

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

Main:	2
Display Thread:	5
Visitor Thread:	6
Doctor Thread:	8

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);
}

```

```

6 // Cleanup and exit
7 pthread_join(display_tid, NULL);
8
9
10 for (int i = 0; i < num_doctors; i++) {
11     pthread_cancel(doctor_tids[i]);
12 }
13
14 for (int i = 0; i < num_visitors; i++) {
15     pthread_cancel(visitor_tids[i]);
16 }
17
18 for (int i = 0; i < num_doctors; i++) {
19     pthread_join(doctor_tids[i], NULL);
20 }
21
22 for (int i = 0; i < num_visitors; i++) {
23     pthread_join(visitor_tids[i], NULL);
24 }
25
26 pthread_mutex_destroy(&room_lock);
27 pthread_cond_destroy(&doctor_cond);
28 pthread_cond_destroy(&visitor_cond);
29
30 return 0;
31 }

```

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.

```
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);
        }
    }
}
```

```
// 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 %d exited the patient's room, stayed %ld ms.\n", id, stay_time_ms);

```

```

81
82     // Handling 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);

```

```
9
10 // Handling Signal Logic
11 if (visitor_waiting >= 3) {
12     pthread_cond_signal(&visitor_cond);
13 } else if (doctor_waiting > 0) {
14     pthread_cond_signal(&doctor_cond);
15 } else {
16     pthread_cond_signal(&visitor_cond);
17 }
18 // Release the Lock
19 pthread_mutex_unlock(&room_lock);
20 }
21 return NULL;
22 }
23
```

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