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 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 4 баллов

Сценарий

Гостиница содержит 10 одноместных номеров. Клиенты (гости) снимают номера на определенное количество суток. Если все номера заняты, клиенты ждут на скамейках перед гостиницей в порядке очереди. Как только освобождается номер, первый в очереди клиент заселяется в освободившийся номер.

Client.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> <stay_duration>
<number_of_clients>\n", argv[0]);
        exit(1);
    }

    char *server_ip = argv[1];
    int stay_duration = atoi(argv[2]);
    int number_of_clients = atoi(argv[3]);

    for (int i = 0; i < number_of_clients; i++) {
        int sock = socket(AF_INET, SOCK_STREAM, 0);
        struct sockaddr_in server;
        server.sin_family = AF_INET;
        server.sin_addr.s_addr = inet_addr(server_ip);
        server.sin_port = htons(PORT);

        connect(sock, (struct sockaddr *)&server, sizeof(server));
        send(sock, &stay_duration, sizeof(int), 0);

        int room_number;
```

```
        recv(sock, &room_number, sizeof(int), 0);

        if (room_number == -1) {
            printf("Client %d: All rooms are occupied. Waiting...\n", i);
        } else {
            printf("Client %d: Checked into room %d for %d days\n", i,
room_number, stay_duration);
        }

        close(sock);
    }

    return 0;
}
```

client.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_CLIENTS 100
#define MAX_ROOMS 10

int rooms[MAX_ROOMS] = {0}; // 0 - свободен, 1 - занят
pthread_mutex_t rooms_lock;
int waiting_queue[MAX_CLIENTS];
int queue_size = 0;

void *client_handler(void *socket_desc) {
    int sock = *(int*)socket_desc;
    int stay_duration;
    recv(sock, &stay_duration, sizeof(int), 0);

    pthread_mutex_lock(&rooms_lock);

    int room_found = 0;
    for (int i = 0; i < MAX_ROOMS; i++) {
        if (rooms[i] == 0) {
            rooms[i] = 1;
            room_found = 1;
            send(sock, &i, sizeof(int), 0);
            break;
        }
    }

    if (!room_found) {
        waiting_queue[queue_size++] = sock;
```

```
        int wait = -1;
        send(sock, &wait, sizeof(int), 0);
    }

    pthread_mutex_unlock(&rooms_lock);

    if (room_found) {
        sleep(stay_duration);
        pthread_mutex_lock(&rooms_lock);
        rooms[stay_duration] = 0;

        if (queue_size > 0) {
            int next_client = waiting_queue[0];
            for (int i = 1; i < queue_size; i++) {
                waiting_queue[i-1] = waiting_queue[i];
            }
            queue_size--;
            send(next_client, &stay_duration, sizeof(int), 0);
        }

        pthread_mutex_unlock(&rooms_lock);
    }

    close(sock);
    free(socket_desc);

    return 0;
}

int main(int argc, char *argv[]) {
    int server_sock, client_sock, *new_sock;
    struct sockaddr_in server, client;
    socklen_t sockaddr_size = sizeof(struct sockaddr_in);

    server_sock = socket(AF_INET, SOCK_STREAM, 0);
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(PORT);

    bind(server_sock, (struct sockaddr *)&server, sizeof(server));
    listen(server_sock, 3);

    pthread_mutex_init(&rooms_lock, NULL);

    while ((client_sock = accept(server_sock, (struct sockaddr *)&client,
&sockaddr_size))) {
        pthread_t client_thread;
        new_sock = malloc(sizeof(int));
        *new_sock = client_sock;
        pthread_create(&client_thread, NULL, client_handler, (void*) new_sock);
        pthread_detach(client_thread);
    }

    close(server_sock);
    pthread_mutex_destroy(&rooms_lock);
```

```
    return 0;
}
```

```
~/HospitableGoodnaturedDigit$ ./client 127.0.0.1 1 5
Client 0: Checked into room 1 for 1 days
Client 1: Checked into room 2 for 1 days
Client 2: Checked into room 3 for 1 days
Client 3: Checked into room 4 for 1 days
Client 4: Checked into room 5 for 1 days
```