МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное учреждение высшего образования
«Чувашский государственный университет И.Н. Ульянова»
Факультет информатики и вычислительной техники
Кафедра вычислительной техники

Системное операционные системы
Лабораторная работа 4
Вариант 5

Выполнил:

Студент группы ИВТ-41-20
Галкин Д.С.

Проверил:

Старший преподаватель
Первов С.Г.

Чебоксары, 2024

Цель работы:

Получение навыков работы с сокетами и обработки обмена данными через сеть.

Задание:

В этой лабораторной работе необходимо реализовать вариант задания соответствующий номеру студента (варианты заданий аналогичны работе №1), при этом программа должна выполняться на нескольких устройствах одновременно и, чтобы они могли обмениваться данными через сеть. Обмен данными реализовать по средствам сокетов.

Ход работы:

Server

Функция потока: reader

```
void *reader_thread(void *arg) {
    struct thread_data *data = (struct thread_data *)arg;
    int index = data->index;

    while (1) {
        pthread_mutex_lock(&priority_mutex);
        while (write_count > 0 || (num_writers_waiting > 0 && read_count >=
patience_threshold_read)) {
            pthread_cond_wait(&read_priority, &priority_mutex);
        }
        read_count++;
        pthread_mutex_unlock(&priority_mutex);

        pthread_mutex_lock(&read_mutex);
        num_readers++;
        if (num_readers == 1) {
            pthread_mutex_lock(&write_mutex);
        }
        pthread_mutex_unlock(&read_mutex);

        char stats_message[BUF_SIZE];
        char inMessage[] = "[READER] --> Читатель %ld читает в библиотеке\n";
        char outMessage[] = "[READER] --> Читатель %ld прочитал книгу в библиотеке\n";
        // Чтение
        printf(inMessage, (intptr_t)index);
        snprintf(stats_message, sizeof(stats_message), inMessage, (intptr_t)index);
        broadcast_message(stats_message);
        // send(sock, stats_message, strlen(stats_message), 0); // Отправляем
сообщение клиенту
        usleep(rand() % 1000000);
        printf(outMessage, (intptr_t)index);
        snprintf(stats_message, sizeof(stats_message), outMessage, (intptr_t)index);
        // send(sock, stats_message, strlen(stats_message), 0); // Отправляем
сообщение клиенту
        broadcast_message(stats_message);

        pthread_mutex_lock(&read_mutex);
        num_readers--;
        if (num_readers == 0) {
            pthread_mutex_unlock(&write_mutex);
        }
        pthread_mutex_unlock(&read_mutex);

        pthread_mutex_lock(&priority_mutex);
        read_count--;
        if (num_writers_waiting > 0 && read_count < patience_threshold_read) {
            pthread_cond_signal(&write_priority);
        }
        pthread_mutex_unlock(&priority_mutex);
```

```
        usleep(rand() % 1000000);
    }

    return NULL;
}
```

Функция потока: writer

```
void *writer_thread(void *arg) {
    struct thread_data *data = (struct thread_data *)arg;
    int index = data->index;

    while (1) {
        pthread_mutex_lock(&priority_mutex);
        num_writers_waiting++;
        while (num_readers > 0 || write_count > 0) {
            pthread_cond_wait(&write_priority, &priority_mutex);
        }
        num_writers_waiting--;
        write_count++;
        pthread_mutex_unlock(&priority_mutex);

        pthread_mutex_lock(&write_mutex);
        char stats_message[BUF_SIZE];
        char inMessage[] = "[WRITER] --> Писатель %ld пишет в библиотеке\n";
        char outMessage[] = "[WRITER] --> Писатель %ld закончил писать в
библиотеке\n";
        // Запись
        printf(inMessage, (intptr_t)index);
        snprintf(stats_message, sizeof(stats_message), inMessage, (intptr_t)index);
        // send(sock, stats_message, strlen(stats_message), 0); // Отправляем
сообщение клиенту
        broadcast_message(stats_message);
        usleep(rand() % 3000000);
        printf(outMessage, (intptr_t)index);
        snprintf(stats_message, sizeof(stats_message), outMessage, (intptr_t)index);
        // send(sock, stats_message, strlen(stats_message), 0); // Отправляем
сообщение клиенту
        broadcast_message(stats_message);
        pthread_mutex_unlock(&write_mutex);

        pthread_mutex_lock(&priority_mutex);
        write_count--;
        if (write_count == 0) {
            if (num_writers_waiting > 0 && read_count >= patience_threshold_read) {
                pthread_cond_signal(&write_priority);
            } else {
                pthread_cond_broadcast(&read_priority);
            }
        }
        pthread_mutex_unlock(&priority_mutex);

        usleep(rand() % 3000000);
    }
}
```

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include "server.h"

#include <signal.h>
#include <string.h>
#include <arpa/inet.h>

#define PORT 9999
#define BUF_SIZE 1024


// #define NUM_READERS 5
// #define NUM_WRITERS 5

struct thread_data {
    int index;
};

#define MAX_CLIENTS 100  // Максимальное количество клиентов
int clients[MAX_CLIENTS]; // Массив для хранения сокетов клиентов
int active_clients = 0;
pthread_mutex_t clients_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t client_count_mutex = PTHREAD_MUTEX_INITIALIZER;
int client_count = 0;

pthread_mutex_t write_mutex;
pthread_mutex_t read_mutex;
pthread_mutex_t priority_mutex;
pthread_cond_t write_priority;
pthread_cond_t read_priority;

void init_server_resources() {
    pthread_mutex_init(&write_mutex, NULL);
    pthread_mutex_init(&read_mutex, NULL);
    pthread_mutex_init(&priority_mutex, NULL);
    pthread_cond_init(&write_priority, NULL);
    pthread_cond_init(&read_priority, NULL);
}

void destroy_server_resources() {
    pthread_mutex_destroy(&write_mutex);
    pthread_mutex_destroy(&read_mutex);
    pthread_mutex_destroy(&priority_mutex);
    pthread_cond_destroy(&write_priority);
    pthread_cond_destroy(&read_priority);
}

// Функция для отправки сообщения всем клиентам, кроме отправителя
void broadcast_message(char *message) {
```

```
    pthread_mutex_lock(&clients_mutex);
    for (int i = 0; i < active_clients; i++) {
        if (clients[i] > 0) {  // Проверка, что сокет активен и не является
отправителем
            send(clients[i], message, strlen(message), 0);
        }
    }
    pthread_mutex_unlock(&clients_mutex);
}

void *reader_thread(void *arg);
void *writer_thread(void *arg);

int main() {
    setbuf(stdout, NULL);
    init_server_resources();

    int server_fd, new_socket;
    pthread_t tid;

    if (setup_server_socket(&server_fd) < 0) {
        destroy_server_resources();
        exit(1); // Возвращаем ошибку, но уже после освобождения ресурсов
    }

    printf("Server is running and waiting for connections on port %d...\n", PORT);

    struct sockaddr_in client_addr;
    socklen_t client_addr_len = sizeof(client_addr);
    while (1) {
        if ((new_socket = accept(server_fd, NULL, NULL)) < 0) {
            perror("accept");
            continue; // Вместо выхода продолжаем работу сервера
        }

        // Вывод информации о подключившемся клиенте
        char client_ip[INET_ADDRSTRLEN];
        inet_ntop(AF_INET, &(client_addr.sin_addr), client_ip, INET_ADDRSTRLEN);

        pthread_create(&tid, NULL, handle_client, (void *)(intptr_t)new_socket);
    }

}

int setup_server_socket(int *server_fd) {
    struct sockaddr_in address;
    int opt = 1;


    if ((*server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        return -1;
    }

    // Настройка сокета для повторного использования порта
    if (setsockopt(*server_fd, SOL_SOCKET, SO_REUSEADDR, &(int){ opt }, sizeof(int)) <
0) {
```

```
        perror("setsockopt(SO_REUSEADDR) failed");
        close(*server_fd);
        return -1;
    }

    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY; // Привязка ко всем интерфейсам
    address.sin_port = htons(PORT);

    if (bind(*server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("bind failed");
        close(*server_fd);
        return -1;
    }
    // что делает htuns
    // зачем нужен listen (2 аргумент)
    if (listen(*server_fd, 10) < 0) {
        perror("listen");
        close(*server_fd);
        return -1;
    }

    return 0; // Успешное завершение
}


void *handle_client(void *arg) {
    int sock = (intptr_t)arg;
    char buffer[BUF_SIZE] = {0};
    int num_threads;
    char *token, *ptr;

    struct sockaddr_in peer_addr;
    socklen_t peer_addr_len = sizeof(peer_addr);
    getpeername(sock, (struct sockaddr *)&peer_addr, &peer_addr_len); // Получение
информации о клиенте

    char client_ip[INET_ADDRSTRLEN];
    inet_ntop(AF_INET, &peer_addr.sin_addr, client_ip, INET_ADDRSTRLEN);
    int client_port = ntohs(peer_addr.sin_port);

    // Добавление нового клиента
    pthread_mutex_lock(&clients_mutex);
    active_clients++;
    for (int i = 0; i < active_clients; i++) {
        if (clients[i] == 0) {
            clients[i] = sock;
            printf("[SERVER] --> Client connected(%d): %s:%d\n", active_clients,
client_ip, client_port);
            break;
        }
    }
    pthread_mutex_unlock(&clients_mutex);

    while (1) { // Бесконечный цикл чтения
        memset(buffer, 0, BUF_SIZE); // Очистите буфер перед каждым чтением
        int read_val = read(sock, buffer, BUF_SIZE);
```

```
        if (read_val > 0) {
            // Разделение сообщения на токены
            token = strtok(buffer, ":");
            if (token != NULL) {
                ptr = strtok(NULL, ":");
                if (ptr != NULL) {
                    num_threads = atoi(ptr); // Преобразование строки в число

                    printf("[CLIENT][%s:%d] --> Client %s create threads: %d\n",
client_ip, client_port, strcmp(token, "Reader") ==0 ? "Reader" : "Writer",
num_threads);

                    struct thread_data *data;
                    pthread_t threads[num_threads];
                    pthread_mutex_lock(&client_count_mutex);
                    for (int i = 0; i < num_threads; i++) {
                        data = malloc(sizeof(struct thread_data)); // Выделяем память
для данных потока
                        data->index = client_count;
                        client_count++;
                        // data->active = &active;
                        if (strcmp(token, "Reader") == 0) {
                            // Создание потоков читателей
                            pthread_create(&threads[i], NULL, reader_thread, (void
*)data);
                        } else if (strcmp(token, "Writer") == 0) {
                            // Создание потоков писателей
                            pthread_create(&threads[i], NULL, writer_thread, (void
*)data);
                        }
                    }
                    pthread_mutex_unlock(&client_count_mutex);

            }
        } else {
            break;
        }
    }

    // Удаление клиента при отключении
    pthread_mutex_lock(&clients_mutex);
    for (int i = 0; i < active_clients; i++) {
        if (clients[i] == sock) {
            clients[i] = 0;
            printf("[SERVER] --> Client disconnected(%d): %s:%d\n", i + 1, client_ip,
client_port);
            active_clients--;
            break;
        }
    }
    pthread_mutex_unlock(&clients_mutex);
    close(sock);
    return NULL;
}
```

Client

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include "server.h"

#define SERVER_PORT 9999
#define SERVER_IP "127.0.0.1" // Use the server IP address
#define BUF_SIZE 1024

int keep_reading = 0; // Global variable for managing data reading
int run_client = 1;

// Function for reading data from the server
void *read_from_server(void *arg) {
    int sock = *(int *) arg;
    while (1) {
        if (!keep_reading)
            continue;
        char server_response[BUF_SIZE] = {0};
        int bytes_read = recv(sock, server_response, BUF_SIZE, 0); // Reading messages
from the server
        if (bytes_read <= 0) {
            break; // Exit the loop if the connection is closed or an error occurred
        }
        printf("%s", server_response); // Print message from the server
    }
    return NULL;
}

int main() {
    struct sockaddr_in serv_addr;
    char buffer[BUF_SIZE] = {0};
    int num_threads; // Ensure this is declared
    char client_message[BUF_SIZE];

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

    memset(&serv_addr, '0', sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(SERVER_PORT);

    if (inet_pton(AF_INET, SERVER_IP, &serv_addr.sin_addr) <= 0) {
        printf("\nInvalid address/ Address not supported \n");
        close(sock);
```

```
        return 0;
    }

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

    // Create and start a thread for reading data from the server
    pthread_t threads;
    keep_reading = 1; // Allow reading data
    pthread_create(&threads, NULL, read_from_server, (void *) &sock);

    while (run_client) {
        keep_reading = 1; // Allow reading data
        while (keep_reading) {
            fgets(buffer, BUF_SIZE, stdin); // Read line input
            if (strcmp(buffer, "\n") == 0) {
                keep_reading = 0; // Stop readng data
                sleep(1);
                printf("Enter the number of niggers of reader threads to create or
type 'exit' to quit: ");

                fgets(buffer, BUF_SIZE, stdin);
                char type; // 'r' for readers, 'w' for writers
                if (sscanf(buffer, "-%c %d", &type, &num_threads) == 2) {
                    if (type == 'r' || type == 'w') {
                        snprintf(client_message, sizeof(client_message), "%s:%d",
(type == 'r') ? "Reader" : "Writer", num_threads);
                        send(sock, client_message, strlen(client_message), 0);
                        printf("[CLIENT] --> Request sent: %s\n", client_message);
                        keep_reading = 1;
                    }
                } else if (strncmp(buffer, "exit\n", 5) == 0) {
                    printf("[CLIENT] --> Closing client...\n");
                    keep_reading = 0;
                    run_client = 0; // Устанавливаем флаг для завершения основного
цикла
                    // pthread_join(threads, NULL);                    close(sock);
                    break; // Выходим из внутреннего цикла
                }
            }
        }
    }
    return 0;
}
```

Пример работы:

Запуск сервера

```
./server
Server is running and waiting for connections on port 9999...
```

Подключение 1 клиента

```
./server
Server is running and waiting for connections on port 9999...
[SERVER] --> Client connected(1): 127.0.0.1:59253
```

Отправка 5 читателей

```
Enter the number of niggers of reader threads to create or type 'exit' to quit: -r 5
[CLIENT] --> Request sent: Reader:5
[READER] --> Читатель 0 читает в библиотеке
[READER] --> Читатель 1 читает в библиотеке
```

```
./server
Server is running and waiting for connections on port 9999...
[SERVER] --> Client connected(1): 127.0.0.1:59253
[CLIENT][127.0.0.1:59253] --> Client Reader create threads: 5
[READER] --> Читатель 0 читает в библиотеке
[READER] --> Читатель 1 читает в библиотеке
[READER] --> Читатель 2 читает в библиотеке
[READER] --> Читатель 3 читает в библиотеке
[READER] --> Читатель 4 читает в библиотеке
[READER] --> Читатель 0 прочитал книгу в библиотеке
[READER] --> Читатель 4 прочитал книгу в библиотеке
[READER] --> Читатель 4 читает в библиотеке
```

Отправка 5 писателей

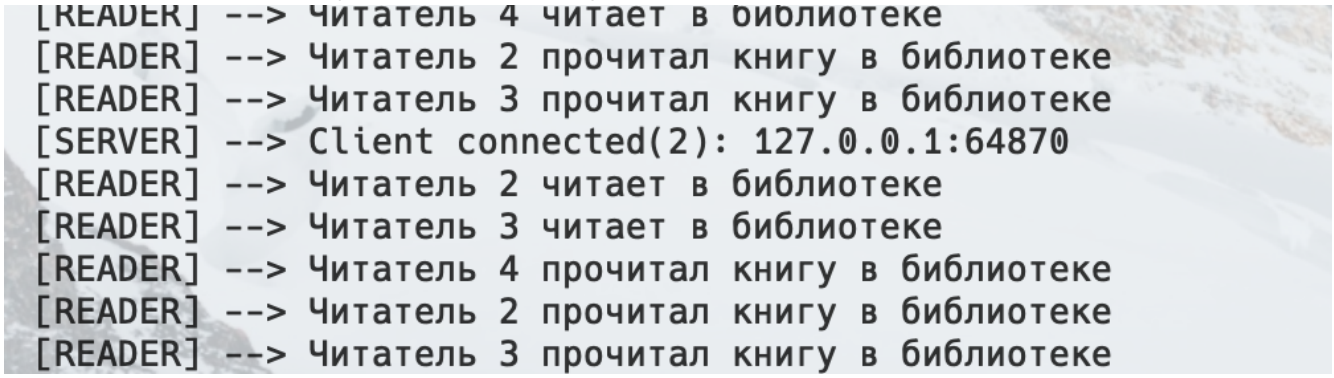
```
[READER] --> Читатель 3 читает в библиотеке

[READER] --> Читатель 0 прочитал книгу в библиотеке
Enter the number of niggers of reader threads to create or type 'exit' to quit: -w 5
[CLIENT] --> Request sent: Writer:5
[READER] --> Читатель 1 читает в библиотеке
[READER] --> Читатель 4 читает в библиотеке
[READER] --> Читатель 4 прочитал книгу в библиотеке
```

```
[READER] --> Читатель 2 читает в библиотеке
[READER] --> Читатель 1 прочитал книгу в библиотеке
[READER] --> Читатель 3 читает в библиотеке
[READER] --> Читатель 3 прочитал книгу в библиотеке
[CLIENT][127.0.0.1:59253] --> Client Writer create threads: 5
[READER] --> Читатель 0 читает в библиотеке
[READER] --> Читатель 4 прочитал книгу в библиотеке
[READER] --> Читатель 2 прочитал книгу в библиотеке
[READER] --> Читатель 0 прочитал книгу в библиотеке
[WRITER] --> Писатель 7 пишет в библиотеке
[WRITER] --> Писатель 7 закончил писать в библиотеке
```

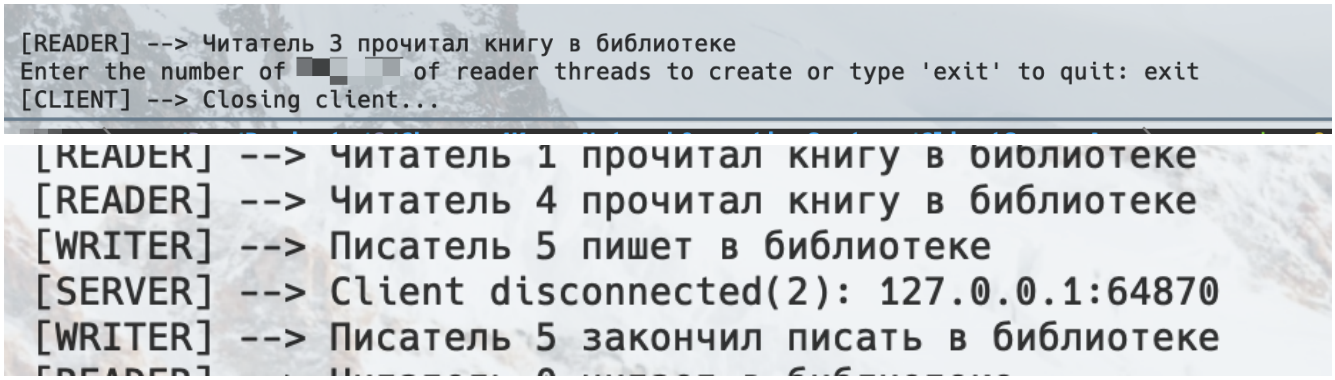
Подключение 2 клиента

Если модель на сервере уже работает, сразу начинает выводиться на подключенном клиенте

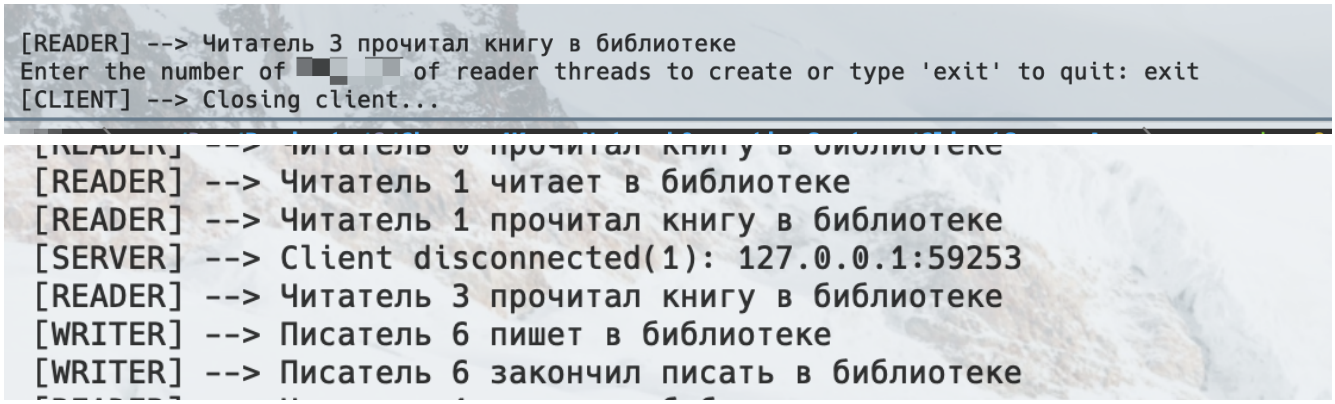



```
❯
./client
[READER] --> Читатель 2 читает в библиотеке
[READER] --> Читатель 3 читает в библиотеке
[READER] --> Читатель 4 прочитал книгу в библиотеке
[READER] --> Читатель 2 прочитал книгу в библиотеке
[READER] --> Читатель 3 прочитал книгу в библиотеке
[WRITER] --> Писатель 9 пишет в библиотеке
[WRITER] --> Писатель 9 закончил писать в библиотеке
[READER] --> Читатель 1 читает в библиотеке
[READER] --> Читатель 0 читает в библиотеке
[READER] --> Читатель 3 читает в библиотеке
[READER] --> Читатель 0 прочитал книгу в библиотеке
```

Отключение 1 клиента



Отключение 2 клиента



Дополнительные вопросы:

1. Что такое htons?
   Используется для преобразования номера IP-порта в порядке байтов узла в номер IP-порта в порядке байтов сети(от старших к младшим).
   Почему именно такой порядок:
   1 - такой порядок более привычный к мышлению людей, поскольку соответствует обычному способу записи чисел от большему к меньшему.
   2 - Исторический стандарт (TCP, UDP) разработаны с таким порядком
   3 - Обеспечение универсальности и взаимопонимания между различными устройствами в сети, независимо от их архитектуры
2. Зачем нужен 2 параметр в методе listen()?
   Кол-во число запросов, которые сервер может обрабатывать одновременно. Каждый раз, когда очередной клиент пытается соединиться с сервером, его запрос ставится в очередь, т.к сервер может быть занят обработкой других запросов. Если очередь заполнена, все последующие запросы будут игнорироваться.

Вывод:

В ходе выполнения лабораторной работы я получил навыки работы с сокетами и обработками данных через сеть.