



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»

ОТЧЕТ

к лабораторной работе №6

По курсу: «Операционные системы»

На тему: «Сокеты»

Студентка ИУ7-65Б
Оберган Т.М

Преподаватель
Рязанова Н.Ю.

Москва, 2020 г.

Оглавление

Задание	3
Задание 1.....	3
Задание 2.....	3
Часть 1	4
Листинг	4
Результат работы	6
Часть 2	6
Листинг	7
Результат работы	11

Задание

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.

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.

Часть 1

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

Листинг 1 – client

```
#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;
}
```

Листинг 2 – server

```
#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;
    }

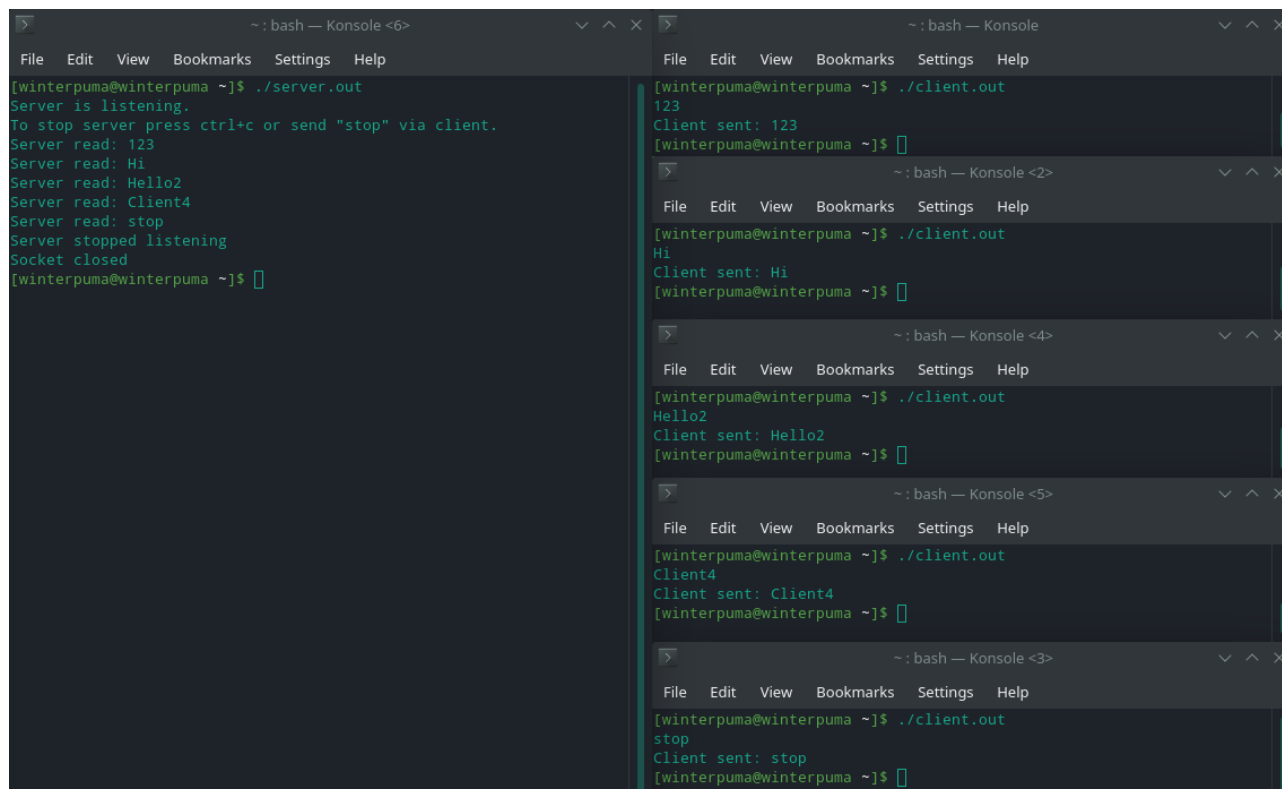
    signal(SIGINT, sigint_handler);
    printf("Server is listening.\nTo stop server press ctrl+c or send \"stop\" via client.\n");

    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;
}

```

Результат работы



```
[winterpuma@winterpuma ~]$ ./server.out
Server is listening.
To stop server press ctrl+c or send "stop" via client.
Server read: 123
Server read: Hi
Server read: Hello2
Server read: Client4
Server read: stop
Server stopped listening
Socket closed
[winterpuma@winterpuma ~]$

[winterpuma@winterpuma ~]$ ./client.out
123
Client sent: 123
[winterpuma@winterpuma ~]$

[winterpuma@winterpuma ~]$ ./client.out
Hi
Client sent: Hi
[winterpuma@winterpuma ~]$

[winterpuma@winterpuma ~]$ ./client.out
Hello2
Client sent: Hello2
[winterpuma@winterpuma ~]$

[winterpuma@winterpuma ~]$ ./client.out
Client4
Client sent: Client4
[winterpuma@winterpuma ~]$

[winterpuma@winterpuma ~]$ ./client.out
stop
Client sent: stop
[winterpuma@winterpuma ~]$
```

Рис. 1 – результат взаимодействия параллельных процессов
на отдельном компьютере

Часть 2

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 `clientHandler()`. 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

Листинг 3 – файл `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 – `server`

```
#include "includes.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;
    }

    // Checking for updates
    // Connections
    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;
}

```

Листинг 5 – client

```
#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();

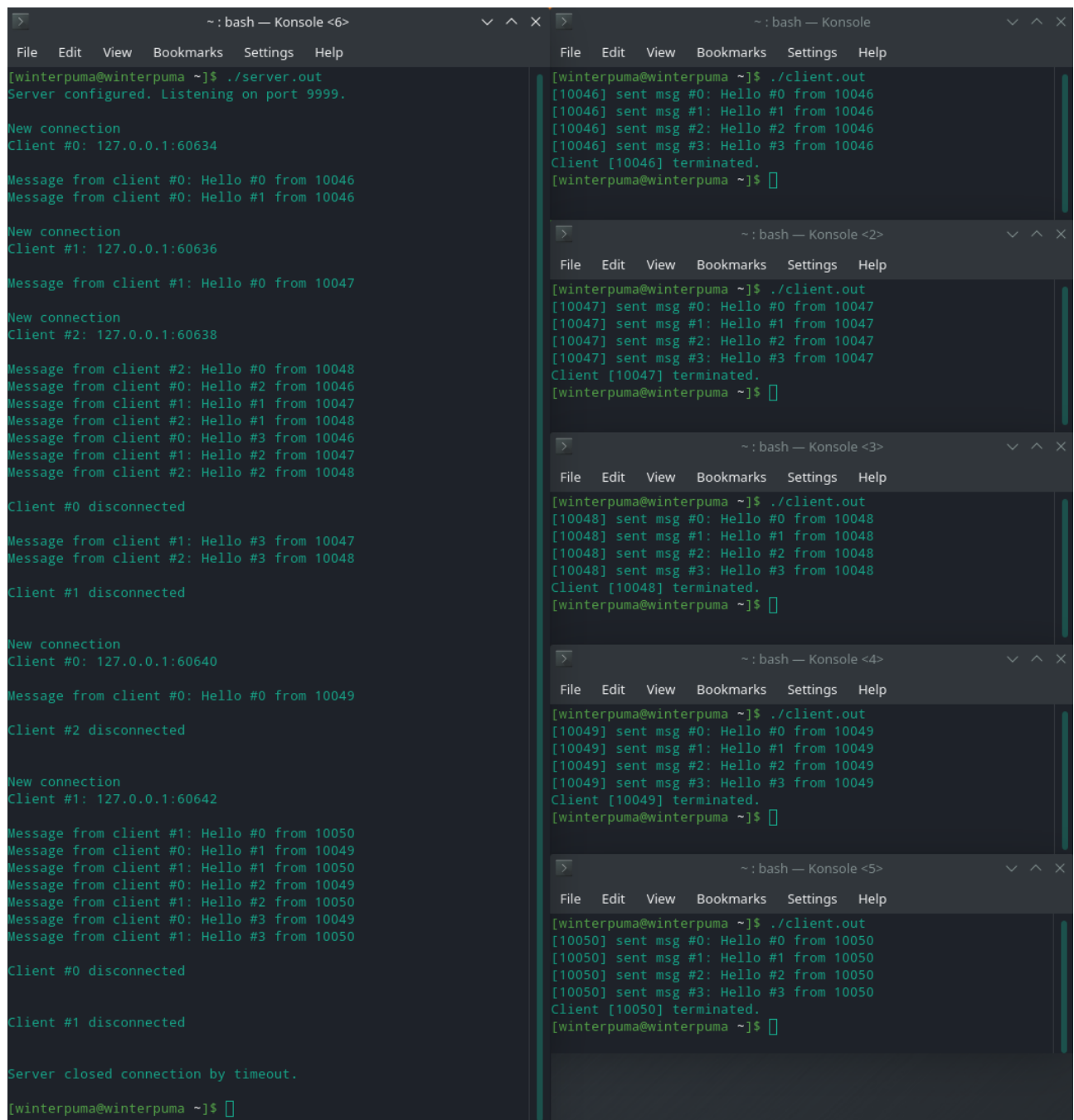
    char msg[MSG_LEN];
    for (int i = 0; i < 4; i++)
    {
        memset(msg, 0, MSG_LEN);
        sprintf(msg, "Hello #%d from %d", i, pid);

        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;
}
```

Результат работы



```
~: bash — Konsole <6>
File Edit View Bookmarks Settings Help

[winterpuma@winterpuma ~]$ ./server.out
Server configured. Listening on port 9999.

New connection
Client #0: 127.0.0.1:60634

Message from client #0: Hello #0 from 10046
Message from client #0: Hello #1 from 10046

New connection
Client #1: 127.0.0.1:60636

Message from client #1: Hello #0 from 10047

New connection
Client #2: 127.0.0.1:60638

Message from client #2: Hello #0 from 10048
Message from client #0: Hello #2 from 10046
Message from client #1: Hello #1 from 10047
Message from client #2: Hello #1 from 10048
Message from client #0: Hello #3 from 10046
Message from client #1: Hello #2 from 10047
Message from client #2: Hello #2 from 10048

Client #0 disconnected

Message from client #1: Hello #3 from 10047
Message from client #2: Hello #3 from 10048

Client #1 disconnected

New connection
Client #0: 127.0.0.1:60640

Message from client #0: Hello #0 from 10049

Client #2 disconnected

New connection
Client #1: 127.0.0.1:60642

Message from client #1: Hello #0 from 10050
Message from client #0: Hello #1 from 10049
Message from client #1: Hello #1 from 10050
Message from client #0: Hello #2 from 10049
Message from client #1: Hello #2 from 10050
Message from client #0: Hello #3 from 10049
Message from client #1: Hello #3 from 10050

Client #0 disconnected

Client #1 disconnected

Server closed connection by timeout.

[winterpuma@winterpuma ~]$
```

```
~: bash — Konsole
File Edit View Bookmarks Settings Help

[winterpuma@winterpuma ~]$ ./client.out
[10046] sent msg #0: Hello #0 from 10046
[10046] sent msg #1: Hello #1 from 10046
[10046] sent msg #2: Hello #2 from 10046
[10046] sent msg #3: Hello #3 from 10046
Client [10046] terminated.
[winterpuma@winterpuma ~]$
```

```
~: bash — Konsole <2>
File Edit View Bookmarks Settings Help

[winterpuma@winterpuma ~]$ ./client.out
[10047] sent msg #0: Hello #0 from 10047
[10047] sent msg #1: Hello #1 from 10047
[10047] sent msg #2: Hello #2 from 10047
[10047] sent msg #3: Hello #3 from 10047
Client [10047] terminated.
[winterpuma@winterpuma ~]$
```

```
~: bash — Konsole <3>
File Edit View Bookmarks Settings Help

[winterpuma@winterpuma ~]$ ./client.out
[10048] sent msg #0: Hello #0 from 10048
[10048] sent msg #1: Hello #1 from 10048
[10048] sent msg #2: Hello #2 from 10048
[10048] sent msg #3: Hello #3 from 10048
Client [10048] terminated.
[winterpuma@winterpuma ~]$
```

```
~: bash — Konsole <4>
File Edit View Bookmarks Settings Help

[winterpuma@winterpuma ~]$ ./client.out
[10049] sent msg #0: Hello #0 from 10049
[10049] sent msg #1: Hello #1 from 10049
[10049] sent msg #2: Hello #2 from 10049
[10049] sent msg #3: Hello #3 from 10049
Client [10049] terminated.
[winterpuma@winterpuma ~]$
```

```
~: bash — Konsole <5>
File Edit View Bookmarks Settings Help

[winterpuma@winterpuma ~]$ ./client.out
[10050] sent msg #0: Hello #0 from 10050
[10050] sent msg #1: Hello #1 from 10050
[10050] sent msg #2: Hello #2 from 10050
[10050] sent msg #3: Hello #3 from 10050
Client [10050] terminated.
[winterpuma@winterpuma ~]$
```

Рис. 2 – Взаимодействие параллельных процессов в сети