Отчет по лабораторной работе №9

«Взаимодействие процессов по сети»

Выполнил: студент группы РИС-22-1б

Поважный Виталий Евгеньевич

Вариант 12

02.03.2025

**Задание**

Цель работы: научиться организовывать взаимодействие двух процессов.

Необходимо разработать программу на Си из двух частей (клиент и сервер). Обмен между клиентом и сервером должен происходить с использованием одного из способов взаимодействия процессов (по варианту). При этом на каждое обращение клиента сервер должен обрабатывать в отдельном потоке (способ организации потока по вариантам).

**Ориентировочная последовательность действий**

**Сервер**

• принимает соединение от клиента (по варианту)

• запускает процесс обработки запроса (по варианту)

• процесс обработки запроса обрабатывает запрос

• процесс обработки запроса передает клиенту ответ

**Клиент**

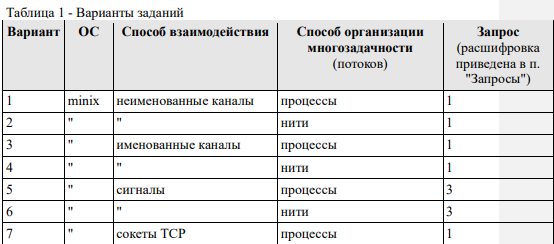
• запрашивает у пользователя данные для запроса (по варианту)

• посылает запрос серверу

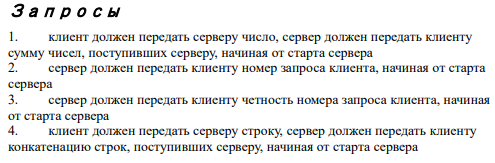
• принимает ответ от сервера

• ответ выдает ответ пользователю

Варианты:







Мой вариант: 12

**Ход работы**

Программа реализует взаимодействие между клиентом и сервером с использованием процесса и потоков. Клиент отправляет строку на сервер, а сервер отвечает номером запроса, увеличиваемым с каждым запросом.

1. **Клиент**:
   * Вводит строку с консоли и отправляет ее на сервер через пайп.
   * Ожидает от сервера номер запроса, который сервер увеличивает с каждым новым запросом.
   * Прекращает работу при вводе строки "exit".
2. **Сервер**:
   * Читает строку от клиента, увеличивает счетчик запросов и создает поток для обработки каждого запроса.
   * Отправляет клиенту номер запроса.
   * Прекращает работу при получении строки "exit" от клиента.
3. **Механизм синхронизации**:
   * Используется мьютекс для синхронизации доступа к глобальному счетчику запросов, чтобы избежать гонки при увеличении номера запроса.

Пример работы программы ниже:

root@minix:~/lab9# ./lab9

Enter a string (or `exit` to quit): hello

Server request number is 1

Enter a string (or `exit` to quit): world

Server request number is 2

Enter a string (or `exit` to quit): 123

Server request number is 3

Enter a string (or `exit` to quit): exit

root@minix:~/lab9#

Рисунок 1 – Пример работы программы.

Код программы:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/wait.h>

#include <minix/mthread.h>

int pipe\_req[2];

int pipe\_res[2];

int request\_count = 0; // Счетчик запросов

mthread\_mutex\_t request\_mutex;

// Эта функция выполняется в отдельном потоке для каждого запроса клиента.

void \*handle\_request(void \*arg) {

mthread\_mutex\_lock(&request\_mutex); //блокирует мьютекс, чтобы только один поток мог изменять

request\_count++;

int request\_number = request\_count;

mthread\_mutex\_unlock(&request\_mutex); // разблокирует мьютекс, позволяя другим потокам работать с request\_count.

// Отправляем номер запроса клиенту

write(pipe\_res[1], &request\_number, sizeof(request\_number));

return NULL;

}

int main(void) {

if (pipe(pipe\_req) == -1 || pipe(pipe\_res) == -1) {

perror("pipe");

exit(EXIT\_FAILURE);

}

if (mthread\_mutex\_init(&request\_mutex, NULL) != 0) {

fprintf(stderr, "mthread\_mutex\_init error\n");

exit(EXIT\_FAILURE);

}

pid\_t pid = fork();

if (pid < 0) {

perror("fork");

exit(EXIT\_FAILURE);

}

if (pid == 0) { // Клиент

close(pipe\_req[0]); // клиент только пишет

close(pipe\_res[1]); // клиент только читает

while (1) {

char input\_str[256];

printf("Enter a string (or 'exit' to quit): ");

fflush(stdout); // сбрасывает буфер вывода, чтобы строка приглашения сразу отобразилась.

// Читаем строку из ввода

if (fgets(input\_str, sizeof(input\_str), stdin) == NULL) {

fprintf(stderr, "Error reading input\n");

break;

}

// Убираем символ новой строки, если он есть

input\_str[strcspn(input\_str, "\n")] = '\0';

if (strcmp(input\_str, "exit") == 0) {

break;

}

// Отправляем строку серверу

write(pipe\_req[1], input\_str, strlen(input\_str) + 1);

int server\_request\_number;

ssize\_t r = read(pipe\_res[0], &server\_request\_number, sizeof(server\_request\_number)); // Клиент читает номер запроса от сервера

if (r <= 0) {

fprintf(stderr, "Server closed or error\n");

break;

}

printf("Server request number is %d\n", server\_request\_number);

}

close(pipe\_req[1]);

close(pipe\_res[0]);

exit(0);

} else { // Сервер

close(pipe\_req[1]); // сервер только читает

close(pipe\_res[0]); // сервер только пишет

// Сервер читает строки от клиента

while (1) {

char input\_str[256];

ssize\_t r = read(pipe\_req[0], input\_str, sizeof(input\_str));

if (r <= 0) {

break;

}

// Строка завершения

if (strcmp(input\_str, "exit") == 0) {

break;

}

// Создаем новый поток для обработки запроса

int \*arg = malloc(sizeof(int));

\*arg = 0; // Здесь можно передать дополнительные данные, если нужно

mthread\_thread\_t thr;

// Создается новый поток с помощью mthread\_create, который выполняет функцию handle\_request.

if (mthread\_create(&thr, NULL, handle\_request, arg) != 0) {

perror("mthread\_create");

free(arg);

break;

}

mthread\_join(thr, NULL); // ожидает завершения потока, чтобы сервер мог продолжить обработку следующего запроса.

}

close(pipe\_req[0]);

close(pipe\_res[1]);

wait(NULL); // Ожидание завершения клиента

mthread\_mutex\_destroy(&request\_mutex);

return 0;

}

}