



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 «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа №6

По предмету: «Операционные системы»

Тема: Сокеты

Преподаватель: Рязанова Н.Ю.

Студент: Гасанзаде М.А.,

Группа: ИУ7-66Б

Москва, 2020 г.

ОГЛАВЛЕНИЕ

ЗАДАНИЕ 1	3
Листинг 1 – server.c	3
Листинг 2 – client.c	4
Результат работы программы	5
ЗАДАНИЕ 2	6
Листинг 3 – includes.h	7
Листинг 4 – client.c	7
Листинг 5 – server.c	8
Результат работы программы	10

ЗАДАНИЕ 1

Написать приложение по модели клиент-сервер, демонстрирующее взаимодействие параллельных процессов на отдельном компьютере с использованием сокетов в файловом пространстве имен: семейство - AF_UNIX, тип - SOCK_DGRAM. При демонстрации работы программного комплекса необходимо запустить несколько клиентов (не меньше 5) и продемонстрировать, что сервер обрабатывает обращения каждого запущенного клиента.

В процессе-сервере с помощью вызова socket() создается сокет семейства AF_UNIX с типом SOCK_DGRAM. С помощью системного вызова bind() происходит связка сокета с локальным адресом. Сервер блокируется на функции recv() и ждет сообщения от процессов клиентов.

В процессе-клиенте создается сокет семейства AF_UNIX с типом SOCK_DGRAM с помощью системного вызова socket(). С помощью функции sendto() отправляется сообщение процессу-серверу.

Листинг 1 – server.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#define SOCK_NAME "mysocket.s"

int sock_fd;

void close_sock(int sock_fd, char *name)
{
    close(sock_fd);
    unlink(name);
}

void sigint_handler(int signum)
{
    close_sock(sock_fd, SOCK_NAME);
    printf("\nSocket was closed due to ctrl+c!\n");
    printf("Server will be stopped.\n");
    exit(0);
}
```

```

int main()
{
    sock_fd = socket(AF_UNIX, SOCK_DGRAM, 0);
    if (sock_fd < 0)
    {
        perror("socket failed");
        return EXIT_FAILURE;
    }

    struct sockaddr srvr_name;
    srvr_name.sa_family = AF_UNIX;
    strcpy(srvr_name.sa_data, SOCK_NAME);
    if(bind(sock_fd,&srvr_name,
strlen(srvr_name.sa_data)+sizeof(srvr_name.sa_family)) < 0)
    {
        perror("bind failed");
        return EXIT_FAILURE;
    }

    signal(SIGINT, sigint_handler);
    printf("Server is listening.\nTo stop server press ctrl+c or send
\"stop\" via client.\n");

    char buf[100];

    while (strcmp(buf, "stop"))
    {
        int bytes = recv(sock_fd, buf, sizeof(buf), 0);
        if (bytes <= 0)
        {
            perror("recv failed");
            close_sock(sock_fd, SOCK_NAME);
            return EXIT_FAILURE;
        }
        buf[bytes] = 0;
        printf("Server read: %s\n", buf);
    }

    printf("Server stopped listening\n");
    close_sock(sock_fd, SOCK_NAME);
    printf("Socket closed\n");
    return 0;
}

```

Листинг 2 – client.c

```

#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#define SOCK_NAME "mysocket.s"

int main()
{
    int sock_fd = socket(AF_UNIX, SOCK_DGRAM, 0);
    if (sock_fd < 0)
    {
        perror("socket failed");
        return EXIT_FAILURE;
    }
}

```

```

struct sockaddr srvr_name;
srvr_name.sa_family = AF_UNIX;
strcpy(srvr_name.sa_data, SOCK_NAME);

char buf[100];
scanf("%99s", buf);
sendto(sock_fd, buf, strlen(buf), 0, &srvr_name,
strlen(srvr_name.sa_data) + sizeof(srvr_name.sa_family));

printf("Client sent: %s\n", buf);
return 0;
}

```

Результат работы программы

```

[wizard@Manjaro 1]$ ./server
Server is listening.
To stop server press ctrl+c or send "stop" via client.
Server read: Hello1
Server read: Heey!2
Server read: Here.client3
Server read: I'm4
Server read: And_last5

[wizard@Manjaro 1]$ ./client
Hello1
Client sent: Hello1

[wizard@Manjaro 1]$ ./client
Heey!2
Client sent: Heey!2

[wizard@Manjaro 1]$ ./client
Here.client3
Client sent: Here.client3

[wizard@Manjaro 1]$ ./client
I'm4
Client sent: I'm4

```

Рис. 1 – результат взаимодействия параллельных процессов
на отдельном компьютере

```

[wizard@Manjaro 1]$ ls -l mysocket.s
srwxr-xr-x 1 wizard wizard 0 мая 1 15:47 mysocket.s

```

Рис. 2 – Сокет в системе

```

[wizard@Manjaro 1]$ lsof -p 1334
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
servr 1334 wizard cwd DIR 8,1 4096 425688 /home/wizard/Desktop/1
servr 1334 wizard rtd DIR 8,1 4096 2 /
servr 1334 wizard txt REG 8,1 17280 425559 /home/wizard/Desktop/1/servr
servr 1334 wizard mem REG 8,1 2145592 271336 /usr/lib/libc-2.31.so
servr 1334 wizard mem REG 8,1 203000 271325 /usr/lib/ld-2.31.so
servr 1334 wizard 0u CHR 136,0 0t0 3 /dev/pts/0
servr 1334 wizard 1u CHR 136,0 0t0 3 /dev/pts/0
servr 1334 wizard 2u CHR 136,0 0t0 3 /dev/pts/0
servr 1334 wizard 3u unix 0x000000000049a01d0 0t0 20275 mysocket.s type=DGRAM

```

Рис. 3 – Информация про сервер

ЗАДАНИЕ 2

Написать приложение по модели клиент-сервер, осуществляющее взаимодействие параллельных процессов, которые выполняются на разных компьютерах. Для взаимодействия с клиентами сервер должен использовать мультиплексирование. Сервер должен обслуживать запросы параллельно запущенных клиентов. При демонстрации работы программного комплекса необходимо запустить несколько клиентов (не меньше 5) и продемонстрировать, что сервер обрабатывает обращения каждого запущенного клиента.

В процессе-сервере с помощью вызова socket() создается сокет семейства AF_INET с типом SOCK_STREAM. С помощью системного вызова bind() происходит связка сокета с адресом, прописанным в SOCKET_ADDRESS. С помощью вызова listen() сокету сообщается, что должны приниматься новые соединения. На каждой итерации цикла создается новый набор дескрипторов set. В него заносятся сокет сервера и сокеты клиентов с помощью функции FD_SET. После этого сервер блокируется на вызове функции select(), она возвращает управление, если хотя бы один из проверяемых сокетов готов к выполнению соответствующей операции. После выхода из блокировки, проверяется наличие новых соединений. При наличии таковых вызывается функция connectHandler(). В этой функции с помощью accept() принимается новое соединение, а также создается сокет, который записывается в массив файловых дескрипторов. Затем происходит обход массива дескрипторов, и, если дескриптор находится в наборе дескрипторов, то запускается функция clientHandeler(). В ней осуществляется считывание с помощью recv() и вывод сообщения от клиента. Если recv() возвращает ноль, то соединение было сброшено. В таком случае выводится сообщение о закрытии сокета.

В процессе-клиенте создается сокет семейства AF_INET с типом SOCK_STREAM с помощью системного вызова socket(). С помощью функции gethostbyname() доменный адрес преобразуется в сетевой и с его помощью

можно установить соединение, используя функцию `connect()`. Затем происходит отправка сообщений серверу.

Листинг 3 – `includes.h`

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/select.h>

#include <arpa/inet.h>
#include <netdb.h>

#define MSG_LEN 256
#define SOCK_ADDR "localhost"
#define SOCK_PORT 9999
```

Листинг 4 – `client.c`

```
#include "includes.h"

int main(void)
{
    // Establishing connection
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0)
    {
        perror("socket failed\n");
        return sock;
    }

    struct hostent* host = gethostbyname(SOCK_ADDR);
    if (!host)
    {
        perror("gethostbyname failed\n ");
        return EXIT_FAILURE;
    }

    struct sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(SOCK_PORT);
    addr.sin_addr = *((struct in_addr*) host->h_addr_list[0]);

    if (connect(sock, (struct sockaddr*) &addr, sizeof(addr)) < 0)
    {
        perror("connect failed\n");
        return EXIT_FAILURE;
    }

    // Sending messages
    int pid = getpid();

    char msg[MSG_LEN];
```

```

    for (int i = 0; i < 4; i++)
    {
        memset(msg, 0, MSG_LEN);
        sprintf(msg, "Hello #%d from %d", i, pid);

        if (send(sock, msg, strlen(msg), 0) < 0)
        {
            perror("send failed: ");
            return EXIT_FAILURE;
        }

        printf("[%d] sent msg #%d: %s\n", pid, i, msg);
        sleep(3);
    }

    printf("Client [%d] terminated.\n", pid);
    return 0;
}

```

Листинг 5 – server.c

```

#include "includes.h"

#include <signal.h>
#define MAX_CLIENTS 10

int clients[MAX_CLIENTS] = { 0 };

void newConnectionHandler(unsigned int fd)
{
    struct sockaddr_in addr;
    int addrSize = sizeof(addr);

    int incom = accept(fd, (struct sockaddr*) &addr, (socklen_t*)
&addrSize);
    if (incom < 0)
    {
        perror("accept failed");
        exit(EXIT_FAILURE);
    }

    int i;
    for (i = 0; i < MAX_CLIENTS; i++)
    {
        if (clients[i] == 0)
        {
            clients[i] = incom;
            break;
        }
    }

    printf("\nNew connection\nClient #%d: %s:%d\n\n",
        i, inet_ntoa(addr.sin_addr),
        ntohs(addr.sin_port));
}

void clientHandler(unsigned int fd, unsigned int client_id)
{
    char msg[MSG_LEN];
    memset(msg, 0, MSG_LEN);
}

```



```

    struct sockaddr_in addr;
    int addrSize = sizeof(addr);

    int recvSize = recv(fd, msg, MSG_LEN, 0);
    if (recvSize == 0)
    {
        getpeername(fd, (struct sockaddr*) &addr, (socklen_t*) &addrSize);
        printf("\nClient #%d disconnected\n\n", client_id);
        close(fd);
        clients[client_id] = 0;
    }
    else
    {
        msg[recvSize] = '\0';
        printf("Message from client #%d: %s\n", client_id, msg);
    }
}

int main(void)
{
    // Establishing connection
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0)
    {
        perror("socket failed\n");
        return EXIT_FAILURE;
    }

    struct sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(SOCK_PORT);
    addr.sin_addr.s_addr = INADDR_ANY;

    if (bind(sock, (struct sockaddr*) &addr, sizeof(addr)) < 0)
    {
        perror("bind failed\n");
        return EXIT_FAILURE;
    }

    if (listen(sock, 3) < 0)
    {
        perror("listen failed ");
        return EXIT_FAILURE;
    }

    // Handling requests
    printf("Server configured. Listening on port %d.\n", SOCK_PORT);

    while (1)
    {
        // Fill sockets
        int max_fd = sock;

        fd_set set;
        FD_ZERO(&set);
        FD_SET(sock, &set);

        for (int i = 0; i < MAX_CLIENTS; i++)
        {
            if (clients[i] > 0)
                FD_SET(clients[i], &set);
        }
    }
}

```

```

        max_fd = (clients[i] > max_fd) ? (clients[i]) : (max_fd);
    }

    // Wait for event in one of sockets
    struct timeval timeout = {15, 0}; // 15 sec
    int select_ret = select(max_fd + 1, &set, NULL, NULL, &timeout);
    if (select_ret == 0)
    {
        printf("\nServer closed connection by timeout.\n\n");
        return 0;
    }
    else if (select_ret < 0)
    {
        perror("select failed");
        return EXIT_FAILURE;
    }

    if (FD_ISSET(sock, &set))
        newConnectionHandler(sock);

    // Messages
    for (int i = 0; i < MAX_CLIENTS; i++)
    {
        int fd = clients[i];
        if ((fd > 0) && FD_ISSET(fd, &set))
            clientHandler(fd, i);
    }
    return 0;
}

```

Результат работы программы

The image displays four terminal windows showing the execution of a network program. The first window shows the server starting and listening on port 9999. The second window shows the server receiving connections from four clients (3021, 3022, 3023, 3024) and sending them messages. The third window shows the server receiving responses from the clients. The fourth window shows the server terminating the clients after a timeout.

```

[wizard@Manjaro 2]$ ./server
Server configured. Listening on port 9999.

New connection
Client #0: 127.0.0.1:54880

Message from client #0: Hello #0 from 3021

New connection
Client #1: 127.0.0.1:54882

Message from client #1: Hello #0 from 3022

New connection
Client #2: 127.0.0.1:54884

Message from client #2: Hello #0 from 3023

New connection
Client #3: 127.0.0.1:54886

Message from client #3: Hello #0 from 3024

New connection
Client #4: 127.0.0.1:54888

[wizard@Manjaro 2]$ ./client
[3025] sent msg #0: Hello #0 from 3025
[3025] sent msg #1: Hello #1 from 3025
[3025] sent msg #2: Hello #2 from 3025
[3025] sent msg #3: Hello #3 from 3025
Client [3025] terminated.
[wizard@Manjaro 2]$

[3022] sent msg #1: Hello #1 from 3022
[3022] sent msg #2: Hello #2 from 3022
[3022] sent msg #3: Hello #3 from 3022
Client [3022] terminated.
[wizard@Manjaro 2]$

[3021] sent msg #0: Hello #0 from 3021
[3021] sent msg #1: Hello #1 from 3021
[3021] sent msg #2: Hello #2 from 3021
[3021] sent msg #3: Hello #3 from 3021
Client [3021] terminated.
[wizard@Manjaro 2]$

[3023] sent msg #0: Hello #0 from 3023
[3023] sent msg #1: Hello #1 from 3023
[3023] sent msg #2: Hello #2 from 3023
[3023] sent msg #3: Hello #3 from 3023
Client [3023] terminated.
[wizard@Manjaro 2]$

Client [2979] terminated.
[wizard@Manjaro 2]$ ./client
[3024] sent msg #0: Hello #0 from 3024
[3024] sent msg #1: Hello #1 from 3024
[3024] sent msg #2: Hello #2 from 3024
[3024] sent msg #3: Hello #3 from 3024
Client [3024] terminated.
[wizard@Manjaro 2]$

```

Рис. 4.1 – Взаимодействие параллельных процессов в сети

```
Terminal - wizard@Manjaro:~/Desktop/2
File Edit View Terminal Tabs Help
Message from client #4: Hello #0 from 3025
Message from client #0: Hello #1 from 3021
Message from client #1: Hello #1 from 3022
Message from client #2: Hello #1 from 3023
Message from client #3: Hello #1 from 3024
Message from client #4: Hello #1 from 3025
Message from client #0: Hello #2 from 3021
Message from client #1: Hello #2 from 3022
Message from client #2: Hello #2 from 3023
Message from client #3: Hello #2 from 3024
Message from client #4: Hello #2 from 3025
Message from client #0: Hello #3 from 3021
Message from client #1: Hello #3 from 3022
Message from client #2: Hello #3 from 3023
Message from client #3: Hello #3 from 3024
Message from client #4: Hello #3 from 3025
Client #0 disconnected
Client #1 disconnected
Client #2 disconnected
```

Рис. 4.2 – Взаимодействие параллельных процессов в сети (сообщения передаваемые клиентами)

```
Client #0 disconnected
Client #1 disconnected
Client #2 disconnected
Client #3 disconnected
Client #4 disconnected
Server closed connection by timeout.
[wizard@Manjaro 2]$
```

Рис. 4.3 – Взаимодействие параллельных процессов в сети (завершение работы клиентов и сервера)

```
[wizard@Manjaro 1]$ lsof -p 1414
```

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE	NAME
server	1414	wizard	cwd	DIR	8,1	4096	426542	/home/wizard/Desktop/2
server	1414	wizard	rtd	DIR	8,1	4096	2	/
server	1414	wizard	txt	REG	8,1	17496	426952	/home/wizard/Desktop/2/server
server	1414	wizard	mem	REG	8,1	2145592	271336	/usr/lib/libc-2.31.so
server	1414	wizard	mem	REG	8,1	203000	271325	/usr/lib/ld-2.31.so
server	1414	wizard	0u	CHR	136,1	0t0	4	/dev/pts/1
server	1414	wizard	1u	CHR	136,1	0t0	4	/dev/pts/1
server	1414	wizard	2u	CHR	136,1	0t0	4	/dev/pts/1
server	1414	wizard	3u	IPv4	21419	0t0	TCP	*:distinct (LISTEN)

Рис. 5 – Информация о сервере

```
[wizard@Manjaro 2]$ lsof -i | grep -E LISTEN
```

server	1478	wizard	3u	IPv4	22641	0t0	TCP	*:distinct (LISTEN)
--------	------	--------	----	------	-------	-----	-----	---------------------

Рис. 6 – Информация об открытых для прослушивания портов