



**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 № 4
по дисциплине «Безопасность систем баз данных»

Выполнил: Аббасалиев Э.Н.,
студент группы ИУ8-61

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

г. Москва
2020 г.

ОГЛАВЛЕНИЕ

ВСТУПЛЕНИЕ	3
1. РАЗЛИЧИЯ МЕЖДУ UNIX-SOCKET И IP-SOCKET	4
2. ПРИМЕР ИСПОЛЬЗОВАНИЯ	5
2.1 Server	5
2.2 Client	8
2.3 Пример использования	11
ЗАКЛЮЧЕНИЕ	12

ВСТУПЛЕНИЕ

Цель работы: реализовать приложение для работы с unix/ip-socket на C99/C++11.

Приложение должно уметь:

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

1. РАЗЛИЧИЯ МЕЖДУ UNIX-SOCKET И IP-SOCKET

Unix Domain Sockets — это технология передачи данных между процессами позикс ОС. То как эта передача производится похоже на IP Sockets, но только внешне, потому что не используется низлежащий протокол TCP/IP, эти данные не выходят за предел локального компьютера.

Internet Socket — это конечная точка в двунаправленном межпроцессном общении через сеть, основаной на протоколе TCP/IP. Процессы находятся на разных компьютерах в сети, при этом может быть вариант, когда процессы находятся на одном компьютере. Под этим термином также подразумевают API, которое предоставляет ОС для доступа к TCP/IP protocol stack.

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. Поэтому, если вы планируете взаимодействовать с процессами на одном хосте, это лучший вариант, чем IP-сокеты.

2. ПРИМЕР ИСПОЛЬЗОВАНИЯ

Для работы с сокетами было реализовано простое эхо-клиент-серверное приложение.

Для начала работы необходимо запустить сервер:

```
$ ./server <filename>
```

Затем необходимо запустить клиент:

```
$ ./client <message>
```

2.1

Server

```
home > elshan > lab05 > server.cpp > ...
1  #include <iostream>
2  #include <unistd.h>
3  #include <malloc.h>
4  #include <stdlib.h>
5  #include <stdio.h>
6  #include <stdint.h>
7  #include <string.h>
8  #include <sys/socket.h>
9  #include <sys/types.h>
10 #include <arpa/inet.h>
11 #include <netinet/in.h>
12
13 #define SOCKET_IN
14
15 int initialization_socket(int* init_socket)
16 {
17     #ifdef SOCKET_UNIX
18         struct sockaddr sockaddr_unix;
19         if ((*init_socket = socket(AF_UNIX, SOCK_STREAM, 0)) < 0)
20         {
21             std::string error = "error";
```

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

#define SOCKET_IN

int initialization_socket(int* init_socket)
{
    #ifdef SOCKET_UNIX
        struct sockaddr sockaddr_unix;
        if ((*init_socket = socket(AF_UNIX, SOCK_STREAM, 0)) < 0)
        {
```

```

        std::string error = "error";
        perror(error.c_str());
        exit(-1);
    }
    sockaddr_unix.sa_family = AF_UNIX;
    printf("server: ");
    char* id_string;
    scanf("%ms", &id_string);
    strcpy(sockaddr_unix.sa_data, id_string);
    if (bind(*init_socket, &sockaddr_unix, sizeof(struct sockaddr)) < 0)
    {
        std::string error = "bind error";
        perror(error.c_str());
        exit(-1);
    }
    if (listen(*init_socket, 5) < 0)
    {
        std::string error = "listen error";
        perror(error.c_str());
        exit(-1);
    }
    printf("socket works on: %s:\n", id_string);
    return 0;
#endif

#ifdef SOCKET_IN
    struct sockaddr_in sockaddr;
    struct in_addr addr;
    if ((*init_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        std::string error = "error";
        perror(error.c_str());
        exit(-1);
    }
    printf("ip: ");
    char* ip_string;
    scanf("%ms", &ip_string);
    inet_aton(ip_string, &addr);
    sockaddr.sin_family = AF_INET;
    printf("port: ");
    char* port_string;
    scanf("%ms", &port_string);
    sockaddr.sin_port = htons(atoi(port_string));
    sockaddr.sin_addr.s_addr = addr.s_addr;
    if (bind(*init_socket, (struct sockaddr*) &sockaddr, sizeof(struct
sockaddr_in)) < 0)
    {
        std::string error = "bind error";
        perror(error.c_str());
        exit(-1);
    }
    if (listen(*init_socket, 7) < 0)
    {
        std::string error = "listen error";
        perror(error.c_str());
        exit(-1);
    }
    printf("socket works: %s:%hu \n", inet_ntoa(addr), ntohs(sockaddr.sin_port));
    return 0;
#endif

    return 1;
}

int main(int argc, char* argv[])
{
    int descrip_list_socket;

```

```

int length_sockaddr = sizeof(struct sockaddr);
if (argc != 2)
{
    std::string error = "amiss arg";
    perror(error.c_str());
    exit(-1);
}
if (initilization_socket(&descrip_list_socket) == 1)
{
    std::string error = "impossible to init sock";
    perror(error.c_str());
    exit(-1);
}
uint8_t buf_swap[255];
bzero((char*) buf_swap, 255);
FILE* out;
while(1)
{
    int sock_copy;
    printf("wait, connection\n");
    #ifdef SOCKET_UNIX
        struct sockaddr client_info_unix;
        if((sock_copy = accept(descrip_list_socket, &client_info_unix,
(socklen_t*)&length_sockaddr)) < 0)
        {
            std::string error = "does not to install connect";
            perror(error.c_str());
            exit(-1);
        }
        printf("received from: %d \n", *((int*)client_info_unix.sa_data));
    #endif

    #ifdef SOCKET_IN
        struct sockaddr_in client_info;
        if ((sock_copy = accept(descrip_list_socket, (struct sockaddr*) &
client_info, (socklen_t*)& length_sockaddr)) < 0)
        {
            std::string error = "does not to install connection";
            perror(error.c_str());
            exit(-1);
        }
        printf("received from: %s:%hu \n", inet_ntoa((struct
in_addr)client_info.sin_addr), ntohs(client_info.sin_port));
    #endif

    printf("connection install\n");
    if ((out = fopen(argv[1], "a")) == NULL)
    {
        std::string error = "does not to open file\n";
        perror(error.c_str());
        exit(-1);
    }
    if(read(sock_copy, buf_swap, 255) < 0)
    {
        std::string error = "does not to read\n";
        perror(error.c_str());
        exit(-1);
        close(sock_copy);
        continue;
    }
    if (write(sock_copy, buf_swap, strlen((char*)buf_swap)) < 0)
    {
        std::string error = "does not to write to sock";
        perror(error.c_str());
        exit(-1);
    }
    7
    if(strcmp((char*)buf_swap, "connection") == 0)

```

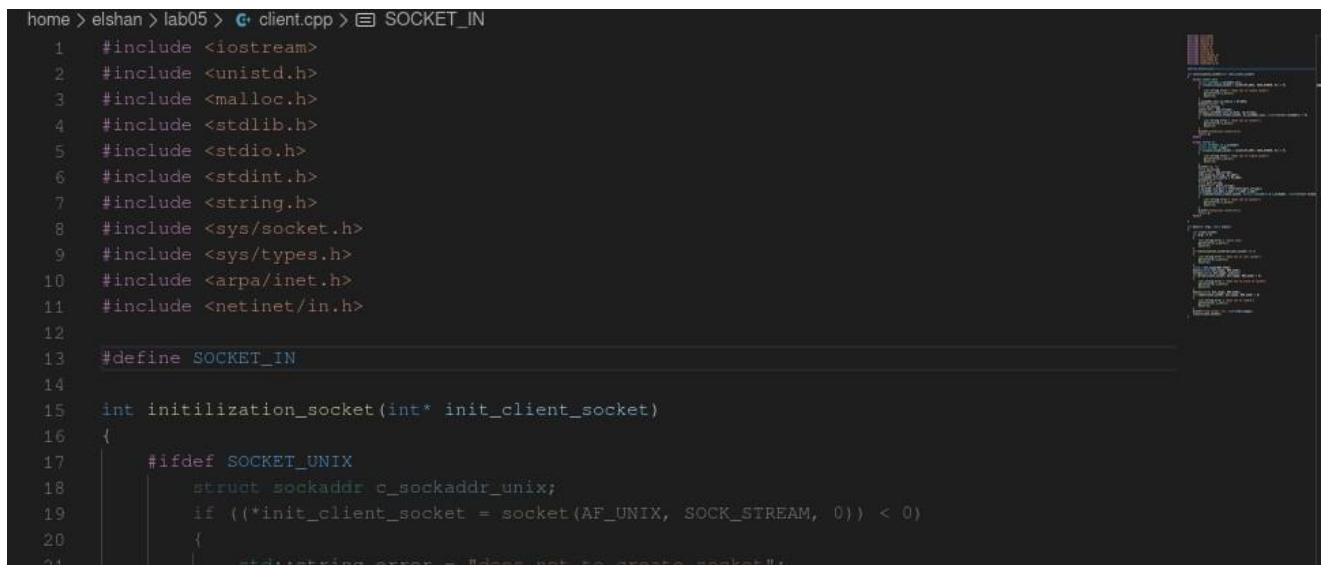
```

    {
        printf("availability\n");
    }
    else
    {
        if (fputs((char*)buf_swap, out) == EOF)
        {
            std::string error = "does not to write to file\n";
            perror(error.c_str());
            exit(-1);
        }
        fputc('\n', out);
    }
    fclose(out);
    close(sock_copy);
}
}

```

2.2

Client



```

home > elshan > lab05 > client.cpp > SOCKET_IN
1  #include <iostream>
2  #include <unistd.h>
3  #include <malloc.h>
4  #include <stdlib.h>
5  #include <stdio.h>
6  #include <stdint.h>
7  #include <string.h>
8  #include <sys/socket.h>
9  #include <sys/types.h>
10 #include <arpa/inet.h>
11 #include <netinet/in.h>
12
13 #define SOCKET_IN
14
15 int initialization_socket(int* init_client_socket)
16 {
17     #ifdef SOCKET_UNIX
18         struct sockaddr c_sockaddr_unix;
19         if ((*init_client_socket = socket(AF_UNIX, SOCK_STREAM, 0)) < 0)
20         {
21             std::string error = "does not to create socket";

```

```

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

```

```
#define SOCKET_IN
```

```

int initialization_socket(int* init_client_socket)
{
    #ifdef SOCKET_UNIX
        struct sockaddr c_sockaddr_unix;
        if ((*init_client_socket = socket(AF_UNIX, SOCK_STREAM, 0)) < 0)
        {
            std::string error = "does not to create socket";
            perror(error.c_str());

```



```

        exit(-1);
    }
    c_sockaddr_unix.sa_family = AF_UNIX;
    printf("server: ");
    char* id_string;
    scanf("%ms", &id_string);
    strcpy(c_sockaddr_unix.sa_data, id_string);
    if (connect(*init_client_socket, &c_sockaddr_unix, sizeof(struct sockaddr)) < 0)
    {
        std::string error = "does not to connect";
        perror(error.c_str());
        exit(-1);
    }
    printf("connection install\n");
    return 0;
#endif

#ifdef SOCKET_IN
    struct sockaddr_in c_sockaddr;
    struct in_addr c_addr;
    if ((*init_client_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        std::string error = "does not to create socket";
        perror(error.c_str());
        exit(-1);
    }
    printf("ip: ");
    char* ip_string;
    scanf("%ms", &ip_string);
    inet_aton(ip_string, &c_addr);
    c_sockaddr.sin_family = AF_INET;
    printf("port: ");
    char* port_string;
    scanf("%ms", &port_string);
    c_sockaddr.sin_port = htons(atoi(port_string));
    c_sockaddr.sin_addr.s_addr = c_addr.s_addr;
    if (connect(*init_client_socket, (struct sockaddr*) &c_sockaddr, sizeof(struct
sockaddr_in)) < 0)
    {
        std::string error = "does not to connect";
        perror(error.c_str());
        exit(-1);
    }
    printf("connection install\n");
    return 0;
#endif

}

int main(int argc, char* argv[])
{
    int client_socket;
    if (argc != 2)
    {
        std::string error = "amiss arg";
        perror(error.c_str());
        exit(-1);
    }
    if (initilization_socket(&client_socket) == 1)
    {
        std::string error = "does not to init socket";
        perror(error.c_str());
        exit(-1);
    }
    int8_t buf_swape[255];
    zero((char*) buf_swape, 255);
    strcpy((char*) buf_swape, argv[1]);

```

```

if (write(client_socket, buf_swape, 255) < 0)
{
    std::string error = "does not to write to socket";
    perror(error.c_str());
    exit(-1);
}
bzero((char*) buf_swape, 255);
if (read(client_socket, buf_swape, 255) < 0)
{
    std::string error = "does not to read\n";
    perror(error.c_str());
    exit(-1);
}
printf("from server: %s", (char*)buf_swape);
close(client_socket);
}

```

2.3 Пример использования

Для того, чтобы продемонстрировать работу приложений, необходимо запустить сервер. Запустим клиента и отправим сообщение, эхо-сервер ответит нам этим же сообщением. История передаваемых данных будет записываться в файл с адресом отправителя.

Сервер запускается командой с одним аргументом – именем файла истории:

\$./server filename

```
[elshan@spaton lab05]$ g++ -Wall -o server server.cpp
[elshan@spaton lab05]$ ./server example.txt
ip: 127.0.0.1
port: 1234
socket works: 127.0.0.1:1234
```

Клиент запускается с одним аргументом – передаваемым сообщением:

\$./client message

```
[elshan@spaton lab05]$ g++ -Wall -o client client.cpp
[elshan@spaton lab05]$ ./client Elshan
ip: 127.0.0.1
port: 1234
connection install
```

```
home > elshan > lab05 > example.txt
1 Elshan
```

ЗАКЛЮЧЕНИЕ

Вывод: в ходе выполнения лабораторной работы было создано эхо-клиент-серверное приложение, для примера работы с сокетами.