|  |
| --- |
| «Санкт-Петербургский государственный электротехнический университет |
| «ЛЭТИ» им. В.И.Ульянова(Ленина)» |
| (СПбГЭТУ «ЛЭТИ») Кафедра вычислительной техники |
|  |
| **Отчет**  **по лабораторным работам № 11**  **по дисциплине** **«Организация процессов и программирование в среде Linux»** |
| Тема: «Работа с сокетами» |
|  |

|  |
| --- |
|  |

|  |  |
| --- | --- |
| Студент гр. 9307 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_Брызгалова Е. А. |
|  |  |
|  |  |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Разумовский Г. В. |

Оглавление

[Цель работы: 2](#_Toc118034666)

[Задание 2](#_Toc118034667)

[Распечатки входных и выходного файлов. 2](#_Toc118034668)

[Текст программы main.cpp 3](#_Toc118034669)

[Вывод 6](#_Toc118034670)

Цель работы:

Знакомство с сокетами, обеспечивающими обмен информацией между процессами.

Задание

1) Написать две программы (сервер и клиент) , которые обмениваются сообщениями через потоковые сокеты. Клиенты проверяют возможность соединения с сервером и в случае отсутствия соединения или истечения времени ожидания отправки сообщения завершают работу. После соединения с сервером они генерируют случайную последовательность чисел и выводят ее на экран, а затем отсылают серверу. Сервер в течение определенного времени ждет запросы от клиентов и в случае их отсутствия завершает работу. При поступлении запроса от клиента сервер порождает обслуживающий процесс, который принимает последовательность чисел, упорядочивает ее и выводит на экран, а затем отсылает обратно клиенту и завершают работу. Клиент полученную последовательность выводит на экран и заканчивает свою работу.

2) Откомпилировать все программы и запустить их

Скриншоты работы каждой программы.

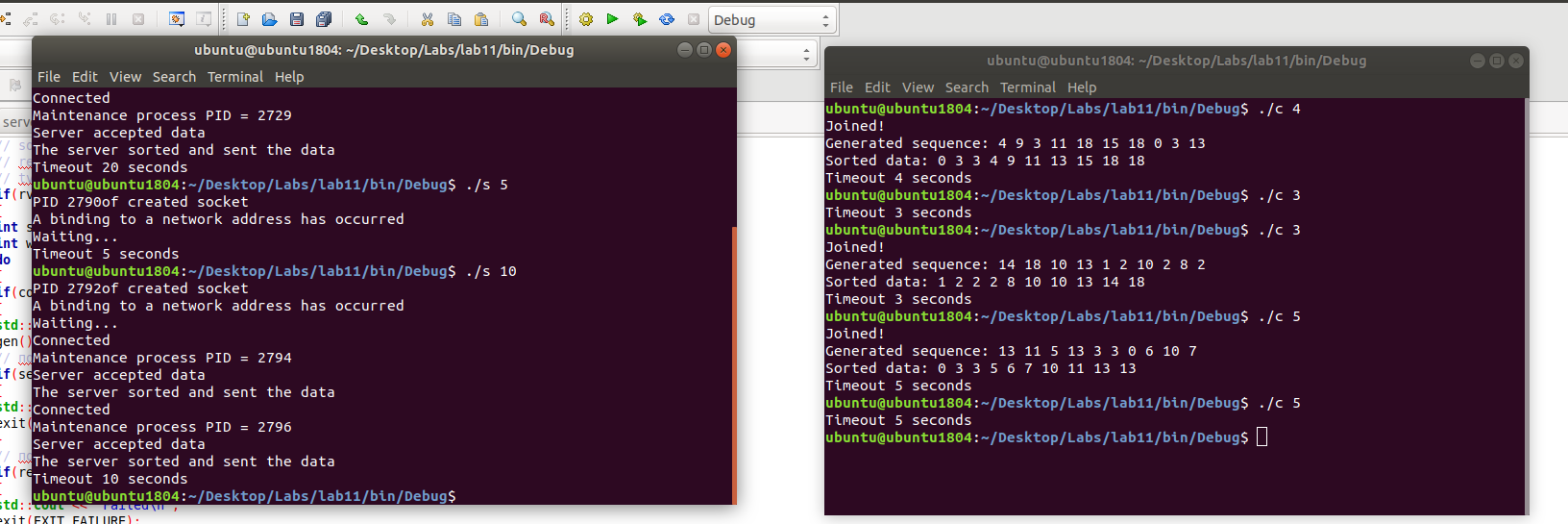


Рисунок 1. Работа програмы

# **Текст программы server**

#include <iostream>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <fcntl.h>

#include <unistd.h>

int sock, attached\_socket;

int reply[10];//

int timeout;

int cmp(const void \*a, const void \*b);

void service\_process();

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

{

struct sockaddr\_in server, client;//Для домена AF\_INET адрес задается следующей структурой

int rv;

struct timeval tv;

timeout = atoi(argv[1]);

tv.tv\_sec = timeout; // установка timeout

tv.tv\_usec = 0;

// создание сокета

// AF\_INET - домен для взаимодействия через сеть по протоколу TCP/IP

// SOCK\_STREAM - потоковый сокет

// 0 - автоматический выбор протокола (для потокового IPPROTO\_TCP)

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

if (sock<0)

{

std::cout << "Error socket creation\n";

return 1;

}

std::cout << "PID " << getpid() << "of created socket\n";

server.sin\_family = AF\_INET; // тип домена - – значение AF\_INET

// IP-адрес в сетевом порядке байт

server.sin\_addr.s\_addr = INADDR\_ANY; // сокет будет связан со всеми интерфейсами локального хоста(0.0.0.0)

server.sin\_port = htons(3434); // Номер порта в сетевом порядке байт, старший байт передавать первым

// если 0, то система самостоятельно выберет номер порта, первые 1024 порта зарезервированы

// привязка сокета к адресу и номеру порта

// sock - дескриптор сокета

// server - cтруктура

if(bind(sock,(struct sockaddr \*) &server , sizeof(server)) < 0)//(дескриптор сокета,указатель на структуру с адресом,длина структуры)

{

std::cout << "can't bind\n";

exit(2);

}

std::cout << "A binding to a network address has occurred\n";

// перевод сокета в пассивное ожидание (создаем очередь)

// sock - дескриптор сокета сервера

// 5 - длина очереди

listen(sock, 5);

std::cout << "Waiting...\n";

fd\_set readfds; // набор дескрипторов на чтение

while(1)

{

FD\_ZERO(&readfds); // очистка набора на чтение

FD\_SET(sock, &readfds); // добавление дескриптора в набор

// определение состояния сокета

// sock+1 - кол-во опрашиваемых дескрипторов сокетов

// readfds набор дескрипторов, которые следует проверять на готовность к чтению

// tv - timeout

rv = select(sock+1, &readfds, NULL, NULL, &tv);//Значением функции является число сокетов, от которых поступила информация

if(rv > 0)

{

// создание нового присоединенного сокета

// sock - дескриптор сокета сервера

// client - содержит указатель на структуру с адресом сервера

attached\_socket = accept(sock, (struct sockaddr \*)&client,(socklen\_t\*)&client);

if(attached\_socket > 0){//Возвращает дескриптор сокета для обмена с клиентом, или -1 в случае ошибки.

std::cout << "Connected\n";

service\_process();

}

}

else

if(rv == 0)

{

std::cout << "Timeout " << timeout << " seconds\n";

break;

}

tv.tv\_sec = timeout;

}

// закрытие принимающего сокета

close(sock);

return 0;

}

void service\_process()

{

// порождение обслуживающего потока

if (fork() == 0)

{

printf("Maintenance process PID = %d\n", getpid());

// закрытие принимающего сокета

close(sock);

// ожидание получения информации. чтение информации

if(recv(attached\_socket, reply, MSG\_LEN, 0) > 0)

{

// обработка информации

printf("Server accepted data\n");

sort(reply, 10, cmp);

printf("The server sorted and sent the data\n");

}

// отправка ответа

send(attached\_socket, reply, 40, 0);

// завершение соединения

close(attached\_socket);

// окончание работы

exit(EXIT\_SUCCESS);

}

}

int cmp(const void \*a, const void \*b)

{

return \*(int\*)a - \*(int\*)b;

}

# **Текст программы reader**

#include <iostream>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <fcntl.h>

#include <unistd.h>

int sock, attached\_socket;

int reply[MSG\_LEN];//

int timeout;

int cmp(const void \*a, const void \*b);

void service\_process();

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

{

struct sockaddr\_in server, client;//Для домена AF\_INET адрес задается следующей структурой

int rv;

struct timeval tv;

timeout = atoi(argv[1]);

tv.tv\_sec = timeout; // установка timeout

tv.tv\_usec = 0;

// создание сокета

// AF\_INET - домен для взаимодействия через сеть по протоколу TCP/IP

// SOCK\_STREAM - потоковый сокет

// 0 - автоматический выбор протокола (для потокового IPPROTO\_TCP)

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

if (sock<0)

{

std::cout << "Error socket creation\n";

return 1;

}

std::cout << "PID " << getpid() << "of created socket\n";

server.sin\_family = AF\_INET; // тип домена - – значение AF\_INET

// IP-адрес в сетевом порядке байт

server.sin\_addr.s\_addr = INADDR\_ANY; // сокет будет связан со всеми интерфейсами локального хоста(0.0.0.0)

server.sin\_port = htons(3434); // Номер порта в сетевом порядке байт, старший байт передавать первым

// если 0, то система самостоятельно выберет номер порта, первые 1024 порта зарезервированы

// привязка сокета к адресу и номеру порта

// sock - дескриптор сокета

// server - cтруктура

if(bind(sock,(struct sockaddr \*) &server , sizeof(server)) < 0)//(дескриптор сокета,указатель на структуру с адресом,длина структуры)

{

std::cout << "can't bind\n";

exit(2);

}

std::cout << "A binding to a network address has occurred\n";

// перевод сокета в пассивное ожидание (создаем очередь)

// sock - дескриптор сокета сервера

// 5 - длина очереди

listen(sock, 5);

std::cout << "Waiting...\n";

fd\_set readfds; // набор дескрипторов на чтение

while(1)

{

FD\_ZERO(&readfds); // очистка набора на чтение

FD\_SET(sock, &readfds); // добавление дескриптора в набор

// определение состояния сокета

// sock+1 - кол-во опрашиваемых дескрипторов сокетов

// readfds набор дескрипторов, которые следует проверять на готовность к чтению

// tv - timeout

rv = select(sock+1, &readfds, NULL, NULL, &tv);//Значением функции является число сокетов, от которых поступила информация

if(rv > 0)

{

// создание нового присоединенного сокета

// sock - дескриптор сокета сервера

// client - содержит указатель на структуру с адресом сервера

attached\_socket = accept(sock, (struct sockaddr \*)&client,(socklen\_t\*)&client);

if(attached\_socket > 0){//Возвращает дескриптор сокета для обмена с клиентом, или -1 в случае ошибки.

std::cout << "Connected\n";

service\_process();

}

}

else

if(rv == 0)

{

std::cout << "Timeout " << timeout << " seconds\n";

break;

}

tv.tv\_sec = timeout;

}

// закрытие принимающего сокета

close(sock);

return 0;

}

void service\_process()

{

// порождение обслуживающего потока

if (fork() == 0)

{

printf("Maintenance process PID = %d\n", getpid());

// закрытие принимающего сокета

close(sock);

// ожидание получения информации. чтение информации

if(recv(attached\_socket, reply, MSG\_LEN, 0) > 0)

{

// обработка информации

printf("Server accepted data\n");

qsort(reply, SEQUENCE\_LEN, sizeof(int), cmp);

printf("The server sorted and sent the data\n");

}

// отправка ответа

send(attached\_socket, reply, SEQUENCE\_LEN\*4, 0);

// завершение соединения

close(attached\_socket);

// окончание работы

exit(EXIT\_SUCCESS);

}

}

int cmp(const void \*a, const void \*b)

{

return \*(int\*)a - \*(int\*)b;

}

Вывод

В процессе выполнения лабораторной работы познакомились с сокетами, обеспечивающими обмен информацией между процессами.