Умертаев Арслан БПИ 227 Вариант 16

Условие

Задача о социалистической гостинице. В гостинице 10 одноместных номеров. Клиенты гостиницы снимают номер на одни или
несколько суток (задается при создании клиента). Если в гостинице нет свободных номеров, клиенты не уходят, а устраиваются на
16
рядом с гостиницей на скамейках и ждут в порядке очереди, пока
любой из номеров не освободится (других гостиниц в городе нет).
Создать клиент–серверное приложение, моделирующее работу гостиницы.
Сервер — это гостиница. Прибывающие гости могут порождаться отдельным общим клиентом.
Другой клиент — это скамейки,
образующие очередь ожидающих гостей.

Решение на 5 баллов

Сценарий

Для решения задачи необходимо создать клиент-серверное приложение, где сервер представляет собой гостиницу с 10 одноместными номерами, а клиенты — это гости, которые пытаются снять номер. Если номера заняты, гости ожидают на скамейках, образуя очередь.

Client.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080

int main(int argc, char *argv[]) {
    if (argc != 4) {
        fprintf(stderr, "Usage: %s <server_ip> <port> <days>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    char *server_ip = argv[1];
    int port = atoi(argv[2]);
    int days = atoi(argv[3]);

    int sock = 0;
    struct sockaddr_in serv_addr;

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\n Socket creation error \n");
        return -1;
    }
```

```
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(port);

    if (inet_pton(AF_INET, server_ip, &serv_addr.sin_addr) <= 0) {
        printf("\nInvalid address/ Address not supported \n");
        return -1;
    }

    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        printf("\nConnection Failed \n");
        return -1;
    }

    write(sock, &days, sizeof(int));

    int room_id;
    read(sock, &room_id, sizeof(int));

    if (room_id > 0) {
        printf("Room assigned: %d\n", room_id);
    } else {
        printf("No room available, waiting in queue...\n");
    }

    close(sock);
    return 0;
}
```

Hotel.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <pthread.h>

#define PORT 8080
#define MAX_ROOMS 10
#define MAX_QUEUE 100

typedef struct {
    int id;
    int occupied;
    int days;
} Room;

typedef struct {
    int socket;
    int days;
} ClientRequest;
```

```
Room rooms[MAX_ROOMS];
ClientRequest queue[MAX_QUEUE];
int queue_size = 0;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

void* handle_client(void* arg) {
    int new_socket = *((int*)arg);
    free(arg);
    int days;
    read(new_socket, &days, sizeof(int));

    pthread_mutex_lock(&mutex);

    int room_found = 0;
    for (int i = 0; i < MAX_ROOMS; i++) {
        if (!rooms[i].occupied) {
            rooms[i].occupied = 1;
            rooms[i].days = days;
            write(new_socket, &rooms[i].id, sizeof(int));
            room_found = 1;
            break;
        }
    }

    if (!room_found) {
        queue[queue_size].socket = new_socket;
        queue[queue_size].days = days;
        queue_size++;
    }

    pthread_mutex_unlock(&mutex);

    if (!room_found) {
        // Wait until a room becomes available
        pthread_mutex_lock(&mutex);
        pthread_cond_wait(&cond, &mutex);
        pthread_mutex_unlock(&mutex);

        // Recheck for available room
        pthread_mutex_lock(&mutex);
        for (int i = 0; i < MAX_ROOMS; i++) {
            if (!rooms[i].occupied) {
                rooms[i].occupied = 1;
                rooms[i].days = queue[0].days;
                write(queue[0].socket, &rooms[i].id, sizeof(int));
                for (int j = 0; j < queue_size - 1; j++) {
                    queue[j] = queue[j + 1];
                }
                queue_size--;
                break;
            }
        }
        pthread_mutex_unlock(&mutex);
    }
```

```
    close(new_socket);
    return NULL;
}

void* release_rooms(void* arg) {
    while (1) {
        sleep(1);

        pthread_mutex_lock(&mutex);
        for (int i = 0; i < MAX_ROOMS; i++) {
            if (rooms[i].occupied) {
                rooms[i].days--;
                if (rooms[i].days == 0) {
                    rooms[i].occupied = 0;
                    pthread_cond_signal(&cond);
                }
            }
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}

int main(int argc, char *argv[]) {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);

    for (int i = 0; i < MAX_ROOMS; i++) {
        rooms[i].id = i + 1;
        rooms[i].occupied = 0;
        rooms[i].days = 0;
    }

    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt,
sizeof(opt))) {
        perror("setsockopt");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

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

    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("bind failed");
        close(server_fd);
        exit(EXIT_FAILURE);
```

```
    }

    if (listen(server_fd, 3) < 0) {
        perror("listen");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    pthread_t release_thread;
    pthread_create(&release_thread, NULL, release_rooms, NULL);

    while (1) {
        if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
(socklen_t*)&addrlen)) < 0) {
            perror("accept");
            close(server_fd);
            exit(EXIT_FAILURE);
        }

        int *pclient = malloc(sizeof(int));
        *pclient = new_socket;
        pthread_t thread;
        pthread_create(&thread, NULL, handle_client, pclient);
        pthread_detach(thread);
    }

    close(server_fd);
    return 0;
}
```

```
~/HospitableGoodnaturedDigit$ ./client
Enter number of days to stay: 2
Message sent
Assigned to room 9

~/HospitableGoodnaturedDigit$ ./client
Enter number of days to stay: 4
Message sent
Assigned to room 10

~/HospitableGoodnaturedDigit$ ./client
Enter number of days to stay: 5
Message sent
All rooms are occupied. Please wait.
```