
}


int main()
```

```
{
        char message[BUFFER_SIZE-24];
        char buffer[BUFFER_SIZE] = {0};
        char *exit = "Test";
        int sock = connect_to_server();
        if (sock < 0) puts("There was an error creating the client socket");

        while (1)
        {
                memset(buffer, 0, sizeof(buffer));
                recv(sock, buffer, BUFFER_SIZE, 0);
                puts(buffer);
                if (!strcmp(buffer, exit)) break;

                memset(message, 0, sizeof(message));
                fgets(message, BUFFER_SIZE-24, stdin);
                message[strlen(message)-1] = '\0';
                send(sock, message, BUFFER_SIZE-24, 0);
        }

        close(sock);
        puts("client exited");
        return 0;
}
```

}

Jareth Harmon

Job function:

1. client/server communication
2. creation of multithreaded server; and a thread_pool for each server
3. using semaphore to restrict access to the train_seats data structure

Work completed so far:

1. multi-threaded server with a defined number of threads
2. each thread can be connected to with a client program
3. semaphore-controlled access to train seats

What is left to do:

Limitations:

1. Server must be closed with CTRL+C; does not seem to leave processes or semaphores locked though

server.c

```
// Group: E
// Name: Jareth Harmon
// Email: jareth.harmon@okstate.edu

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/wait.h>
#include <arpa/inet.h>  // client
#include <sys/socket.h>
#include <netinet/in.h> // server
#include <pthread.h>
#include <semaphore.h>
#include <time.h>
#include "Backend.h"
#include "Reservation.h"
#include "Server.h"

#define IP "127.0.0.1"
#define PORT 8019
#define BUFFER_SIZE 2048
#define THREAD_POOL_SIZE 3
#define SEAT_CAPACITY 27

Client priority_queue[THREAD_POOL_SIZE];

pthread_t thread_pool[THREAD_POOL_SIZE];
pthread_t queue_t;

sem_t semaphores[THREAD_POOL_SIZE];
sem_t queue, enforce_thread_limit;

int thread_in_use[THREAD_POOL_SIZE] = {0};
int num_seats = 27, pq_size = 0, run = 1;

void example(int purchased_seats)
{
        char out[120];
        if (purchased_seats > num_seats)
        {
                // do something if there are not enough seats left
                puts("not enough seats left");
```

```
	}
	else
	{
		// let the client access the seat list for selection/deselection
		num_seats -= purchased_seats;
		sprintf(out, "Number of seats remaining: %d", num_seats);
		puts(out);
	}
	sleep(7);				// simulates client choosing seats
	sem_post(&queue);	// signal to the queue that it can allow a new client to access the seat
list
}

// insert a new member into the priority queue; at the correct spot relative to its priority
int enqueue(int index_n, int priority_n)
{
	for (int i = 0; i < THREAD_POOL_SIZE; i++)
	{
		// if the current index in the priority queue is empty; just insert
		// means that it is the lowest priority or the first
		// probably not needed
		if (priority_queue[i].index == -1)
		{
			priority_queue[i].index = index_n;
			priority_queue[i].priority = priority_n;
			pq_size++;
			break;
		}
		if (priority_queue[i].priority < priority_n)
		{
			int temp = pq_size;
			while (temp > i)
			{
				// start at the cell after the last populated cell;
				// shift each member to the right by 1 UNTIL
				// you reach the insertion point of the newly enqueued member
				priority_queue[temp].index = priority_queue[temp - 1].index;
				priority_queue[temp].priority = priority_queue[temp - 1].priority;
				temp--;
			}
			// insert the new member
			priority_queue[i].index = index_n;
			priority_queue[i].priority = priority_n;
			pq_size++;
			break;
		}
	}
```

```
}

// increase the priority of all members of the queue by 1
void increase_priority()
{
	for (int i = 0; i < pq_size; i++)
	{
		if (priority_queue[i].index != -1) priority_queue[i].priority++;
	}
}

// remove a member from the queue at the specified index (will always be 0 in our use case though)
void dequeue(int i)
{
	if (i < THREAD_POOL_SIZE)
	{
		while (i < pq_size)
		{
			// shift all members > i to the left
			priority_queue[i].index = priority_queue[i + 1].index;
			priority_queue[i].priority = priority_queue[i + 1].priority;
			i++;
		}
		pq_size--; // decrease size of queue
		priority_queue[pq_size].index = -1; // set last elements index to -1 (checked
elsewhere)
		increase_priority(); // increase_priority of all current members
	}
}

// returns the member of the queue with the highest priority (the one at index 0)
int get_highest_priority()
{
	int index_n = priority_queue[0].index;
	int priority_n = priority_queue[0].priority;
	char temp[40];
	sprintf(temp, "index %d : priority %d", index_n, priority_n);
	puts(temp);
	dequeue(0);
	return index_n;
}

// thread for handling the priority queue
void *queue_thread(void *arg)
{
	// calls sem_wait on each thread's semaphore so that the thread will actually wait
	for (int i = 0; i < THREAD_POOL_SIZE; i++) sem_wait(&semaphores[i]);
```

```
	while(1)//while(run)
	{
		if (pq_size > 0) // busy wait; not ideal but need some condition here
		{
			// this could potentially cause an issue if the client disconnects early; not sure if we are meant to handle that or not
			sem_wait(&queue); // wait for the queue to be ready to use
			int index = get_highest_priority(); // get the index of the thread with the highest priority
			sem_post(&semaphores[index]); // allow that thread to access the seat list
		}
	}
	pthread_exit(NULL);
}

void *server_thread(void *arg)
{
	int purchased_seats, sock, client_choice, id_index, priority;
	char buffer[BUFFER_SIZE] = {0};
	char message[BUFFER_SIZE - 24];

	char *temp = (char *)arg;
	char *rest = temp;
	char *temp1 = strtok_r(temp, ",", &rest);
	char *temp2 = strtok_r(NULL, ",", &rest);
	char *temp3 = strtok_r(NULL, ",", &rest);
	char name = *temp3;

	sock = atoi(temp1);
	id_index = atoi(temp2);

	// Interact with client here
	while (1)
	{
		// send menu to client
		char* menu = "1. Make a reservation\n2. Inquiry about the ticket\n3. Modify the reservation\n4. Cancel the reservation\n5. Exit the program";
		char* makeRes = "How many travelers: ";
		char* getName = "Enter passenger name: ";
		char* getGender = "Enter passenger gender: ";
		char* getDOB = "Enter passenger DOB: ";
		char* getGovID = "Enter passenger gov ID: ";
		//char* getTravelDate = "Enter travel date (ex: MMDDYYYY): ";
		char* getTravelDate = "Enter travel date (today or tomorrow): ";
		char today[32];
		char tomorrow[32];
```

```
                memset(buffer, 0, sizeof(buffer));
                send(sock, menu, BUFFER_SIZE, 0);

                // receive client choice
                recv(sock, buffer, BUFFER_SIZE, 0);
                client_choice = atoi(buffer);
                // these functions will need to communicate with the user according to their
specifications; and parse their input into variables
                if (client_choice == 1) { // Make reservation
                        // Get # travelers
                        memset(buffer, 0, sizeof(buffer));
                        send(sock, makeRes, BUFFER_SIZE-24, 0);
                        recv(sock, buffer, BUFFER_SIZE, 0);
                        int numTravelers = atoi(buffer);
                        Reservation* reservations =
(Reservation*)malloc(sizeof(Reservation)*numTravelers);

                        get_date(today, tomorrow);

                        // get Travel date
                        memset(buffer, 0, sizeof(buffer));
                        send(sock, getTravelDate, BUFFER_SIZE, 0);
                        recv(sock, buffer, BUFFER_SIZE, 0);
                        char travelDate[32];
                        while (1)
                        {
                                if (!strcasecmp(buffer, "today"))
                                {
                                        strcpy(travelDate, today);
                                        break;
                                }
                                else if (!strcasecmp(buffer, "tomorrow"))
                                {
                                        strcpy(travelDate, tomorrow);
                                        break;
                                }
                                else
                                {
                                        memset(buffer, 0, sizeof(buffer));
                                        send(sock, getTravelDate, BUFFER_SIZE, 0);
                                        recv(sock, buffer, BUFFER_SIZE, 0);
                                }
                        }
                        //int travelDate = atoi(buffer);

                        // Enough seats?
                        int seatCount;
```

```
                        char arr[112];
                        available_seats(travelDate, arr, &seatCount);

                        char tete[120];
                        sprintf(tete, "==%d==", seatCount);
                        puts(tete);

                        if (seatCount >= numTravelers){
                                enqueue(id_index, numTravelers);
                                sem_wait(&semaphores[id_index]);

                                memset(arr, 0, sizeof(arr));
                                available_seats(travelDate, arr, &seatCount);
                                memset(tete, 0, sizeof(tete));
                                sprintf(tete, "==%d==", seatCount);
                                puts(tete);
                                if (seatCount < numTravelers)
                                {
                                        char* notEnough = "There are no longer enough seats!
Disconnecting\nPress Enter to exit.";
                                        send(sock, notEnough, BUFFER_SIZE-24, 0);
                                        recv(sock, buffer, BUFFER_SIZE, 0);
                                        break;
                                }

                                // Get info
                                for (int i = 0; i < numTravelers; i++){
                                        //char tmp[120];
                                        //sprintf(tmp, "%d", travelDate);
                                        //strcpy((reservations+i)->travelDate, tmp);
                                        strcpy((reservations+i)->travelDate, travelDate);
                                        //*(reservations+i)->travelDate = travelDate;
                                        // get name
                                        memset(buffer, 0, sizeof(buffer));
                                        send(sock, getName, BUFFER_SIZE, 0);
                                        recv(sock, buffer, BUFFER_SIZE, 0);
                                        strcpy((reservations+i)->customerName, buffer);
                                        // get dob
                                        memset(buffer, 0, sizeof(buffer));
                                        send(sock, getDOB, BUFFER_SIZE, 0);
                                        recv(sock, buffer, BUFFER_SIZE, 0);
                                        strcpy((reservations+i)->dob, buffer);
                                        // get gender
                                        memset(buffer, 0, sizeof(buffer));
                                        send(sock, getGender, BUFFER_SIZE, 0);
                                        recv(sock, buffer, BUFFER_SIZE, 0);
                                        strcpy((reservations+i)->gender, buffer);
```

```
                                        // get govID
                                        memset(buffer, 0, sizeof(buffer));
                                        send(sock, getGovID, BUFFER_SIZE, 0);
                                        recv(sock, buffer, BUFFER_SIZE, 0);
                                        (reservations+i)->govID = atoi(buffer);
                                }
                                char* confirmMessage = "Do you want to make reservation (yes/no):
";
                                memset(buffer, 0, sizeof(buffer));
                                send(sock, confirmMessage, BUFFER_SIZE, 0);
                                recv(sock, buffer, BUFFER_SIZE, 0);
                                puts(buffer);
                                if (strcmp(buffer, "yes") == 0){
                                        for(int j = 0; j < numTravelers; j++){
                                                char seats[112];
                                                int trav_temp;
                                                available_seats((reservations+j)->travelDate, seats,
&trav_temp);
                                                //available_seats(atoi((reservations+j)->travelDate),
seats, &trav_temp);
                                                char *seatMessage = "Pick your seat: ";
                                                memset(message, 0, sizeof(message));
                                                sprintf(message, "%s\n%s", seats, seatMessage);
                                                send(sock, message, BUFFER_SIZE-24, 0);

                                                memset(buffer, 0, sizeof(buffer));
                                                recv(sock, buffer, BUFFER_SIZE, 0);
                                                strcpy((reservations+j)->seat, buffer);
                                                puts(buffer);

                                                if (!check_seat(travelDate, buffer))
                                                {
                                                        memset(message, 0, sizeof(message));
                                                        sprintf(message, "Seat %s is not available,
please pick another seat.\nPress Enter to try again.", buffer);
                                                        send(sock, message, BUFFER_SIZE-24, 0);
                                                        recv(sock, buffer, BUFFER_SIZE, 0);
                                                        j--;
                                                }
                                        }
                                        make_reservation(name, reservations, numTravelers);
                                        memset(message, 0, sizeof(message));
                                        receipt(reservations, numTravelers, name, message);
                                        strcat(message, "\nPlease press Enter to return to main
menu");
                                        send(sock, message, BUFFER_SIZE, 0);
```

```
                                        memset(buffer, 0, sizeof(buffer));
                                        recv(sock, buffer, BUFFER_SIZE-24, 0);

                                        free(reservations);
                                }
                                //available_seats(travelDate, arr, &seatCount);

                                //tete[120];
                                ///sprintf(tete, "==Seatcount:%d==", seatCount);
                                //puts(tete);
                                sem_post(&queue);
                        }else{
                                char* notEnough = "There are not enough seats!\nPress Enter to
return to menu.";
                                send(sock, notEnough, BUFFER_SIZE-24, 0);
                                recv(sock, buffer, BUFFER_SIZE, 0);
                                //break;
                        }
                }
                else if (client_choice == 2){ // inquiry
                        char* inq = "Enter your ticket number: ";
                        Reservation* info =
(Reservation*)malloc(sizeof(Reservation)*SEAT_CAPACITY);
                        send(sock, inq, BUFFER_SIZE, 0);
                        recv(sock, buffer, BUFFER_SIZE, 0);
                        int trav;
                        inquiry(buffer, info, &trav);
                        char info_message[BUFFER_SIZE];
                        memset(info_message, 0, sizeof(info_message));
                        char temp[BUFFER_SIZE] = "";
                        memset(temp, 0, sizeof(temp));
                        char as[120];
                        memset(as, 0, sizeof(as));
                        sprintf(as, "Number of travellers: %d\n", trav);
                        strcat(info_message, as);
                        for (int i = 0; i < trav; i++){
                                memset(temp, 0, sizeof(temp));
                                sprintf(temp, "%s\t%s\t%s\t%d\t%s\t%s\t%s\n",
(info+i)->customerName, (info+i)->dob, (info+i)->gender, (info+i)->govID, (info+i)->travelDate,
(info+i)->seat, (info+i)->ticket_number);
                                strcat(info_message, temp);
                                puts(temp);
                        }
                        strcat(info_message, "Press Enter to return to main menu.");
                        send(sock, info_message, BUFFER_SIZE-24, 0);
                        recv(sock, buffer, BUFFER_SIZE, 0);
                }
```

```
                else if (client_choice == 3){
                        char* modify_options = "Select which to modify\nA. Change Seat\nB. Change
Travel Date\nC. Remove Travelers\nD. Add Travelers\n";
                        send(sock, modify_options, BUFFER_SIZE, 0);
                        recv(sock, buffer, BUFFER_SIZE, 0);
                        if (buffer[0] == 'A'){ // Change seat
                                enqueue(id_index, 1);
                                sem_wait(&semaphores[id_index]);

                                // get ticket
                                char* inq = "Enter your ticket number: ";
                                send(sock, inq, BUFFER_SIZE, 0);
                                recv(sock, buffer, BUFFER_SIZE, 0);
                                char ticket[20] = "";
                                strcpy(ticket, buffer);
                                // get number of travelers on ticket
                                Reservation* temp =
(Reservation*)malloc(sizeof(Reservation)*SEAT_CAPACITY);
                                int trav;
                                inquiry(ticket, temp, &trav);
                                // get travelDate
                                char travDate[32] = "";
                                strcpy(travDate, temp->travelDate);
                                // Enough seats?
                                int seatCount;
                                char arr[112];
                                available_seats(travDate, arr, &seatCount);
                                //available_seats(atoi(travDate), arr, &seatCount);

                                char* modifyName = "Enter name to be modified: ";
                                send(sock, modifyName, BUFFER_SIZE-24, 0);
                                recv(sock, buffer, BUFFER_SIZE, 0);
                                char r_name[64];
                                strcpy(r_name, buffer);
                                // Reservation* info =
(Reservation*)malloc(sizeof(Reservation)*SEAT_CAPACITY);
                                char seats[112];
                                int temporary = 0;
                                //available_seats(atoi(temp->travelDate), seats, &temporary);
                                available_seats(temp->travelDate, seats, &temporary);
                                if (seatCount <= 0){
                                        char* full = "Cannot change seats the train is full.\n";
                                        send(sock, full, BUFFER_SIZE, 0);
                                        sem_post(&queue);
                                        break;
                                } else{
                                        char seat_msg[BUFFER_SIZE];
```

```
                                        sprintf(seat_msg, "%s\n%s: ", "Select an available seat",
seats);
                                        send(sock, seat_msg, BUFFER_SIZE, 0);
                                        recv(sock, buffer, BUFFER_SIZE, 0);
                                        char newSeat[BUFFER_SIZE];
                                        strcpy(newSeat, buffer);
                                        update_train_seats(ticket, r_name, newSeat, name);

                                        char* complete_msg = "Modification complete.\nPress Enter
to return to the menu.";
                                        send(sock, complete_msg, BUFFER_SIZE-24, 0);
                                        recv(sock, buffer, BUFFER_SIZE, 0);
                                        free(temp);
                                        sem_post(&queue);
                                }

                        }
                        else if (buffer[0] == 'B'){ // change travel date
                                enqueue(id_index, 1);
                                sem_wait(&semaphores[id_index]);

                                // get ticket
                                char* modTicket = "Enter ticket to be modified: ";
                                send(sock, modTicket, BUFFER_SIZE, 0);
                                recv(sock, buffer, BUFFER_SIZE, 0);
                                char ticket[BUFFER_SIZE-24];
                                strcpy(ticket, buffer);

                                get_date(today, tomorrow);

                                char currDate[32];
                                char newTravelDate[32];
                                get_travel_date(ticket, currDate);

                                if (!strcmp(currDate, today)) strcpy(newTravelDate, tomorrow);
                                else if (!strcmp(currDate, tomorrow)) strcpy(newTravelDate, today);

                                char msgNewDate[BUFFER_SIZE-24];
                                sprintf(msgNewDate, "Your current travel date is %s, do you want to
change it to %s? (yes/no)", currDate, newTravelDate);

                                while (1)
                                {
                                        memset(buffer, 0, sizeof(buffer));
                                        send(sock, msgNewDate, BUFFER_SIZE-24, 0);
                                        recv(sock, buffer, BUFFER_SIZE, 0);
```

```
                                        if (!strcasecmp(buffer, "yes"))
                                        {
                                                Reservation* info =
(Reservation*)malloc(sizeof(Reservation)*SEAT_CAPACITY);
                                                int trav;
                                                inquiry(ticket, info, &trav);
                                                for (int i = 0; i < trav; i++){
                                                        strcpy((info+i)->travelDate, newTravelDate);
                                                }
                                                char seats[112];
                                                int seatCount;
                                                available_seats(newTravelDate, seats, &seatCount);
                                                if (seatCount <= trav){
                                                        char* full = "Cannot change seats the train is
full.\n";
                                                        send(sock, full, BUFFER_SIZE, 0);
                                                        break;
                                                }else{
                                                        cancel_reservation(ticket);
                                                        for (int i = 0; i < trav; i++){
                                                                int temporary1 = 0;
                                                                available_seats((info+i)->travelDate,
seats, &temporary1);

                                                                char ttmp[BUFFER_SIZE-24];
                                                                sprintf(ttmp, "Please choose your
seats for %s:\n%s", newTravelDate, seats);
                                                                send(sock, ttmp, BUFFER_SIZE-24, 0);


                                                                recv(sock, buffer, BUFFER_SIZE, 0);
                                                                strcpy((info+i)->seat, buffer);
                                                                add_travelers(name, (info+i), 1,
ticket);
                                                        }
                                                }
                                                char* msgDatChanged = "Date has been
changed.\nPress Enter to return to menu.";
                                                send(sock, msgDatChanged, BUFFER_SIZE-24, 0);
                                                recv(sock, buffer, BUFFER_SIZE, 0);
                                                free(info);
                                                break;
                                        }
                                        else if (!strcasecmp(buffer, "no")) break;
                                }
                                sem_post(&queue);
                        }
                        else if (buffer[0] == 'C'){ // remove travelers
                                enqueue(id_index, 1);
```

```
                                sem_wait(&semaphores[id_index]);

                                // get ticket
                                char* modTicket = "Enter ticket to be modified: ";
                                send(sock, modTicket, BUFFER_SIZE, 0);
                                recv(sock, buffer, BUFFER_SIZE, 0);
                                char ticket[BUFFER_SIZE-24];
                                strcpy(ticket, buffer);

                                // get number of travelers
                                Reservation* info =
(Reservation*)malloc(sizeof(Reservation)*SEAT_CAPACITY);
                                int trav;
                                inquiry(ticket, info, &trav);
                                //char* more = "yes";
                                while(1)
                                {
                                        char msg[BUFFER_SIZE-24] = "Which traveler would you like
to remove?\n";
                                        for (int i = 0; i < trav; i++){
                                                strcat(msg, (info+i)->customerName);
                                                strcat(msg, "\n");
                                        }
                                        send(sock, msg, BUFFER_SIZE, 0);
                                        recv(sock, buffer, BUFFER_SIZE, 0);
                                        char rmvName[BUFFER_SIZE-24];
                                        strcpy(rmvName, buffer);

                                        remove_traveler(ticket, rmvName);
                                        inquiry(ticket, info, &trav);

                                        char* again = "Remove another traveler? (yes/no) ";
                                        send(sock, again, BUFFER_SIZE-24, 0);
                                        recv(sock, buffer, BUFFER_SIZE, 0);
                                        if (strcasecmp(buffer, "yes")) break;
                                }

                                free(info);
                                sem_post(&queue);
                        }
                        else if (buffer[0] == 'D'){ // add travelers
                                enqueue(id_index, 1);
                                sem_wait(&semaphores[id_index]);

                                // get ticket
                                char* modTicket = "Enter ticket to be modified: ";
                                send(sock, modTicket, BUFFER_SIZE-24, 0);
```

```
                                recv(sock, buffer, BUFFER_SIZE, 0);
                                char ticket[BUFFER_SIZE-24];
                                strcpy(ticket, buffer);

                                // get number of travelers on ticket
                                Reservation* info =
(Reservation*)malloc(sizeof(Reservation)*SEAT_CAPACITY);
                                int trav;
                                inquiry(ticket, info, &trav);
                                // get travelDate
                                char travDate[32];
                                strcpy(travDate, info->travelDate);

                                int seatsAvail;
                                char seats[112];
                                //available_seats(atoi(travDate), seats, &seatsAvail);
                                available_seats(travDate, seats, &seatsAvail);

                                char* msgHowMany = "How many travelers to add: ";
                                send(sock, msgHowMany, BUFFER_SIZE-24, 0);
                                recv(sock, buffer, BUFFER_SIZE, 0);
                                int newTravelers = atoi(buffer);

                                Reservation* newReservations =
(Reservation*)malloc(sizeof(Reservation)*newTravelers);

                                if (seatsAvail < newTravelers){
                                        char* msgStupid = "Not enough available seats\nPress Enter
to return to menu.";
                                        send(sock, msgStupid, BUFFER_SIZE-24, 0);
                                        recv(sock, buffer, BUFFER_SIZE, 0);
                                }else{
                                        // Get info
                                        for (int i = 0; i < newTravelers; i++){
                                                strcpy((newReservations+i)->travelDate, travDate);
                                                // get name
                                                send(sock, getName, BUFFER_SIZE, 0);
                                                recv(sock, buffer, BUFFER_SIZE, 0);
                                                strcpy((newReservations+i)->customerName, buffer);
                                                // get dob
                                                send(sock, getDOB, BUFFER_SIZE, 0);
                                                recv(sock, buffer, BUFFER_SIZE, 0);
                                                strcpy((newReservations+i)->dob, buffer);
                                                // get gender
                                                send(sock, getGender, BUFFER_SIZE, 0);
                                                recv(sock, buffer, BUFFER_SIZE, 0);
                                                strcpy((newReservations+i)->gender, buffer);
```

```
                                                // get govID
                                                send(sock, getGovID, BUFFER_SIZE, 0);
                                                recv(sock, buffer, BUFFER_SIZE, 0);
                                                (newReservations+i)->govID = atoi(buffer);

                                                int temporary1 = 0;
                                                available_seats((newReservations+i)->travelDate,
seats, &temporary1);
                                                char ttmp[BUFFER_SIZE-24];
                                                sprintf(ttmp, "Please choose your seat %s:\n%s",
(newReservations+i)->customerName, seats);
                                                send(sock, ttmp, BUFFER_SIZE-24, 0);
                                                recv(sock, buffer, BUFFER_SIZE, 0);
                                                strcpy((newReservations+i)->seat, buffer);

                                                add_travelers(name, (newReservations+i), 1, ticket);
                                        }
                                }
                                free(info);
                                free(newReservations);
                                sem_post(&queue);
                        }
                        else{
                                char* stupid = "Please enter a valid option!\n";
                                send(sock, stupid, BUFFER_SIZE, 0);
                        }
                }
                else if (client_choice == 4) {
                        char* cancelTicket = "Enter ticket number to cancel: ";
                        send(sock, cancelTicket, BUFFER_SIZE, 0);
                        recv(sock, buffer, BUFFER_SIZE, 0);
                        cancel_reservation(buffer);
                        char* cancelTicket2 = "Your ticket has been canceled.\nPress Enter to return
to menu.";
                        send(sock, cancelTicket2, BUFFER_SIZE-24, 0);
                        recv(sock, buffer, BUFFER_SIZE, 0);
                }
                else if (client_choice == 5) break;
                // else send_error();
        }
        // send_exit_message();
        memset(message, 0, sizeof(message));
        send(sock, "Test", BUFFER_SIZE-24, 0);

        thread_in_use[id_index] = 0;
        close(sock);
        sem_post(&enforce_thread_limit);
```

```
	puts("thread exited");
	pthread_exit(NULL);
}

// likely not worth the effort
/*
void *detect_quit(void *arg)
{
	char input[200];
	char out[210];
	//while (run)
	//{
		//scanf("%s", input);

		fgets(input, 200, stdin);
		sprintf(out, "++%s++", input);
		for (int i = 0; i < 200; i++)
		{
			if (input[i] == '\n')
			{
				input[i] = '\0';
				break;
			}
		}
		if (!strncasecmp(input, "quit", 7))
		{
			run = 0;
			//pthread_cancel(queue_t);
			//for (int i = 0; i < THREAD_POOL_SIZE; i++)
			//{
			//	pthread_cancel(thread_pool[i]);
			//}
		}
		else puts(out);
	//}
	strcpy(out, "");
	sprintf(out, "=%d=", run);
	puts(out);
	puts("quit exit");
	pthread_exit(NULL);
}
*/

// forces the socket to recreate itself after admitting one client; this allows telling the client to try the next server when this one is full
// we cannot use the backlog of listen() for this purpose because it has a minimum of 16 (ie 16 clients could be waiting at once; ignoring the open servers)
```

```
int create_socket(int port)
{
	int server_fd, sock, opt = 1;
	struct sockaddr_in address;
	struct sockaddr_storage storage;
	socklen_t addrlen;

	if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
	{
		puts("Server socket creation failed");
		return -1;
	}
	//else puts("Server socket creation success");

	// this is necessary or the socket will fail to bind() on the second time through
	// from what I understand, by default it will wait a few minutes before fully closing the bound
socket
	if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt, sizeof(opt)))
  {
		puts("setsockopt");
		return -1;
  }

	address.sin_family = AF_INET;
	address.sin_port = htons(port);
	address.sin_addr.s_addr = INADDR_ANY;

	if ((bind(server_fd, (struct sockaddr *)&address, sizeof(address))) < 0)
	{
		puts("Server socket bind failed");
		return -1;
	}
	//else puts("Server socket bind success");

	if (listen(server_fd, 0) < 0) // might need to be 1 minimum; not sure (this is so we can have it
try a different server
	{
		puts("Server socket listen failed");
		return -1;
	}
	//else puts("Listening...");

	addrlen = sizeof(storage);
	if ((sock = accept(server_fd, (struct sockaddr *)&address, &addrlen)) < 0)
	{
		puts("Server socket accept failed");
		return -1;
```

```
	}
	close(server_fd);
	return sock;
}

int Server(char name, int port)
{
	srand(time(NULL));
	int thread_index = 0, i, sock;

	sem_init(&queue, 0, 1);
	sem_init(&enforce_thread_limit, 0, THREAD_POOL_SIZE);
	for (i = 0; i < THREAD_POOL_SIZE; i++) sem_init(&semaphores[i], 0, 1);

	pthread_create(&queue_t, NULL, queue_thread, NULL);

	while(1)//while (run)
	{
		sem_wait(&enforce_thread_limit);
		sock = create_socket(port);
		if (sock < 0) return 1;

		while (thread_in_use[thread_index])// && run)
		{
			thread_index++;
			if (thread_index >= THREAD_POOL_SIZE) thread_index = 0;
		}
		thread_in_use[thread_index] = 1;
		char temp[10];
		sprintf(temp, "%d,%d,%c", sock, thread_index, name);
		puts(temp);
		pthread_create(&thread_pool[thread_index], NULL, server_thread, (void *)temp);
	}
	sem_destroy(&queue);
	sem_destroy(&enforce_thread_limit);
	for (i = 0; i < THREAD_POOL_SIZE; i++) sem_destroy(&semaphores[i]);
	puts("server exited");
	return 0;
}
```

ALL

Main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <semaphore.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/wait.h>
#include <sys/mman.h>
#include <semaphore.h>
#include <pthread.h>
#include "Reservation.h"
#include "Backend.h"
#include "Server.h"

#define PORT 8019
#define SERVER_COUNT 5

int main(){
   // Create semaphores and shared memory object
   sem_t *file_write;
   sem_t *file_read;
   int shm_fd;
   int *ptrReaders;
   init_sync(file_write, file_read, shm_fd, ptrReaders);

   // create a bunch of processes to read at the same time
   char name = 'A';
   int parentid = getpid();
   for (int i = 0; i < SERVER_COUNT; i++){
      if (fork() == 0){
         Server(name + i, PORT + i);
         return 0;
      }
   }

   for(int j = 0; j < 5; j++){
      wait(NULL);
   }
```

```
    // desync
    desync(file_write, file_read, shm_fd, ptrReaders);

    return 0;
}
```