```c
// GROUP B
// Nathan Baker
// nathan.t.baker@okstate.edu

#include "header.h"
#define SUMMARY 0 // for semaphore referencing

pthread_mutex_t lock;

int get_customer_info(int socket, struct clientInformation* c) {
    // repeatedly send prompts and scan customer responses.
    // fill clientInformation structure.
    char m[1000];
    strcpy(m,"0Please enter your full name: ");
    send(socket, &m, sizeof(m), MSG_NOSIGNAL);
    read(socket, &m, sizeof(m));
    sscanf(m,"%50[^\n]",c->ClientName);
    printf("%s\n",c->ClientName);
    strcpy(m,"0Please enter your date of birth [MM/DD/YYYY]: ");
    send(socket, &m, sizeof(m), MSG_NOSIGNAL);
    read(socket, &m, sizeof(m));
    sscanf(m,"%50[^\n]",c->DateOfBirth);
    printf("%s\n",c->DateOfBirth);
    strcpy(m,"0Please enter your gender [M, F, Other]: ");
    send(socket, &m, sizeof(m), MSG_NOSIGNAL);
    read(socket, &m, sizeof(m));
    sscanf(m,"%10[^\n]",c->Gender);
    printf("%s\n",c->Gender);
    strcpy(m,"0Please enter your GovernmentID number: ");
    send(socket, &m, sizeof(m), MSG_NOSIGNAL);
    read(socket, &m, sizeof(m));
    sscanf(m,"%d",&c->GovernmentID);
    printf("%d\n",c->GovernmentID);
    strcpy(m,"0Please enter your desired date of travel [MM/DD/YYYY]: ");
    send(socket, &m, sizeof(m), MSG_NOSIGNAL);
    read(socket, &m, sizeof(m));
    sscanf(m,"%50[^\n]",c->DateOfTravel);
    printf("%s\n",c->DateOfTravel);
    strcpy(m,"0Please enter the number of travelers: ");
    send(socket, &m, sizeof(m), MSG_NOSIGNAL);
    read(socket, &m, sizeof(m));
    sscanf(m,"%d",&c->NumberOfTravelers);
    printf("%d\n",c->NumberOfTravelers);
    return 0;
}

int get_customer_ticket(int socket, struct clientInformation* c) {
    // ask customer for their ticket number and scan the response into the struct.
    char m[1000];
    int ticket;
    strcpy(m,"0Please enter your ticket number: ");
    send(socket, &m, sizeof(m), MSG_NOSIGNAL);
    read(socket, &m, sizeof(m));
    sscanf(m,"%d",&c->ticket);
    printf("%d\n",c->ticket);
    return 0;
}

int change_read_count(int offset) {
    // file used to share readcount variable between servers.
    // update readcount by offset.
    FILE *fp;
    fp = fopen ("summary_read_count.txt", "r");
    int num;
    if (fp == NULL) num = 0;
    else {
        fscanf(fp,"%d", &num);
        fclose(fp);
    }
    if (offset == 0) return num;
    fp = fopen ("summary_read_count.txt", "r");
    fprintf(fp,"%d",num+offset);
    fclose(fp);
    return num+offset; // return new readcount
}

int verify_enough_seats(int socket, int train, struct clientInformation* c) {
    // compare seats in train file to seats requested.
    // at this point thread already has access to train semaphore.
    int available = seatChecker(train);
    if ((c->NumberOfTravelers) > available) { // if not enough seats
        char m[1000];
        snprintf(m,1000,"1Sorry, there are only %d seats availble for the selected date.\nReservation cancelled.\n",available);
        send(socket, &m, sizeof(m), MSG_NOSIGNAL);
        signal_write(train);
        return -1; // send customer back to menu.
    }
    return 0;
}

int confirm_purchase(int socket, int train, struct clientInformation* c) {
    // ask user for confirmation.
    char m[1000];
    snprintf(m,1000,"0\nDo you want to make reservation (yes/no): ");
    send(socket, &m, sizeof(m), MSG_NOSIGNAL);
    read(socket, &m, sizeof(m));
    if (strcmp(m,"yes") == 0) return 0; // proceed.
    else {
        snprintf(m,1000,"1Reservation cancelled.\n");
        send(socket, &m, sizeof(m), MSG_NOSIGNAL);
        signal_write(train); // release semaphore.
        return -1; // send customer back to menu.
    }
}

int confirm_cancel(int socket, struct clientInformation* c) {
    // ask user for confirmation.
    char m[1000];
```

```
109     snprintf(m,1000,"0\nAre you sure you want to cancel your reservation (yes/no): ");
110     send(socket, &m, sizeof(m), MSG_NOSIGNAL);
111     read(socket, &m, sizeof(m));
112     if (strcmp(m,"yes") == 0) return 0; // proceed.
113     else {
114         return -1; // send customer back to menu.
115     }
116 }
117
118 int confirm_modify(int socket, struct clientInformation* c) {
119     // inform user about modification constraints and ask for confirmation.
120     char m[1000];
121     snprintf(m,1000,"0\nReservation modifications include reducing the number of seats or changing seat choice.\nIf you want to reserve additional seats you must make a new reservation.\nAre you sure you
122     send(socket, &m, sizeof(m), MSG_NOSIGNAL);
123     read(socket, &m, sizeof(m));
124     if (strcmp(m,"yes") == 0) {
125         int n;
126         snprintf(m,1000,"0What is your new desired number of travelers [up to %d]? ",c->NumberOfTravelers);
127         send(socket, &m, sizeof(m), MSG_NOSIGNAL);
128         read(socket, &m, sizeof(m));
129         sscanf(m,"%d",&n); // read customer response
130         if (n > c->NumberOfTravelers || n < 1) { // ensure that customer requested a valid number of seats.
131             snprintf(m,1000,"1Invalid selection. Modification cancelled.\n");
132             send(socket, &m, sizeof(m), MSG_NOSIGNAL);
133             read(socket, &m, sizeof(m));
134             return -1; // send customer back to menu.
135         }
136         return n; // return the new NumberOfTravelers
137     } else {
138         return -1; // send customer back to menu.
139     }
140 }
141
142 void send_available_seats(int socket, int train, struct clientInformation* c) {
143     char output[100];
144     show_available(train, output); // populates output with train string.
145     char m[1000];
146     snprintf(m,1000,"0\nPlease choose %d of the following available seats [only spaces between each seat]:\n%s\n",c->NumberOfTravelers,output);
147     send(socket, &m, sizeof(m), MSG_NOSIGNAL); // send message asking custoemr to pick from available seats.
148 }
149
150 void show_available(int trainNum, char* output) {
151     FILE *fp ;
152     char c;
153
154     printf("Opening the file train in read mode \n");
155     if (trainNum == 1) {
156         fp = fopen ("train1.txt","r"); // opening an existing file
157     } else if (trainNum == 2) {
158         fp = fopen ("train2.txt", "r"); // opening an existing file
159     }
160     if (fp == NULL) {
161       printf ("Could not open file train \n");
162       return;
163     }
164     printf("Reading train file.\n");
165     int count = 0;
166     int char_index = 0;
167     while (1) {
168         c = fgetc (fp); // read one character = one seat
169         if (c == '0') { // available
170             output[char_index++] = 'A'+(count / 5);
171             output[char_index++] = '0'+(count % 5 + 1); // create seat number via pointer arithmetic.
172         } else if (c == '1') { // unavailable
173             output[char_index++] = '-';
174             output[char_index++] = '-';
175         } else {
176             output[char_index++] = '\0'; // end
177             break;
178         }
179         output[char_index++] = ' ';
180         count++;
181         if ((count % 5) == 0) output[char_index++] = '\n'; // new row
182     }
183     printf("Closing the file train \n");
184     fclose (fp); // Closing the file
185     return;
186 }
187
188 int check_seat(int train, int row, int column) {
189     // check if particular seat in train is available.
190     // thread already has semaphore.
191     if (!(row < 5 && row >= 0 && column < 5 && column >= 0)) return -1; // if invalid seat.
192     FILE *fp;
193     if (train == 1) {
194         fp = fopen ("train1.txt", "r");
195     } else if (train == 2) {
196         fp = fopen ("train2.txt", "r");
197     }
198     int index = row*5 + column; // 2d -> 1d address
199     printf("%d, %d, %d\n",row, column, index);
200     char c;
201     for (int i=0; i<index; i++) { // loop until seat in question.
202         c = fgetc(fp);
203     }
204     c = fgetc(fp);
205     fclose(fp);
206     if (c == '0') return 0; // available
207     else return -1; // unavailable
208 }
209
210 int write_seat(int train, int row, int column, int update) {
211     // updates individual seat status in train file.
212     // thread already holds semaphore.
213     FILE *fp;
214     if (train == 1) {
215         fp = fopen ("train1.txt", "r+");
216     } else if (train == 2) {
217         fp = fopen ("train2.txt", "r+");
```

```
    }
    int index = row*5 + column; // 2d -> 1d address.
    fseek(fp, index, SEEK_SET); // go to character index.
    if (update == 1) fputc('1',fp);
    else if (update == 0) fputc('0',fp); // write new value.
    fclose(fp);
    return 0;
}

int verify_selection(int socket, int train, struct clientInformation* c, char* m) {
    // thread already holds semaphore.
    char n[1000];
    strcpy(n,m); // make copy of seat selection string to allow memmove without losing original data.
    printf("%s\n",n);
    char seat[3];
    int offset; // to hold number of scanned bytes
    for (int i=0; i<c->NumberOfTravelers; i++) {
        if (sscanf(n," %2c%n",seat,&offset) != 1) { // could not scan two non-space characters
            printf("seat verification failed.\n");
            signal_write(train);
            char msg[1000];
            strcpy(msg,"1\nError: not enough seats were selected. Reservation cancelled.\n");
            send(socket, &msg, sizeof(msg), MSG_NOSIGNAL);
            return -1; // send customer back to menu.
        }
        memmove(n, n+offset, 1000); // move string head pointer by number of bytes read.
        int row = seat[0] - 65; // 'A' -> 0
        int column = seat[1] - 49; // '1' -> 0
        if (check_seat(train,row,column) == -1) { // seat not available.
            printf("seat verification failed.\n");
            signal_write(train);
            char msg[1000];
            strcpy(msg,"1\nError: one or more of the selected seats is not availble. Reservation cancelled.\n");
            send(socket, &msg, sizeof(msg), MSG_NOSIGNAL);
            return -1; // send customer back to menu.
        }
    }
    if (sscanf(n," %2c%n",seat,&offset) == 1) { // extra seat was requested.
        printf("seat verification failed.\n");
        signal_write(train);
        char msg[1000];
        strcpy(msg,"1\nError: too many seats were selected. Reservation cancelled.\n");
        send(socket, &msg, sizeof(msg), MSG_NOSIGNAL);
        return -1; // send customer back to menu.
    }
    strcpy(c->seats, m); // fill struct field with seat selection.
    return 0;
}

int add_to_train(int train, struct clientInformation* c, char* m) {
    // reserve seats in train file.
    // thread already holds semaphore.
    char output[100];
    show_available(train, output); // for server-side output only
    printf("%s\n",output);
    char n[1000];
    strcpy(n,m); // to move string pointer without losing original data.
    char seat[3];
    int offset;
    for (int i=0; i<c->NumberOfTravelers; i++) { // loop through selected seats..
        sscanf(n," %2c%n",seat,&offset); // scan seat number.
        memmove(n, n+offset, 1000); // move forward by number of scanned bytes.
        int row = seat[0] - 65; // 'A' -> 0
        int column = seat[1] - 49; // '1' -> 0
        write_seat(train,row,column,1); // set seat to unavailable
    }
    show_available(train, output); // for server-side output only
    printf("%s\n",output);
    return 0;
}

int remove_from_train(struct clientInformation* c, int train) {
    // open seats in train file.
    // thread already holds semaphore.
    char output[100];
    show_available(train, output); // for server-side output only
    printf("%s\n",output);
    char n[100];
    strcpy(n,c->seats);
    char seat[3];
    int offset;
    printf("num: %d\n seats: %s\n",c->NumberOfTravelers,n);
    for (int i=0; i<c->NumberOfTravelers; i++) {
        sscanf(n," %2c%n",seat,&offset); // scan seat number.
        memmove(n, n+offset, 100); // move forward by number of scanned bytes.
        int row = seat[0] - 65; // 'A' -> 0
        int column = seat[1] - 49; // '1' -> 0
        write_seat(train,row,column,0); // set seat to available.
    }
    show_available(train, output); // for server-side output only
    printf("%s\n",output);
    return 0;
}

int get_train(struct clientInformation* c) {
    // determine which train to interact with based on provided date.
    char date[50];
    int train;
    GetTodayDate(date);
    printf("customer date: %s\n",c->DateOfTravel);
    printf("%s\n",date);
    if (strcmp(c->DateOfTravel,date) == 0) train = 1; // train 1 is for today.
    else {
        GetTomorrowDate(date);
        printf("%s\n",date);
        if (strcmp(c->DateOfTravel,date) == 0) train = 2; // train 2 is for tomorrow.
        else train = -1; // invalid date.
    }
    printf("train %d\n",train);
```

```
    printf("train %d\n",train);
    return train;
}

int signal_read(int train) { // separate function for signaling semaphore.
    if (train != 0) return 0;
    char sem_name[25];
    strcpy(sem_name,"/summary_read"); // semaphore only for adjust readcount.
    sem_t* sem;
    if ((sem = sem_open(sem_name, O_RDWR)) == SEM_FAILED) {
        printf("failed to open read semaphore for summary.\nerror number:%d",errno);
        exit(1);
    }
    sem_post(sem);
    return 0;
}

int wait_read(int train) { // separate function for waiting for semaphore.
    if (train > 0) return 0;
    char sem_name[25];
    strcpy(sem_name,"/summary_read"); // semaphore only for adjust readcount.
    sem_t* sem;
    if ((sem = sem_open(sem_name, O_RDWR)) == SEM_FAILED) {
        printf("failed to open read semaphore for summary.\nerror number:%d",errno);
        exit(1);
    }
    sem_wait(sem);
    return 0;
}

int signal_write(int train) { // separate function for signaling semaphore.
    char sem_name[25];
    if (train > 0) snprintf(sem_name,25,"/train%d",train);
    else if (train == SUMMARY) strcpy(sem_name,"/summary_write");
    sem_t* sem;
    if ((sem = sem_open(sem_name, O_RDWR)) == SEM_FAILED) {
        printf("failed to open write semaphore for train%d.\nerror numububer:%d",train,errno);
        exit(1);
    }
    sem_post(sem);
    return 0;
}

int wait_write(int train) { // separate function for waiting for semaphore.
    char sem_name[25];
    if (train > 0) snprintf(sem_name,25,"/train%d",train);
    else if (train == SUMMARY) strcpy(sem_name,"/summary_write");
    sem_t* sem;
    if ((sem = sem_open(sem_name, O_RDWR)) == SEM_FAILED) {
        printf("failed to open write semaphore for train%d.\nerror number:%d",train,errno);
        exit(1);
    }
    sem_wait(sem);
    return 0;
}

int check_thread_permission(int id, int train, int seats, int* seats_for_thread) {
    seats_for_thread[id+(train-1)*NUM_THREADS] = seats; // post requested seats to the shared array.
    int largest;
    if (seats <= 0) return -1;
    if (train <= 0) return -1;
    while (1) {
        wait_write(train); // wait for current thread to finish with train file.
        largest = 1;
        for (int i=0; i<NUM_THREADS; i++) { // see if this thread has largest number of requested seats.
            if (seats_for_thread[id+(train-1)*NUM_THREADS] < seats_for_thread[i+(train-1)*NUM_THREADS]) {
                largest = 0;
                break;
            }
        }
        if (largest == 1) return 0; // proceed.
        signal_write(train); // release semaphore if thread is not chosen.
        sleep(1); // try again in 1 second.
    }
}

int serve_customer(int socket, int t_id, int s_id, int* seats_for_thread) {
    const struct clientInformation empty_struct;
    struct clientInformation c;
    char m[1000];
    int first = 1;
    while (1) {
        c = empty_struct; // reset customer struct when back to menu.
        c.server = s_id; // set server id.
        if (first) { // initial greeting.
            snprintf(m,1000,"0Hello! My name is THREAD-%d, How may I assist you today?\n\t1. Make a reservation.\n\t2. Inquiry about a ticket.\n\t3. Modify the reservation.\n\t4. Cancel the reservation.\n\t5. E
            first = 0;
        } else { // back to menu message.
            strcpy(m,"0\nIs there anything else I can help you with today?\n\t1. Make a reservation.\n\t2. Inquiry about a ticket.\n\t3. Modify the reservation.\n\t4. Cancel the reservation.\n\t5. Exit the program.\n")
        }
        send(socket, &m, sizeof(m), MSG_NOSIGNAL);
        read(socket, &m, sizeof(m));
        sscanf(m,"%d",&c.MenuOption); // scan menu option from customer.
        printf("%d\n",c.MenuOption);
        if (c.MenuOption == 5) {
            strcpy(m,"2Exiting...Thank you and have a good day!\n"); // client will terminate socket.
            send(socket, &m, sizeof(m), MSG_NOSIGNAL);
            return 0; // thread frees up.
        }
        if (c.MenuOption == 1) { // make reservation
            if (get_customer_info(socket,&c) == -1) continue; // fill clientInformation struct.
            char date[50];
            int train;
            GetTodayDate(date);
            printf("%s\n",date);
            if (strcmp(c.DateOfTravel,date) == 0) train = 1; // train 1 = today.
            else {
                GetTomorrowDate(date);
                if (strcmp(c.DateOfTravel,date) == 0) train = 2; // train 2 = tomorrow.
                else train = -1;
```

```
            else train = -1;
        }
        if (train == -1) {
            strcpy(m,"1Sorry, there is no train available for the selected date.\nReservation cancelled.\n");
            send(socket, &m, sizeof(m), MSG_NOSIGNAL);
            return -1;
        }
        strcpy(m,"1Please wait...\n");
        send(socket, &m, sizeof(m), MSG_NOSIGNAL);
        strcpy(m,"");
        if (check_thread_permission(t_id,train, c.NumberOfTravelers,seats_for_thread) == -1) continue; // thread priority
        // at this point the thread has the train semaphore.
        if (verify_enough_seats(socket, train, &c) == -1) continue; // if failed, return to menu and release semaphore.
        if (confirm_purchase(socket, train, &c) == -1) continue; // if failed, return to menu and release semaphore.
        send_available_seats(socket, train, &c);
        read(socket, &m, sizeof(m)); // read seat selection.
        if (verify_selection(socket, train, &c, m) == -1) continue; // if failed, return to menu and release semaphore.
        add_to_train(train, &c, m); // update train file.
        signal_write(train); // release train semaphore.
        wait_write(SUMMARY);
        addCustomer(&c,1); // update summary file after waiting for access.
        signal_write(SUMMARY);
        snprintf(m,1000,"1Reservation confirmed! Your ticket number is %d.\n",c.ticket);
        send(socket, &m, sizeof(m), MSG_NOSIGNAL);
        continue; // return to menu.
      }
      if (c.MenuOption == 2) { // inquiry
        if (get_customer_ticket(socket,&c) == -1) continue; // ask for ticket
        char results[500];
        // procedure for allowing multiple readers or one writer.
        wait_read(SUMMARY);
        if (change_read_count(1) == 1) wait_write(SUMMARY);
        signal_read(SUMMARY);
        printCustomerInfo(&c,results); // read customer info from summary file. populates results string.
        wait_read(SUMMARY);
        if (change_read_count(-1) == 0) signal_write(SUMMARY);
        signal_read(SUMMARY);
        snprintf(m,1000,"1%s\n",results); // print inquiry results.
        send(socket, &m, sizeof(m), MSG_NOSIGNAL);
        continue; // return to menu.
      }
      if (c.MenuOption == 3) { // modify.
        if (get_customer_ticket(socket,&c) == -1) continue; // ask for ticket number.
        // procedure for allowing multiple readers or one writer.
        wait_read(SUMMARY);
        if (change_read_count(1) == 1) wait_write(SUMMARY);
        signal_read(SUMMARY);
        createCustomer(&c); // read customer info from summary file. populates struct.
        wait_read(SUMMARY);
        if (change_read_count(-1) == 0) signal_write(SUMMARY);
        signal_read(SUMMARY);
        int train = get_train(&c); // determine which train was used.
        char original_seats[100];
        strcpy(original_seats,c.seats);
        if (train == -1) {
            snprintf(m,1000,"1The date for this train has passed, cannot modify reservation.\n");
            send(socket, &m, sizeof(m), MSG_NOSIGNAL);
            continue; // return to menu.
        }
        int new_number = confirm_modify(socket,&c); // confirm modify. get new number of seats.
        if (new_number == -1) continue;
        wait_write(train); // wait for write access.
        remove_from_train(&c,train); // remove previous reservation from train.
        c.NumberOfTravelers = new_number;
        send_available_seats(socket, train, &c); // send available seats and ask for input.
        read(socket, &m, sizeof(m)); // read new selection.
        if (verify_selection(socket, train, &c, m) == -1) continue; // if fails, return to menu and release semaphore.
        add_to_train(train, &c, c.seats); // add updated reservation.
        signal_write(train);
        snprintf(m,1000,"1Reservation modified.\n");
        snprintf(c.modified,200,"Reservation modified by server %d. Original seats: [%s]",s_id,original_seats); // add note
        wait_write(SUMMARY);
        changeOldCustomer(&c); // update summary file.
        signal_write(SUMMARY);
        send(socket, &m, sizeof(m), MSG_NOSIGNAL);
        continue; // return to menu.
      }
      if (c.MenuOption == 4) { // delete
        if (get_customer_ticket(socket,&c) == -1) continue; // get ticket number
        // procedure for allowing multiple readers or one writer.
        wait_read(SUMMARY);
        if (change_read_count(1) == 1) wait_write(SUMMARY);
        signal_read(SUMMARY);
        createCustomer(&c); // read customer info from summary file. populates struct.
        wait_read(SUMMARY);
        if (change_read_count(-1) == 0) signal_write(SUMMARY);
        signal_read(SUMMARY);
        int train = get_train(&c); // determine which train was used.
        if (train == -1) {
            snprintf(m,1000,"1The date for this train has passed, no need to cancel reservation.\n");
            send(socket, &m, sizeof(m), MSG_NOSIGNAL);
            continue; // return to menu.
        }
        if (confirm_cancel(socket,&c) == -1) continue; // confirm cancel.
        wait_write(train);
        remove_from_train(&c,train); // remove reservation from train after acquiring semaphore.
        signal_write(train);
        wait_write(SUMMARY);
        deleteCustomer(&c); // remove line from summary file after acquiring semaphore.
        signal_write(SUMMARY);
        snprintf(m,1000,"1Reservation cancelled.\n");
        send(socket, &m, sizeof(m), MSG_NOSIGNAL);
        continue; // return to menu.
      }
      break; // end connection if invalid option provided.
    }
    return 0;
}

int thread_loop(void* arg) {
```

```c
int thread_loop(void* args) {
    struct customer_queue* q = (struct customer_queue*) args;
    int id;
    for (int i=0; i<NUM_THREADS; i++) { // get thread ID
        if (q->threads[i] == pthread_self()) {
            id = i;
            break;
        }
    }
    while(1) {
        q->seats_for_thread[id] = 0;
        q->seats_for_thread[id+NUM_THREADS] = 0; // set requested seats to 0 in shared array.
        int my_customer = -1;
        pthread_mutex_lock(&lock); // customer queue = critical section.
        if (q->waiting > 0) { // if customer is waiting
            my_customer = q->sockets[q->first]; // get customer socket descriptor
            q->sockets[q->first] = 0; // remove socket descriptor
            q->first = q->first + 1; // move head index
            q->waiting = q->waiting -1; // decrease waiting number
        }
        pthread_mutex_unlock(&lock);
        if (my_customer >= 0) serve_customer(my_customer,id,q->port,q->seats_for_thread); // serve customer if exists
    }
}

int create_socket(int port, struct sockaddr_in* address) {
    // standard procedure for creating server socket with specified port number
    int server_fd;
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket failed");
        exit(1);
    } else {
        printf("Server socket created\n");
    }
    address->sin_family = AF_INET;
    address->sin_addr.s_addr = INADDR_ANY;
    address->sin_port = htons(8000+port);
    if (bind(server_fd, (struct sockaddr*) address, sizeof(*address)) < 0) {
        perror("bind failed");
        exit(1);
    } else {
        printf("Server socket bound\n");
    }
    if (listen(server_fd, 2) < 0) {
        perror("listen");
        exit(1);
    } else {
        printf("Server socket is listening\n");
    }
    return server_fd;
}

int initialize_semaphores_threads(struct customer_queue* q, int reset_semaphores) {
    q->first = q->waiting = 0;
    pthread_mutex_init(&lock, NULL);
    for (int i=0; i<NUM_THREADS; i++) { // kickoff threads
        if (pthread_create(&(q->threads[i]), NULL, (void *)&thread_loop, (void *)q) != 0) {
            perror("Failed to create thread");
        }
    }
    if (reset_semaphores == 1) { // reset semaphores if specified in command line arguments.
        sem_unlink("/train1");
        sem_unlink("/train2");
        sem_unlink("/summary_read");
        sem_unlink("/summary_write");
    }
    // initialize semaphores with value = 1 if not already exist
    if ((sem_open("/train1", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR, 1)) == SEM_FAILED) {
        printf("failed to open semaphore for train0.\nerror number:%d",errno);
        exit(1);
    }
    if ((sem_open("/train2", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR, 1)) == SEM_FAILED) {
        printf("failed to open semaphore for train1.\nerror number:%d",errno);
        exit(1);
    }
    if ((sem_open("/summary_read", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR, 1)) == SEM_FAILED) {
        printf("failed to open semaphore for train1.\nerror number:%d",errno);
        exit(1);
    }
    if ((sem_open("/summary_write", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR, 1)) == SEM_FAILED) {
        printf("failed to open semaphore for train1.\nerror number:%d",errno);
        exit(1);
    }
}

int server_loop(int server_fd, int port, struct sockaddr_in* address, struct customer_queue* q) {
    while(1) { // wait for new connections.
        int addrlen = sizeof(*address);
        int new_socket;
        if ((new_socket = accept(server_fd, (struct sockaddr*) address,(socklen_t*) &addrlen)) < 0) {
            perror("Could not accept connection.");
            exit(1);
        }
        printf("new socket accepted.\n");
        char m[1000];
        pthread_mutex_lock(&lock);
        if (q->waiting < 100) {
            q->sockets[q->first+q->waiting] = new_socket; // add new connection to queue.
            q->waiting = q->waiting + 1;
            snprintf(m,1000,"0Thank you for choosing Server %d. One of our threads will be with you shortly...\n",port);
            send(new_socket, &m, sizeof(m), 0);
        } else { // if not room in queue.
            snprintf(m,1000,"2Sorry, There are already %d customers waiting to be served. Please try again later.\n",100);
            send(new_socket, &m, sizeof(m), 0);
        }
        pthread_mutex_unlock(&lock);
    }
    return 0;
}
```

```
653
654 int main(int argc, char const *argv[]) {
655     int port = 0;
656     if (argc > 1) port = atoi(argv[1]); // read server id from args
657     if (port > 4 || port < 1) {
658         printf("no valid server index provided.\n");
659         exit(1);
660     }
661     int reset = 0;
662     if (argc > 2 && strcmp(argv[2],"-r") == 0) reset = 1; // read reset specifier. [if recovering from crash]
663     struct sockaddr_in address;
664     int server_fd = create_socket(port, &address);
665     struct customer_queue q;
666     q.port = port; // to give server id to threads
667     initialize_semaphores_threads(&q, reset);
668     server_loop(server_fd,port,&address,&q);
669     return 0;
670 }
```