Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления
КАФЕДРА Информационная безопасность (ИУ8)

Безопасность Систем Баз Данных

Отчет
по Лабораторной работе №5
“Работа с unix-socket”

Выполнил:
Евула А. С.,
студент группы ИУ8-63

Проверил:
Зенькович С. А.,
старший преподаватель
кафедры ИУ8

Москва, 2021

ОГЛАВЛЕНИЕ

ЦЕЛЬ РАБОТЫ

Разработать приложения для работы с UNIX-сокетами.

ОСНОВНАЯ ЧАСТЬ

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

UNIX-сокет - механизм обмена данными, не использующая сетевого протокола для взаимодействия (обмен данными происходит локально). Используется в операционных системах, поддерживающих стандарт POSIX, для межпроцессного взаимодействия. В дополнение к отсылаемым данным процессы могут отсылать файловые дескрипторы через соединение на основе UDS.

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

В ходе выполнения лабораторной работы были реализованы сервер и клиент.

1. СЕРВЕР

- создает локальный сокет
- соединяет его с именем сокета
- ожидает подключения клиентов
- устанавливает подключение с ними
- запрашивает имя (идентификация)
- получает и отправляет данные клиенту
- ведет отчет передаваемых данных (логирование в `server.log`)
- при получении сообщения “exit” отключается от клиента

На входной вопрос (сообщение) клиента сервер отвечает случайно фразой по принципу “8-ball”

Для удобного использования были вынесены 2 метода `print_state` (логирует информацию и выводит ее в нужный поток), `send_to_client` (логирует отправляемое сообщение и отправляет его клиенту).

1. СОЗДАНИЕ И НАСТРОЙКА СОКЕТА СЕРВЕРА

```
unlink(SERVER_NAME);

int listenFd = socket(AF_UNIX, SOCK_STREAM, 0);
if (listenFd < 0)
{
    print_state(log, 1, "ERROR: socket creation failure\n");
    exit(1);
}

struct sockaddr_un serverAddr{};
serverAddr.sun_family = AF_UNIX;
strncpy(serverAddr.sun_path, SERVER_NAME, sizeof(serverAddr.sun_path) - 1);

if (bind(listenFd, (const struct sockaddr *)&serverAddr, sizeof(serverAddr))
< 0)
{
    print_state(log, 1, "ERROR: bind failure\n");
    exit(1);
}
```

2. ФУНКЦИЯ ВЫВОДА В ПОТОК С ЗАПИСЬЮ В ЛОГ

```
template <typename... Args>
void print_state(FILE *log, int code, char *format, Args... args)
{
    fprintf((code == 0) ? stdout : stderr, format, args...);
    fprintf(log, format, args...);
}
```

3. ФУНКЦИЯ ОТПРАВКИ СООБЩЕНИЯ КЛИЕНТУ С ЗАПИСЬЮ В ЛОГ

```
void send_to_client(FILE *log, int &client, char *msg)
{
    char buffer[BUF_SIZE] = {};

    print_state(log, 0, "[%s] %s\n", SERVER_NAME, msg);

    sprintf(buffer, "%s\n", msg);
    write(client, buffer, strlen(buffer));
}
```

Исходный код в Приложение А и Приложение В (массив возможных ответов на вопрос клиента).

2. КЛИЕНТ

- создает локальный сокет
- соединяет его с именем сокета
- устанавливает подключение с сервером
- получает и отправляет данные серверу
- при вводе "exit" отключается от сервера

1. СОЗДАНИЕ СОКЕТА КЛИЕНТА И ПОДКЛЮЧЕНИЕ К СЕРВЕРУ

```
int clientFd = socket(AF_UNIX, SOCK_STREAM, 0);
if (clientFd < 0)
{
    fprintf(stderr, "ERROR: socket creation failure\n");
    exit(1);
}

struct sockaddr_un serverAddr
{
};
serverAddr.sun_family = AF_UNIX;
strncpy(serverAddr.sun_path, SERVER_NAME, sizeof(serverAddr.sun_path) - 1);

if (connect(clientFd, (const struct sockaddr *)&serverAddr,
sizeof(serverAddr)) < 0)
{
    fprintf(stderr, "ERROR: connection failure\n");
    exit(1);
}
```

Исходный код в Приложение Б.

3. ПРИМЕР ВЗАИМОДЕЙСТВИЯ КЛИЕНТА И СЕРВЕРА

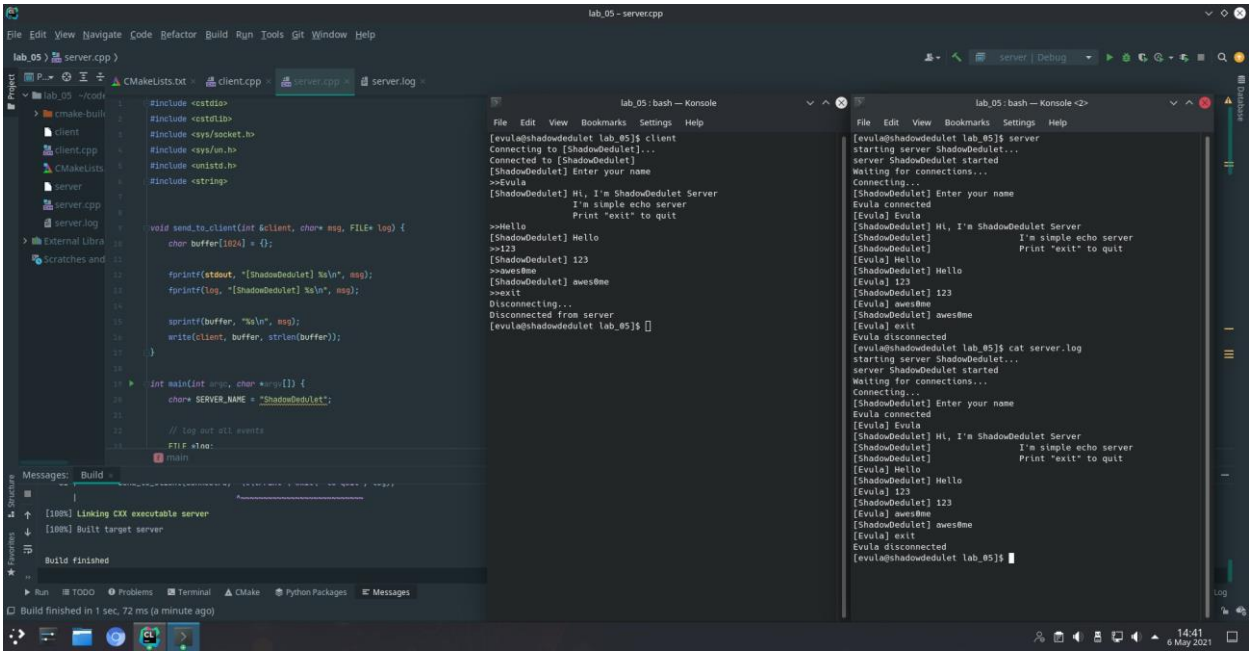

Рисунок 1 – пример использования unix-сокетов

ВЫВОДЫ

Были получены знания о Unix-сокетах и принципах взаимодействия сервер-клиент.

ПРИЛОЖЕНИЕ А

```
#include <cstdio>
#include <cstdlib>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#include <string>
#include "Answers.h"

#define SERVER_NAME "ShadowDedulet"
#define BUF_SIZE 1024

template <typename... Args>
void print_state(FILE *log, int code, char *format, Args... args)
{
    fprintf((code == 0) ? stdout : stderr, format, args...);
    fprintf(log, format, args...);
}

void send_to_client(FILE *log, int &client, char *msg)
{
    char buffer[BUF_SIZE] = {};

    print_state(log, 0, "[%s] %s\n", SERVER_NAME, msg);

    sprintf(buffer, "%s\n", msg);
    write(client, buffer, strlen(buffer));
}

int main(int argc, char *argv[])
{
    FILE *log;
    log = fopen("./server.log", "w");

    srand(time(nullptr));

    print_state(log, 0, "starting server %s...\n", SERVER_NAME);

    unlink(SERVER_NAME);

    int listenFd = socket(AF_UNIX, SOCK_STREAM, 0);
    if (listenFd < 0)
    {
        print_state(log, 1, "ERROR: socket creation failure\n");
        exit(1);
    }

    struct sockaddr_un serverAddr
    {
    };
    serverAddr.sun_family = AF_UNIX;
```

```
    /// 'sizeof(sockaddr_un.sun_path)-1' fixes warning when using
'sockaddr_un.sun_path - 1'
    strncpy(serverAddr.sun_path, SERVER_NAME, sizeof(serverAddr.sun_path) -
1);

    if (bind(listenFd, (const struct sockaddr *)&serverAddr,
sizeof(serverAddr)) < 0)
    {
        print_state(log, 1, "ERROR: bind failure\n");
        exit(1);
    }

    if (listen(listenFd, 5) < 0)
    {
        print_state(log, 1, "ERROR: listening failure\n");
        exit(1);
    }

    print_state(log, 0, "server %s started\n", SERVER_NAME);
    print_state(log, 0, "Waiting for connections...\n");

    char clientName[BUF_SIZE] = {};
    while (true)
    {
        int connectFd = accept(listenFd, nullptr, nullptr);
        if (connectFd < 0)
        {
            print_state(log, 1, "ERROR: client connection failure\n");
        }

        // Authentication part
        print_state(log, 0, "Connecting...\n");
        send_to_client(log, connectFd, "Enter your name");
        read(connectFd, clientName, BUF_SIZE);
        print_state(log, 0, "%s connected\n", clientName);
        print_state(log, 0, "[%s] %s\n", clientName, clientName);

        send_to_client(log, connectFd, "Hi, I'm ShadowDedulet\t");
        send_to_client(log, connectFd, "\t\t8-ball Server");
        send_to_client(log, connectFd, "\t\tPrint \"exit\" to quit");

        while (true)
        {
            char buffer[BUF_SIZE] = {};
            read(connectFd, buffer, BUF_SIZE);

            print_state(log, 0, "[%s] %s\n", clientName, buffer);

            if (std::string(buffer) == "exit")
            {
                print_state(log, 0, "%s disconnected\n", clientName);
                close(connectFd);
                break;
            }
```

```
            size_t index = rand() % Answers::answers.size();
            char *answer = Answers::answers[index];
            send_to_client(log, connectFd, answer);
        }
        break;
    }
    close(listenFd);
    unlink(SERVER_NAME);
    exit(0);
}
```

ПРИЛОЖЕНИЕ Б

```
#include <cstdio>
#include <cstdlib>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>

#define SERVER_NAME "ShadowDedulet"
#define BUF_SIZE 1024

int main(int argc, char *argv[])
{
    fprintf(stdout, "Connecting to [%s]...\n", SERVER_NAME);

    int clientFd = socket(AF_UNIX, SOCK_STREAM, 0);
    if (clientFd < 0)
    {
        fprintf(stderr, "ERROR: socket creation failure\n");
        exit(1);
    }

    struct sockaddr_un serverAddr
    {
    };
    serverAddr.sun_family = AF_UNIX;
    strncpy(serverAddr.sun_path, SERVER_NAME, sizeof(serverAddr.sun_path) -
1);

    if (connect(clientFd, (const struct sockaddr *)&serverAddr,
sizeof(serverAddr)) < 0)
    {
        fprintf(stderr, "ERROR: connection failure\n");
        exit(1);
    }

    fprintf(stdout, "Connected to [%s]\n", SERVER_NAME);

    ssize_t len;
    while (true)
    {
        char read_buf[BUF_SIZE] = {}, write_buf[BUF_SIZE] = {};
        len = read(clientFd, read_buf, BUF_SIZE);
        if (len <= 0)
        {
            fprintf(stdout, "Disconnecting...\n");
            break;
        }

        fprintf(stdout, "[%s] %s>>", SERVER_NAME, read_buf);
        fscanf(stdin, "%[^\n]%*c", write_buf);
        write(clientFd, write_buf, strlen(write_buf));
```

```
    }

    fprintf(stdout, "Disconnected from server\n");
    close(clientFd);
    exit(0);
}
```

ПРИЛОЖЕНИЕ В

```
#ifndef LAB_05_ANSWERS_H
#define LAB_05_ANSWERS_H

#include <vector>

struct Answers
{
    static const std::vector<char *> answers;
};
const std::vector<char *> Answers::answers = {
    "It is certain (Бесспорно)",
    "It is decidedly so (Предрешено)",
    "Without a doubt (Никаких сомнений)",
    "You may rely on it (Можешь быть уверен в этом)",
    "As I see it, yes (Мне кажется — «да»)",
    "Most likely (Вероятнее всего)",
    "Outlook good (Хорошие перспективы)",
    "Yes (Да)",
    "Reply hazy, try again (Пока не ясно, попробуй снова)",
    "Ask again later (Спроси позже)",
    "Better not tell you now (Лучше не рассказывать)",
    "Concentrate and ask again (Сконцентрируйся и спроси опять)",
    "Don’t count on it (Даже не думай)",
    "My reply is no (Мой ответ — «нет»)",
    "Outlook not so good (Перспективы не очень хорошие)",
    "Very doubtful (Весьма сомнительно)"};

#endif //LAB_05_ANSWERS_H
```