|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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**

|  |  |
| --- | --- |
| **Дисциплина** Операционные системы  **Тема** \_Сокеты\_  **Студент** \_Ильясов И. М.\_  **Группа** \_ИУ7-63Б\_  **Оценка (баллы)** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **Преподаватель** \_Рязанова Н. Ю.\_ |  |

Москва, 2020 г.

**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.

**Пример работы программы**

Ниже приведен пример работы программы из задания №1. На данном рисунке 6 процессов-клиентов отправляют процессу-серверу сообщение типа “Good day from client X!”, где X – идентификатор процесса-клиента.

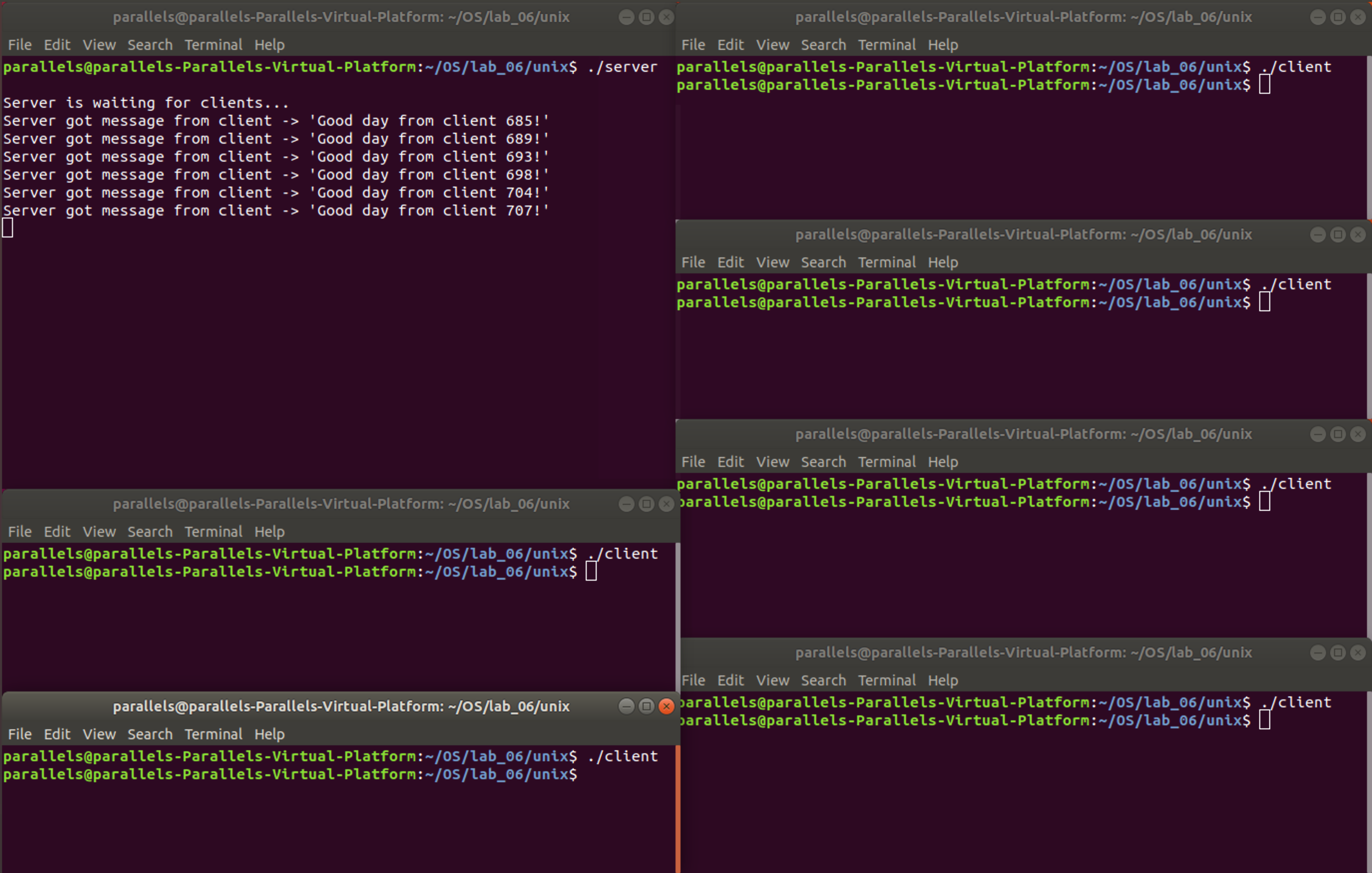


Рисунок 1. Пример работы программы из задания №1.

В процессе-сервере производится системный вызов *socket()*, при помощи которого создается сокет типа *SOCK\_DGRAM* типа *AF\_UNIX*. После этого производится системный вызов *bind()*, который привязывает сокет к локальному адресу. Затем процесс-сервер при помощи системного вызова *recv()* происходит блокировка процесса-сервера (процесс-сервер ожидает ввод сообщений от процессов-клиентов).

В процессах-клиентах также производится системный вызов *socket()*, который создает сокет типа *SOCK\_DGRAM* семейства *AF\_UNIX*. Отправка сообщений процессами-клиентами процессу-серверу производится вызовом функции *sendto()*.

**Листинг программы**

Ниже приведено 3 листинга – листинг с кодом программы клиента, листинг с кодом программы сервера и содержимое вспомогательного файла-хедера *info.h*.

**Листинг 1. Код программы клиента.**

#include "info.h"

int main()

{

int sock = socket(AF\_UNIX, SOCK\_DGRAM, 0);

if (sock < 0)

{

perror("Socket error\n");

return(sock);

}

struct sockaddr\_un server\_addr;

server\_addr.sun\_family = AF\_UNIX;

strcpy(server\_addr.sun\_path, NAME\_SOCKET);

char message[LEN\_MESSAGE];

sprintf(message, "Good day from client %d!", getpid());

sendto(sock, message, strlen(message), 0,

(struct sockaddr \*)&server\_addr, sizeof(server\_addr));

close(sock);

return 0;

}

**Листинг 2. Код программы сервера.**

#include "info.h"

int sock;

void catch\_sigint(int signum)

{

printf("\nSocket is closing (Ctrl+C signal was caught)!\n”);

close(sock);

unlink(NAME\_SOCKET);

return 0;

}

int main(void)

{

char message[LEN\_MESSAGE];

struct sockaddr\_un addr;

sock = socket(AF\_UNIX, SOCK\_DGRAM, 0);

if (sock < 0)

{

perror(“Socket error\n”);

return sock;

}

addr.sun\_family = AF\_UNIX;

strcpy(addr.sun\_path, NAME\_SOCKET);

if (bind(sock, (struct sockaddr \*)&addr, sizeof(addr)) < 0)

{

perror(“Bind error\n”);

close(sock);

unlink(NAME\_SOCKET);

return -1;

}

printf(“\nServer is waiting for clients…\n”);

signal(SIGINT, catch\_sigint);

for (;;)

{

int size = recv(sock, message, sizeof(message), 0);

if (size < 0)

{

perror(“Recv error\n”);

close(sock);

unlink(NAME\_SOCKET);

return size;

}

message[size] = 0;

printf(“Server got message from client -> ‘%s’\n”, message);

}

printf(“Socket is closing\n”);

close(sock);

unlink(NAME\_SOCKET);

return 0;

}

**Листинг 3. Содержимое вспомогательного файла *info.h*.**

#ifndef \_INFO\_H\_

#define \_INFO\_H\_

#define LEN\_MESSAGE 256

#define NAME\_SOCKET “socket.soc”

#include <stdio.h>

#include <stdlib.h>

#include <sys/un.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <time.h>

#include <string.h>

#include <time.h>

#include <signal.h>

#include <unistd.h>

#endif

**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.

**Пример работы программы**

Ниже приведен пример работы программы из задания №2. На данном рисунке 6 процессов-клиентов отправляют процессу-серверу сообщение типа “Message #X from Y!”, где X – номер сообщения от процесса-клиента, Y – идентификатор процесса-клиента.

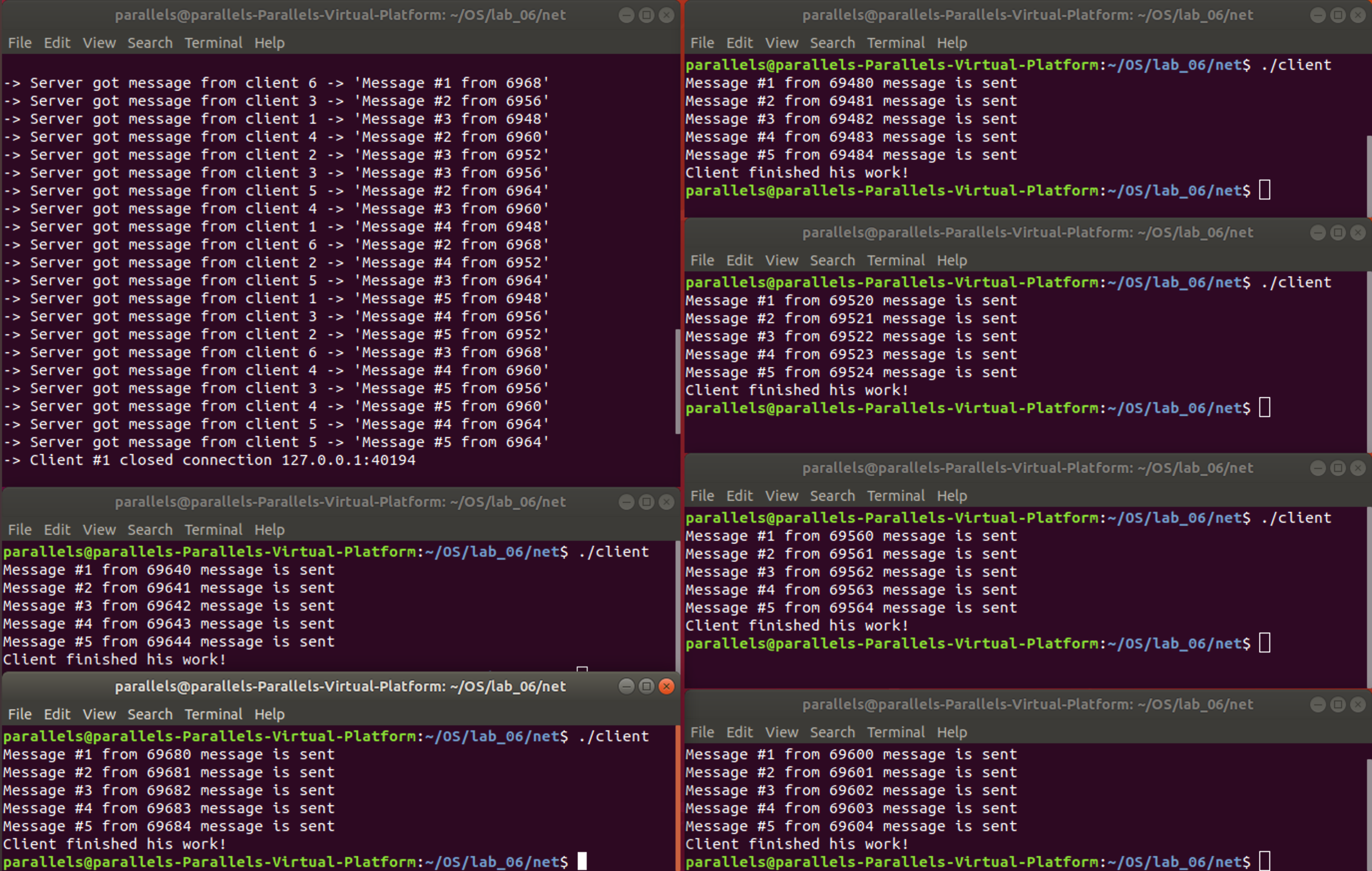
****

Рисунок 2. Пример работы программы из задания №2.

В процессе-сервере производится системный вызов *socket()*, при помощи которого создается сетевой сокет типа *SOCK\_STREAM* семейства *AF\_INET*. Далее системный вызов *bind()* связывает сокет с адресом, который прописанным во вспомогательном файле *info.h*. Затем процесс-сервер при помощи системного вызова *listen()* переводится в режим ожидания и ожидает запрос на соединение от процессов-клиентов. При каждом входе в цикл производится очистка набора дескрипторов при помощи макроса *FD\_ZERO* исоздание нового набора дескрипторов. В созданный набор заносится сокет сервера макросом *FD\_SET*. После этого процесс-сервер блокируется системным вызовом *select()*, который возвращает управление в функции при наличии запроса от процесса-клиента. Если блокировка снимается и управление возвращается процессу-серверу, он производит проверку на наличие новых соединений от процессов-клиентов и при их наличии вызывает функцию *connection\_handling()*. В этой функции принимается новое соединение при помощи вызова функции *accept()* и создается еще один сокет, который потом заносится в массив файловых дескрипторов. После этого осуществляется проход по массиву файловых дескрипторов и при условии если файловый дескриптор находится в наборе дескрипторов (проверка производится при помощи макроса *FD\_ISSET*), то вызывается функция *check\_info\_from\_clients*, в которой при помощи системного вызова *recv()* процессом-сервером считываются сообщения от процессов-клиентов. Далее при условии, что системный вызов *recv()* не вернул нулевое значение, выводятся сообщения, полученные процессом-сервером от процессов-клиентов. Если все же системный вызов *recv()* вернул нулевое значение, то получается, что соединение было сброшено, а значит этот сокет закрывается и удаляется из массива файловых дескрипторов.

В процессах-клиентах производится системный вызов *socket()*, при помощи которого создается сетевой сокет типа *SOCK\_STREAM* семейства *AF\_INET*. Функция *gethostbyname()* производит преобразование доменный адрес процесса-сервера в сетевой адрес. Далее вызывается функция *connect()*, которая устанавливает соединение с сокетом процесса-сервера, и в цикле при помощи системного вызова *send()* производится отправка сообщений процессами-клиентами сообщений процессу-серверу.

**Листинг программы**

Ниже приведено 3 листинга – листинг с кодом программы клиента, листинг с кодом программы сервера и содержимое вспомогательного файла-хедера *info.h*.

**Листинг 1. Код программы клиента.**

#include "info.h"

int main()

{

int sock = socket(AF\_INET, SOCK\_STREAM, 0);

if (sock < 0)

{

perror("Socket error\n");

exit(-1);

}

struct hostent\* host = gethostbyname(SOCKET\_ADDRESS);

if (!host)

{

perror("Getting host by name error\n");

close(sock);

exit(-1);

}

struct sockaddr\_in server\_addr;

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(SOCKET\_PORT);

server\_addr.sin\_addr = \*((struct in\_addr\*) host->h\_addr\_list[0]);

if (connect(sock, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) < 0)

{

perror("Connect error\n");

close(sock);

return -1;

}

char message[LEN\_MESSAGE];

for (int i = 0; i < QNT\_MESSAGE; i++)

{

memset(message, 0, LEN\_MESSAGE);

sprintf(message, "Message #%d from %d", i + 1, getpid());

printf("%s", message);

if (send(sock, message, strlen(message), 0) < 0)

{

perror("Send error\n");

close(sock);

return -1;

}

printf("\%d message is sent\n", i);

sleep(rand() % 5 + 1);

}

printf("Client finished his work!\n");

close(sock);

return 0;

}

**Листинг 2. Код программы сервера.**

#include "info.h"

void connection\_closing();

int clients[MAX\_CLIENTS] = {0};

int soc;

int connection\_handling()

{

struct sockaddr\_in client\_addr;

int client\_size = sizeof(client\_addr);

int client\_socket = accept(soc, (struct sockaddr\*)&client\_addr, (socklen\_t\*)&client\_size);

if (client\_socket < 0)

{

perror("Accept error\n");

connection\_closing();

close(soc);

return -1;

}

printf("\n\n-> Server got new connection:\nip = %s:%d\n",inet\_ntoa(client\_addr.sin\_addr), ntohs(client\_addr.sin\_port));

for (int i = 0; i < MAX\_CLIENTS; i++)

{

if (clients[i] == 0)

{

clients[i] = client\_socket;

printf("It is client #%d\n", i+1);

return 0;

}

}

return -1;

}

void check\_info\_from\_clients(int client\_sock , int client\_index)

{

char message[LEN\_MESSAGE];

struct sockaddr\_in client\_addr ;

int addr\_size = sizeof(client\_addr);

int message\_size = recv(client\_sock, message, LEN\_MESSAGE, 0);

if (message\_size == 0)

{

getpeername(client\_sock, (struct sockaddr\*)&client\_addr, (socklen\_t\*)&addr\_size);

printf("\n-> Client #%d closed connection %s:%d\n", client\_index + 1,

inet\_ntoa(client\_addr.sin\_addr), ntohs(client\_addr.sin\_port));

close(client\_sock);

clients[client\_index] = 0;

}

else

{

message[message\_size] = '\0';

printf ("\n-> Server got message from client %d -> '%s'", client\_index + 1, message);

}

}

void connection\_closing()

{

for (int i = 0; i < MAX\_CLIENTS; i++)

{

if (clients[i] != 0)

{

close(clients[i]);

}

}

}

int main(void)

{

soc = socket(AF\_INET, SOCK\_STREAM, 0);

if (soc < 0)

{

perror("Socket error\n");

return -1;

}

struct sockaddr\_in addr;

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(SOCKET\_PORT);

addr.sin\_addr.s\_addr = INADDR\_ANY;

if (bind(soc, (struct sockaddr\*)&addr, sizeof(addr)) < 0)

{

perror ("Bind error\n");

close(soc);

return -1;

}

if (listen(soc, 3) < 0)

{

perror("Listen error\n");

close(soc);

return -1;

}

printf("Server launched on ip %s:%d\n", inet\_ntoa(addr.sin\_addr), ntohs(addr.sin\_port));

for (;;)

{

fd\_set fds;

int max\_fd;

FD\_ZERO(&fds);

FD\_SET(soc, &fds);

max\_fd = soc;

for ( int i = 0; i < MAX\_CLIENTS; i++)

{

if (clients[i] > 0)

{

FD\_SET(clients[i], &fds);

}

if (clients[i] > max\_fd)

{

max\_fd = clients[i];

}

}

if (select(max\_fd + 1, &fds , NULL, NULL, NULL) < 0)

{

perror("Select error\n");

connection\_closing();

close(soc);

return -1;

}

if (FD\_ISSET(soc, &fds))

{

if (connection\_handling() < 0)

{

perror("No space for new connections\n");

connection\_closing();

close(soc);

return -1;

}

}

for (int i = 0; i < MAX\_CLIENTS; i++)

{

if ((clients[i] > 0) && FD\_ISSET(clients[i], &fds))

{

check\_info\_from\_clients(clients[i], i);

}

}

}

connection\_closing();

close(soc);

return 0;

}

**Листинг 3. Содержимое вспомогательного файла *info.h*.**

#ifndef \_INFO\_H\_

#define \_INFO\_H\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <signal.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netdb.h>

#define LEN\_MESSAGE 128

#define MAX\_CLIENTS 10

#define SOCKET\_ADDRESS "localhost"

#define SOCKET\_PORT 6951

#define QNT\_MESSAGE 5

#endif