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)

Отчёт
по лабораторной работе №5
по дисциплине «Безопасность систем баз данных»

Выполнил: Песоцкий А. А.,
студент группы ИУ8-61

Проверил: Зенькович С. А.,
ассистент каф. ИУ8

Москва 2020 г.

ОГЛАВЛЕНИЕ

Table of Contents

ВСТУПЛЕНИЕ

Цель работы:

Ознакомиться с IP-socket и unix-socket, а также разработать приложение на C++11, которое должно уметь:

1. Получать данные из socket.
2. Передавать данные в socket.
3. Идентифицировать отправителя/получателя.
4. Вести историю передаваемых данных.

РАЗЛИЧИЯ МЕЖДУ IP-SOCKET И UNIX-SOCKET

Сокет UNIX представляет собой механизм межпроцессного взаимодействия , который позволяет осуществлять обмен данными между двунаправленных процессов , работающих на одной и той же машине.

Сокет IP (особенно сокеты TCP / IP) – это механизм, позволяющий осуществлять связь между процессами по сети. В некоторых случаях можно использовать сокеты TCP / IP для связи с процессами, запущенными на одном компьютере (с помощью интерфейса обратной связи).

Доменные сокеты UNIX знают, что они выполняются в одной и той же системе, поэтому они могут избежать некоторых проверок и операций (например, маршрутизации); что делает их быстрее и легче, чем IP-сокеты.

Как реализуется взаимодействие со стороны сервера:

— системный вызов *socket* создаёт сокет, но этот сокет не может использоваться совместно с другими процессами;
— сокет именуется. Для локальных сокетов домена *AF_UNIX*(*AF_LOCAL*) адрес будет задан именем файла. Сетевые сокеты *AF_INET* именуются в соответствии с их ip/портом;
— системный вызов *listen(int socket, int backlog)* формирует очередь входящих подключений. Второй параметр backlog определяет длину этой очереди;
— эти подключения сервер принимает с помощью вызова *accept*, который создаёт новый сокет, отличающийся от именованного сокета. Этот новый сокет применяется только для взаимодействия с данным конкретным клиентом.

С точки зрения клиента подключение происходит несколько проще:

— вызывается *socket*;
— и *connect*, используя в качестве адреса именованный сокет сервера.

ПРИМЕР ИСПОЛЬЗОВАНИЯ ПРИЛОЖЕНИЯ

Выбор типа сокета осуществляется с помощью define:

- #define SOCKET_INET
- #define SOCKET_UNIX

Сервер реализован как эхо-сервер – в ответ на переданное сообщение клиент получает его обратно с выводом в консоль.

1. Пример использования с unix-socket

Для начала протестируем приложения на unix-socket. Сервер запускается командой с одним аргументом – именем файла истории, а затем в зависимости от типа сокета запрашивается ввод необходимых данных:

```
$ ./server filename
```

```
[artem@asymmetriq lab_5]$ ./server out.txt
SERVER IDENTIFIER: random_id
listening socket works on: random_id:
waiting for connection
accepted from: 2137759437
connection established
```

Рисунок 1. Запуск сервера, unix-socket

Клиент запускается с одним аргументом – передаваемым сообщением, а затем в зависимости от типа сокета запрашивается ввод необходимых данных:

```
$ ./client message
```

```
[artem@asymmetriq lab_5]$ ./client hello!
SERVER IDENTIFIER: random_id
connection established
ANSWER FROM SERVER: hello![artem@asymmetriq lab_5]$
```

Рисунок 2. Подключения клиента, unix-socket

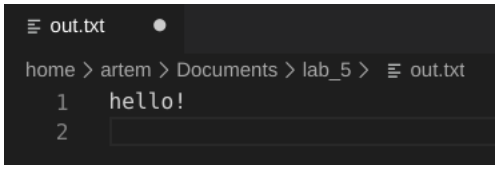

Рисунок 3. История переданных сообщений

2. Пример использования с IP-socket

Сервер аналогично запускается командой с одним аргументом – именем файла истории, а затем в зависимости от типа сокета запрашивает ввод необходимых данных:

```
$ ./server filename
```

```
[artem@asymmetriq lab_5]$ ./server out.txt
IP: 127.0.0.1
PORT: 2345
listening socket works on: 127.0.0.1:2345
waiting for connection
accepted from: 127.0.0.1:44098
connection established
```

Рисунок 4. Запуск сервера, IP-socket

Клиент запускается с одним аргументом – передаваемым сообщением, а затем в зависимости от типа сокета запрашивается ввод необходимых данных:

```
$ ./client message
```

```
[artem@asymmetriq lab_5]$ ./client wow
IP: 127.0.0.1
PORT: 2345
connection established
ANSWER FROM SERVER: wow[artem@asymmetriq lab_5]$
```

Рисунок 5. Подключение клиента, IP-socket

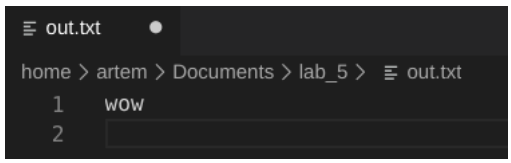

Рисунок 6. Рисунок 3. История переданных сообщений

ЗАКЛЮЧЕНИЕ

Вывод:
В ходе выполнения лабораторной работы удалось разработать приложение для работы с ip-socket и unix-socket, а также получить практические навыки.

ПРИЛОЖЕНИЕ

Исходный код сервера:

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <malloc.h>

#define MAX_CONNECTIONS 5 //кол-во подключений (можно сколько угодно)
#define DATA_BUF_SIZE 255 // размер буфера передаваемых данных
#define CHECK_MSG "connection_check" // это сообщение служит для проверки
соединения

#define SOCKET_INET // переменная для вида сокета

typedef uint8_t byte; // тайпдефы для удобства и читаемости
typedef uint16_t word;
typedef uint32_t dword;

void terminate(std::string err_msg) // остановить программу
{
        perror(err_msg.c_str());
        exit(-1);
}

unsigned int socket_init(int* list_socket_fd_p) // ф-ия инициализации сокета
{
        #ifdef SOCKET_INET // если инет
                if((*list_socket_fd_p = socket(AF_INET, SOCK_STREAM, 0)) < 0) //
создаем дескриптор сокета
                        terminate("list_sock_creating error"); // если не
создался, то ошибка, далее так везде

                struct sockaddr_in server_attrs; // объявляем структуру для
дальнейшей инициализации сокета
                struct in_addr server_addr;        // объявляем структуру для
записи адреса ip

                printf("IP: "); // просим ip
                char* ip_str;
                scanf("%ms", &ip_str);
                inet_aton(ip_str, &server_addr); // переводим адрес из строки в
long и записываем в server_add.s_addr

                server_attrs.sin_family = AF_INET; //в атрибуты пишем инет

                printf("PORT: "); // просим порт
                char* port_str;
                scanf("%ms", &port_str);
                server_attrs.sin_port = htons(atoi(port_str)); // порт из строчки
в unsigned short

                server_attrs.sin_addr.s_addr = server_addr.s_addr; // адрес
записываем в структуру

                if(bind(*list_socket_fd_p, (struct sockaddr*) &server_attrs,
sizeof(struct sockaddr_in)) < 0)
```

```
                        terminate("sock binding error"); // связываем дескриптор с
созданной нами структуре

                if(listen(*list_socket_fd_p, MAX_CONNECTIONS) < 0)
terminate("listen error"); // делаем его слушающим

                printf("listening socket works on: %s:%hu \n",
inet_ntoa(server_addr), ntohs(server_attrs.sin_port)); // пишем инфу о сервере

                return 0;

        #endif

        #ifdef SOCKET_UNIX // если юникс

                if((*list_socket_fd_p = socket(AF_UNIX, SOCK_STREAM, 0)) < 0) //
аналогично AF_UNIX
                        terminate("list_sock_creating error");

                struct sockaddr server_attrs_u; // аналогичная структура, только
для unix

                server_attrs_u.sa_family = AF_UNIX;

                printf("SERVER IDENTIFIER: "); // просим идентификатор
                char* id_str;
                scanf("%ms", &id_str);
                strcpy(server_attrs_u.sa_data, id_str);

                if(bind(*list_socket_fd_p, &server_attrs_u, sizeof(struct
sockaddr)) < 0)
                        terminate("sock binding error"); // связываем

                if(listen(*list_socket_fd_p, 5) < 0) terminate("listen error"); //
слушаем

                printf("listening socket works on: %s:\n", id_str); // пишем инфу
о сервере

                return 0;
        #endif

                return 1;
}

int main(int argc, char* argv[])
{
        if(argc != 2) terminate("wrong arguments"); // проверяем кол-ов аргументов

        int list_socket_fd; //дескриптор слушающего сокета
        int addrlen = sizeof(struct sockaddr); // длина структуры

        if(socket_init(&list_socket_fd) == 1) terminate("unable to init socket");
// инициализируем сокет

        byte exchange_buf[DATA_BUF_SIZE]; // буфер обмена
        bzero((char*) exchange_buf, DATA_BUF_SIZE); // заполняем его нулями

        FILE* output; // выходной файл с историей

        while(1)
        {
                int socket_copy_fd; // копия сокета для обмена информацией

                printf("waiting for connection\n");

                #ifdef SOCKET_INET // если инет то выводим информацию о клиенте в
соответствии со стрктурой sockadd_in
```

```
                        struct sockaddr_in client_attrs;
                        if((socket_copy_fd = accept(list_socket_fd, (struct
sockaddr*) &client_attrs, // принимем соединение
                                (socklen_t*)&addrlen)) < 0) terminate("unable to
establish connection");
                        printf("accepted from: %s:%hu \n", inet_ntoa((struct
in_addr)client_attrs.sin_addr), // пишем инфу о клиенте
                                ntohs(client_attrs.sin_port));
                #endif

                #ifdef SOCKET_UNIX  // если юникс
                        struct sockaddr client_attrs_u; // структура sockaddr
                        if((socket_copy_fd = accept(list_socket_fd,
&client_attrs_u,
                                (socklen_t*)&addrlen)) < 0) terminate("unable to
establish connection");
                        printf("accepted from: %d \n",
*((int*)client_attrs_u.sa_data));
                #endif

                printf("connection established\n");

                if(read(socket_copy_fd, exchange_buf, DATA_BUF_SIZE) < 0) //
читаем с сокета
                {
                        terminate("unable to read\n");
                        close(socket_copy_fd);
                        continue;
                }

                if((output = fopen(argv[1], "a")) == NULL) terminate("unable to
open file\n"); // открываем файл

                if(strcmp((char*)exchange_buf, CHECK_MSG) == 0) // если сообщение
проверки соединения
                {
                        printf("accessibility check requested\n");
                }
                else
                {
                        if(fputs((char*) exchange_buf, output) == EOF)
terminate("unable to write to file\n"); // пишем в файл
                        fputc('\n', output);
                }

                if(write(socket_copy_fd, exchange_buf,
strlen((char*)exchange_buf)) < 0) terminate("unable to write to socket");  // пишем
эхо-ответ на клиент

                fclose(output); // закрываем файл
                close(socket_copy_fd); //закрываем сокет
        }
}
```

Исходный код клиента:

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <malloc.h>
```

```
#define MAX_CONNECTIONS 5
#define DATA_BUF_SIZE 255

#define SOCKET_INET // аналогично как и на сервере

typedef uint8_t byte;
typedef uint16_t word;
typedef uint32_t dword;

void terminate(std::string err_msg)
{
        perror(err_msg.c_str());
        exit(1);
}

int sock_init(int* client_sock_fd_p)
{
        #ifdef SOCKET_INET // инет
                if((*client_sock_fd_p = socket(AF_INET, SOCK_STREAM, 0)) < 0)
terminate("unable to create socket"); //создаем дескриптор сокета

                struct sockaddr_in client_attrs; // структура для связывания с
сокетом
                struct in_addr client_addr;

                printf("IP: "); // заполняем структуру так же как и на сервере
                char* ip_str;
                scanf("%ms", &ip_str);
                inet_aton(ip_str, &client_addr);

                client_attrs.sin_family = AF_INET;

                printf("PORT: ");
                char* port_str;
                scanf("%ms", &port_str);
                client_attrs.sin_port = htons(atoi(port_str));

                client_attrs.sin_addr.s_addr = client_addr.s_addr;

                if(connect(*client_sock_fd_p, (struct sockaddr*) &client_attrs,
sizeof(struct sockaddr_in)) < 0) terminate("unable to connect");
                printf("connection established\n"); // подключаемся к серверу

                return 0;
        #endif

        #ifdef SOCKET_UNIX // юникс
                if((*client_sock_fd_p = socket(AF_UNIX, SOCK_STREAM, 0)) < 0)
terminate("unable to create socket");

                struct sockaddr client_attrs_u; // аналогично

                client_attrs_u.sa_family = AF_UNIX;
                printf("SERVER IDENTIFIER: ");
                char* id_str;
                scanf("%ms", &id_str);
                strcpy(client_attrs_u.sa_data, id_str);

                if(connect(*client_sock_fd_p, &client_attrs_u, sizeof(struct
sockaddr)) < 0) terminate("unable to connect");
                printf("connection established\n");

                return 0;
        #endif
}

int main(int argc, char* argv[])
```

```
{
        if(argc != 2) terminate("wrong arguments");


        int client_sock_fd;
        if(sock_init(&client_sock_fd) == 1) terminate("unable to init socket");

        byte exchange_buf[DATA_BUF_SIZE]; // буфер обмена
        bzero((char*) exchange_buf, DATA_BUF_SIZE); // нулями заполняем
        strcpy((char*) exchange_buf, argv[1]); // копируем строку в буфер

        if(write(client_sock_fd, exchange_buf, DATA_BUF_SIZE) < 0)
terminate("unable to write to socket");
        bzero((char*) exchange_buf, DATA_BUF_SIZE); // пишем буфер
        if(read(client_sock_fd, exchange_buf, DATA_BUF_SIZE) < 0) terminate("unable
to read\n"); // читаем сообщение от сервера
        printf("ANSWER FROM SERVER: %s", (char*)exchange_buf); // выводим его

        close(client_sock_fd); // закрываем сокет
}
```