EE 463 (Operating Systems)

Section: C3

Semester: Winter 2023

Multithreaded C Program for Patient Room Access Control In A Hospital Using Pthread

Name	ID#
Hayan Al-Machnouk	1945954

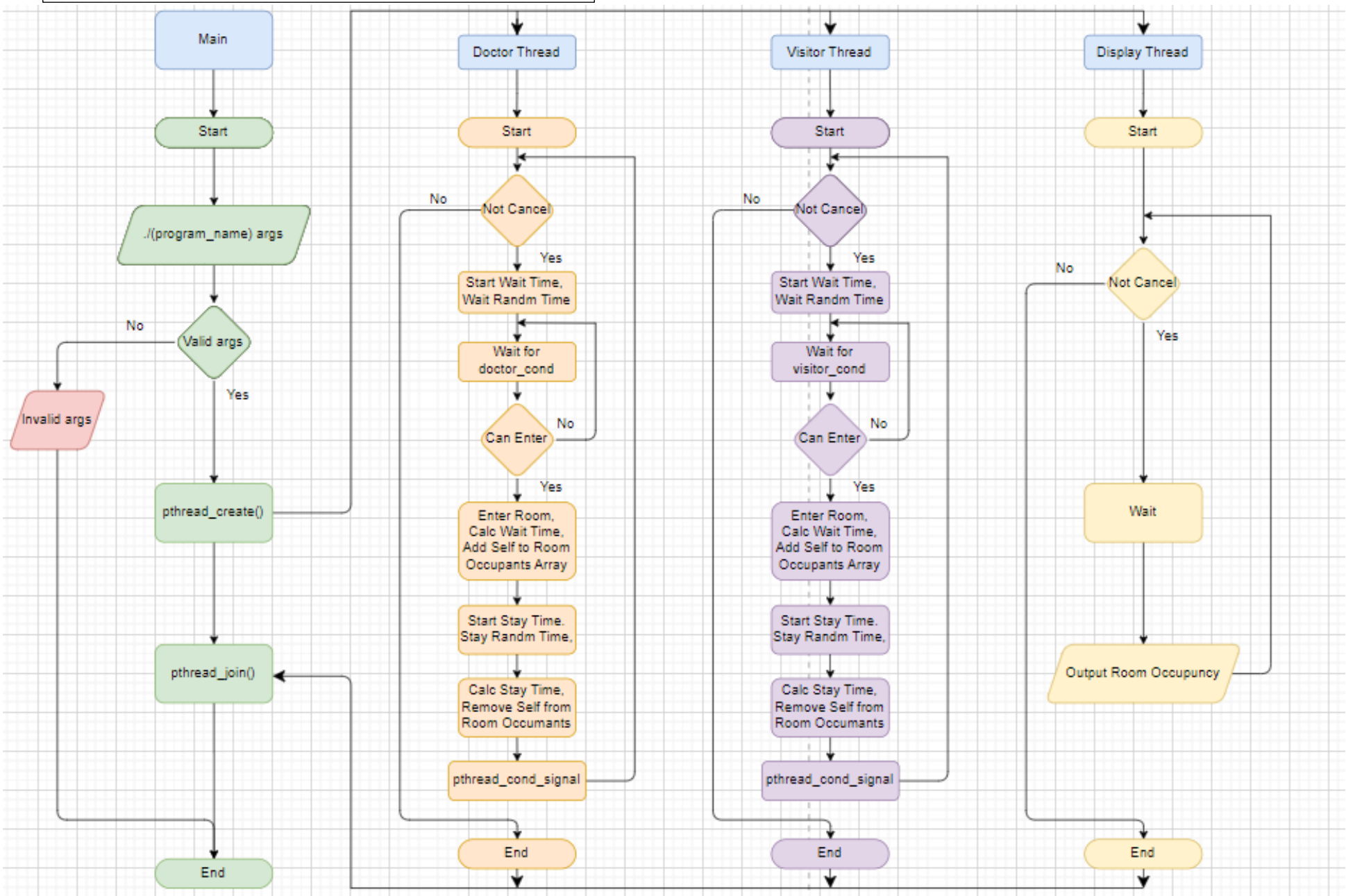
Course Teacher: Dr. Abdulghani M. Al-Qasimi

Department of Electrical and Computer Engineering

King Abdulaziz University, Jeddah, KSA

Contents

Flow Chart:


Main
Start
./(program_name) args
Valid args
No
Yes
Invalid args
pthread_create()
pthread_join()
End
Doctor Thread
Start
Not Cancel
No
Yes
Start Wait Time, Wait Randm Time
Wait for doctor_cond
Can Enter
No
Yes
Enter Room, Calc Wait Time, Add Self to Room Occupants Array
Start Stay Time. Stay Randm Time,
Calc Stay Time, Remove Self from Room Occumants
pthread_cond_signal
End
Visitor Thread
Start
Not Cancel
No
Yes
Start Wait Time, Wait Randm Time
Wait for visitor_cond
Can Enter
No
Yes
Enter Room, Calc Wait Time, Add Self to Room Occupants Array
Start Stay Time. Stay Randm Time,
Calc Stay Time, Remove Self from Room Occumants
pthread_cond_signal
End
Display Thread
Start
Not Cancel
No
Yes
Wait
Output Room Occupuncy
End

Pseudocode:

1. Initialize variables: doctor_waiting, visitor_waiting, room_doctors, room_visitors, num_doctors, num_doctors_arg, num_visitors, num_visitors_arg
2. Initialize mutex and condition variables: room_lock, doctor_cond, visitor_cond
3. Main function:
    a. Get the number of doctor and visitor threads from command line arguments or assigne default random
    b. Create an array of pthreads for doctors and visitors
    c. Initialize the array with the appropriate number of doctor and visitor threads
4. Create a function "doctor_thread"
    a. Start an infinite loop
    b. Wait for a random period of time before trying to enter the room
    c. Acquire the lock
    d. Increment the number of doctor_waiting
    e. Wait until no one in the room using a while loop and pthread_cond_wait
    f. Add the doctor's id to the array of doctors in the room
    g. Release the lock
    h. Wait for a random period of time before leaving the room
    i. Acquire the lock
    j. Remove the doctor's id from the array of doctors in the room
    k. If there are at least 3 visitor waiting, signal the visitor_cond
    l. Else if there are more than 0 doctor waiting, signal the doctor_cond
    m. Else signal visitor_cond
    n. Release the lock
5. Create a function "visitor_thread"
    a. Start an infinite loop
    b. Wait for a random period of time before trying to enter the room
    c. Acquire the lock
    d. Increment the number of visitor_waiting
    e. Wait until less than 3 people inside room using while loop and pthread_cond_wait
    f. A the visitor's id to the array of visitors in the room
    g. Release the lock
    h. Wait for a random period of time before leaving the room
    i. Acquire the lock
    j. Remove the visitor's id from the array of visitors in the room
    k. If there are at least 1 doctor waiting, signal the doctor_cond
    l. Else if there are more than 0 visitor waiting, signal the visitor_cond
    m. Else signal doctor_cond
    n. Release the lock

Program:

The implementation of each method will be described and shown bellow.

Main:

function initializes random number of doctors and visitors, creates the display thread, and creates the doctor and visitor threads. It then waits for the display thread to finish, sends cancellation signals to all other threads, waits for them to finish, and releases all resources.

```
int main(int argc, char* argv[]) {

    // initialize random values
    srand(time(0));
    int num_doctors = rand() % MAX_DOCTORS_THREADS + 1;
    int num_visitors = rand() % MAX_VISITORS_THREADS + 1;

    // Handle command line arguments
    if (argc == 1) {
        ; // nop
    }
    if (argc < 1 || argc > 3) {
        printf("Invalid Arguments");
        return 1;
    }
    if (argc >= 2) {
        num_doctors_arg = atoi(argv[1]);
        if (!(num_doctors_arg < 1 || num_doctors_arg > MAX_DOCTORS_THREADS)) {···
        } else {
            printf("Invalid number of Doctors. Using random value %d\n", num_visitors);
        }
    }
    if (argc == 3) {
        num_visitors_arg = atoi(argv[2]);
        if (!(num_visitors_arg < 1 || num_visitors_arg > MAX_VISITORS_THREADS)) {
            num_visitors = num_visitors_arg;
        } else {
            printf("Invalid number of Visitors. Using random value %d\n", num_visitors);
        }
    }
```

```
printf("Display --> Created %d Doctors, and %d Visitors.\n", num_doctors, num_visitors);

pthread_mutex_init(&room_lock, NULL);
pthread_cond_init(&doctor_cond, NULL);
pthread_cond_init(&visitor_cond, NULL);

// create display thread
pthread_t display_tid;
pthread_create(&display_tid, NULL, display_thread, NULL);

// create doctors threads
pthread_t doctor_tids[num_doctors];
for (int i = 0; i < num_doctors; i++) {
    int* id = malloc(sizeof(int));
    *id = i;
    pthread_create(&doctor_tids[i], NULL, doctor_thread, (void*) id);
}

// create visitors threads
pthread_t visitor_tids[num_visitors];
for (int i = 0; i < num_visitors; i++) {
    int* id = malloc(sizeof(int));
    *id = i;
    pthread_create(&visitor_tids[i], NULL, visitor_thread, (void*) id);
}
```

```
    // Cleanup and exit
    pthread_join(display_tid, NULL);

    for (int i = 0; i < num_doctors; i++) {
        pthread_cancel(doctor_tids[i]);
    }

    for (int i = 0; i < num_visitors; i++) {
        pthread_cancel(visitor_tids[i]);
    }

    for (int i = 0; i < num_doctors; i++) {
        pthread_join(doctor_tids[i], NULL);
    }

    for (int i = 0; i < num_visitors; i++) {
        pthread_join(visitor_tids[i], NULL);
    }

    pthread_mutex_destroy(&room_lock);
    pthread_cond_destroy(&doctor_cond);
    pthread_cond_destroy(&visitor_cond);

    return 0;
}
```

Display Thread:

The function waits for a random period of time before printing the current status of the room. It acquires the room lock, prints the number of doctors and visitors in the room, and releases the lock.

```
/* ============================================================================
void* display_thread(void* arg) {
    while (1) {
        // Wait for a period of time before printing the room status
        usleep(400000);

        // Acquire the lock
        pthread_mutex_lock(&room_lock);
        printf("Display --> In the room: %d doctor(s) [ ", num_doctors);
        for (int i = 0; i < num_doctors; i++) {
            printf("D%d ", room_doctors[i]);
        }
        printf("], and %d visitor(s) [ ", num_visitors);
        for (int i = 0; i < num_visitors; i++) {
            printf("V%d ", room_visitors[i]);
        }
        printf("].\n");
        // Release the lock
        pthread_mutex_unlock(&room_lock);
    }
    return NULL;
}
```

Visitor Thread:

The function first waits for a random period of time before attempting to enter the room. Then, it acquires the lock on room_lock and waits on visitor_cond until there are no doctors in the room and the number of visitors is less than MAX_ROOM_VISITORS. Once it is allowed to enter, it increments num_visitors and prints a message indicating that it has entered the room. The thread then releases the lock and waits for a random period of time before attempting to leave the room. When it is time to leave the room, it acquires the lock again, decrements num_visitors, prints a message indicating that it has left the room, and broadcasts on visitor_cond to unblock any waiting threads.

```c
void* visitor_thread(void* arg) {
    int id = *((int*) arg);
    free(arg);


    while (1) {
        // Start wait time
        struct timespec start, end;
        clock_gettime(CLOCK_MONOTONIC, &start);

        // Wait for a random period of time before trying to enter the room
        usleep(rand() % 5000000);

        // Acquire the lock
        pthread_mutex_lock(&room_lock);

        visitor_waiting++;
        printf("Visitor V%d wishes to visit the patient, waiting.\n", id);

        // Wait until there are no doctors in the room and the number of visitors is less than MAX_ROOM_VISITORS
        while (num_doctors > 0 || num_visitors >= MAX_ROOM_VISITORS) {
            pthread_cond_wait(&visitor_cond, &room_lock);
        }
```

```c
    // Add visitor's id to the array of visitors in the room
    visitor_waiting--;
    room_visitors[num_visitors] = id;
    num_visitors++;

    // Calculate wait time
    clock_gettime(CLOCK_MONOTONIC, &end);
    long wait_time_ms = (end.tv_sec - start.tv_sec) * 1000 + (end.tv_nsec - start.tv_nsec) / 1000000;
    printf("Visitor V%d entered the patient's room, waited for %ld ms.\n", id, wait_time_ms);

    // Release the lock
    pthread_mutex_unlock(&room_lock);

    // Start stay time
    clock_gettime(CLOCK_MONOTONIC, &start);

    // Wait for a random period of time before leaving the room
    usleep(rand() % 150000);

    // Acquire the lock
    pthread_mutex_lock(&room_lock);
```

```
// Remove visitor's id from the array of visitors in the room
for (int i = 0; i < num_visitors; i++) {
    if (room_visitors[i] == id) {
        // shift array elements
        for (int j = i; j < num_visitors - 1; j++) {
            room_visitors[j] = room_visitors[j + 1];
        }
        break;
    }
}
num_visitors--;

// Calculate stay time
clock_gettime(CLOCK_MONOTONIC, &end);
long stay_time_ms = (end.tv_sec - start.tv_sec) * 1000 + (end.tv_nsec - start.tv_nsec) / 1000000;
printf("Visitor V%d exited the patient's room, stayed %ld ms.\n", id, stay_time_ms);
```

```
81
82          // Handeling Signal Logic
83          if (visitor_waiting >= 3) {
84              pthread_cond_signal(&visitor_cond);
85          } else if (doctor_waiting > 0) {
86              pthread_cond_signal(&doctor_cond);
87          } else {
88              pthread_cond_signal(&visitor_cond);
89          }
90          // Release the lock
91          pthread_mutex_unlock(&room_lock);
92      }
93      return NULL;
94  }
95
```

Doctor Thread:

The function first waits for a random period before attempting to enter the room. Then, it acquires the lock on room_lock and waits on doctor_cond until there are no visitors in the room. Once it is allowed to enter, it increments num_doctors and prints a message indicating that it has entered the room. The thread then releases the lock and waits for a random period before attempting to leave the room. When it is time to leave the room, it acquires the lock again, decrements num_doctors, prints a message indicating that it has left the room, and broadcasts on doctor_cond and visitor_cond to unblock any waiting threads.

```
void* doctor_thread(void* arg) {
    int id = *((int*) arg);
    free(arg);

    while (1) {

        // Start wait time
        struct timespec start, end;
        clock_gettime(CLOCK_MONOTONIC, &start);

        // Wait for a random period of time before trying to enter the room
        usleep(rand() % 5000000);

        // Acquire the lock
        pthread_mutex_lock(&room_lock);

        doctor_waiting++;
        printf("Doctor D%d wishes to see the patient, waiting.\n", id);

        // Wait until there are no one in the room
        while ((num_visitors > 0) || (num_doctors > 0)) {
            pthread_cond_wait(&doctor_cond, &room_lock);
        }
```

```
// Add doctors's id to the array of doctors in the room
doctor_waiting--;
room_doctors[num_doctors] = id;
num_doctors++;

// Calculate wait time
clock_gettime(CLOCK_MONOTONIC, &end);
long wait_time_ms = (end.tv_sec - start.tv_sec) * 1000 + (end.tv_nsec - start.tv_nsec) / 1000000;
printf("Doctor D%d entered the patient's room, waited for %ld ms.\n", id, wait_time_ms);
// Release the lock
pthread_mutex_unlock(&room_lock);

// Start stay time
clock_gettime(CLOCK_MONOTONIC, &start);

// Wait for a random period of time before leaving the room
usleep(rand() % 300000);
```

```
// Acquire the lock
pthread_mutex_lock(&room_lock);
for (int i = 0; i < num_doctors; i++) {
    if (room_doctors[i] == id) {
        // shift array elements
        for (int j = i; j < num_doctors - 1; j++) {
            room_doctors[j] = room_doctors[j + 1];
        }
        break;
    }
}
num_doctors--;

// Calculate stay time
clock_gettime(CLOCK_MONOTONIC, &end);
long stay_time_ms = (end.tv_sec - start.tv_sec) * 1000 + (end.tv_nsec - start.tv_nsec) / 1000000;
printf("Doctor D%d exited the patient's room, stayed %ld ms.\n", id, stay_time_ms);
```

```
        // Handeling Signal Logic
        if (visitor_waiting >= 3) {
            pthread_cond_signal(&visitor_cond);
        } else if (doctor_waiting > 0) {
            pthread_cond_signal(&doctor_cond);
        } else {
            pthread_cond_signal(&visitor_cond);
        }
        // Release the lock
        pthread_mutex_unlock(&room_lock);
    }
    return NULL;
}
```

Unit Tests for all the program can be found in the Tests Folder