Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и Информационных технологий

Кафедра Информатики

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4**

Преподаватель \_\_\_\_\_\_\_\_ Пересунько П. В.

подпись, дата инициалы, фамилия

Студент КИ15–17Б, 031510065 \_\_\_\_\_\_\_\_\_\_\_ Радионов Т.В.

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2016

**1 Цель работы**

**2 Исходные тексты программ**

/\*! \file server.c

\* \brief Файл-сервер

\* \details Имитация сервера, который принимает сообщения от клиента

\* и обрабатывает данные. Завершает работу при получении сообщения "exit"

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

/\*! \brief Обработка сообщения клиента

\* \details Считывает сообщение из сокета, обрабатывает данные

\* и выводит на экран.

\* \param clientSocket Сокет

\* \return 0 - успешное выполнение программы,

\* EXIT\_FAILURE - какая либо ошибка

\*/

int server(int clientSocket)

{

for (int step = 0; step < 2; step++)

{

int length = 0;

char\* text = NULL;

// Считываем длину сообщение: если возвращается 0, то клиент прервал сессию

if (recv(clientSocket, &length, sizeof(length), 0) == 0)

return 0;

// Выделяем память для text и заполняем "0"

text = (char\*)malloc(length);

bzero(text, length);

// Считываем текст и выводим его

recv(clientSocket, text, length, 0);

// Если клиент отправил сообщение "exit", то завершаем работу сервера

if (0 == strcmp(text, "exit"))

{

// Освобождаем память

free(text);

return 0;

}

switch (step)

{

int \*array;

int numOfElements = 0;

case 0:

{

// Подсчет элементов в пришедшей строке массива чисел

for (int i = 0; i < length; i++)

{

if (text[i] == ' ' || text[i] == '.')

{

numOfElements++;

}

}

// Выделение памяти под массив

array = (int\*)malloc(numOfElements\*sizeof(int));

// Разбиение text на элементы и помещение их в массив

char\* buffer = strtok(text, " ");

int i = 0;

printf("message 1:\n");

while (buffer != NULL)

{

array[i] = atoi(buffer);

printf("a[%d] = %d\n", i, array[i]);

buffer = strtok(NULL, " .");

i++;

}

free(buffer);

break;

}

case 1:

{

int key;

key = atoi(text);

printf("message 2:\n");

printf("key = %d\n", key);

printf("searching element...\n");

// Алгоритм бинарного поиска элемента по ключу

int l = 0;

int u = numOfElements;

int m;

while (l <= u)

{

m = (l + u) / 2;

if (array[m] < key)

{

l = m + 1;

}

if (array[m] > key)

{

u = m - 1;

}

if (array[m] == key)

{

break;

}

m = -1;

}

printf("Позиция искомого элемента: %d\n", m);

numOfElements = 0;

default:

break;

}

}

// Освобождаем память

free(text);

}

printf("===============================\n");

return 0;

}

/\*! \brief Главная функция

\* \param argc количество аргументов

\* \param argv массив системных переменных

\* \return 0 - успешное выполнение программы,

\* EXIT\_FAILURE - какая либо ошибка

\*/

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

{

printf("start Server...");

int socketFileDescriptor = -1;

int portNumber = 3000;

struct sockaddr\_in name;

int clientSentQuitMessage;

// Создаем сокет

socketFileDescriptor = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

int i = 1;

setsockopt(socketFileDescriptor, SOL\_SOCKET, SO\_REUSEADDR,

(const char \*)&i, sizeof(i));

bzero((char \*)&name, sizeof(name));

// Индетификация сервера

name.sin\_family = AF\_INET;

name.sin\_port = htons((u\_short)portNumber);

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

if (-1 == bind(socketFileDescriptor, (const struct sockaddr \*)&name, sizeof(name)))

{

perror("bind "); // !!! заменить на fprintf(stderr, "bind error\n");

close(socketFileDescriptor);

exit(1);

}

//"Прослушивание" соединения

if (-1 == listen(socketFileDescriptor, 5))

{

perror("listen "); // !!! заменить на fprintf(stderr, "listen error\n");

close(socketFileDescriptor);

exit(1);

}

/\* Принимать соединения через функцию server(),

обрабатываязапрос клиента.

Повторно поулчать сообщения до тех пор, пока Клиент

не отправит "exit". \*/

struct sockaddr\_in clientName;

socklen\_t clientNameLength = sizeof(clientName);

int clientSocketFileDescriptor;

// Разрешение соединения

clientSocketFileDescriptor = accept(socketFileDescriptor,

(struct sockaddr \*)&clientName,

&clientNameLength);

printf(" Success!\n");

// Удержание соединения, пока не получим сообщении о конце сессии

do

{

clientSentQuitMessage = server(clientSocketFileDescriptor);

} while (!clientSentQuitMessage);

// Завершение соединения

close(clientSocketFileDescriptor);

// Разъединение и закрытие сокета

shutdown(socketFileDescriptor, SHUT\_RD);

close(socketFileDescriptor);

return 0;

}

/\*! \file client.c

\* \brief Файл-клиент

\* \details Имитация клиента, который шифрует сообщения, вводимые пользователем

\* и посылает через сокет на сервер. Завершает работу при вводе в любое время "exit"

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

/\*! \brief Запись текста в сокет

\* \details Запись текста, получаемого от фалового дескриптора.

\* \param socketFileDescriptor Дескриптор

\* \param text Сообщение

\*/

void WriteText(int socketFileDescriptor, const char\* text)

{

// Записываем байты, включая "\0" (память для text)

int length = strlen(text) + 1;

send(socketFileDescriptor, &length, sizeof(length), 0);

// Записываем сообщение

send(socketFileDescriptor, text, length, 0);

}

/\*! \brief Главная функция

\* \param argc количество аргументов

\* \param argv массив системных переменных

\* argv[0] - имя программы

\* argv[1] - IP-адрес сокета

\* argv[2] - номер порта сокета

\* \return 0 - успешное выполнение программы,

\* EXIT\_FAILURE - какая либо ошибка

\*/

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

{

int socketFileDescriptor;

int portNumber = 3000;

struct sockaddr\_in name;

memset((char \*)&name, 0, sizeof(name));

// Назначаем имя сервера сокету

name.sin\_family = AF\_INET;

name.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

if (INADDR\_NONE == name.sin\_addr.s\_addr)

{

perror("inet\_addr"); // !!! заменить на fprintf(stderr, "Address error\n");

exit(1);

}

name.sin\_port = htons((u\_short)portNumber);

// Создание сокета

socketFileDescriptor = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if (0 > socketFileDescriptor)

perror("socket"); // !!! заменить на fprintf(stderr, "Socket error\n");

int i = 1;

setsockopt(socketFileDescriptor, SOL\_SOCKET, SO\_REUSEADDR,

(const char \*)&i, sizeof(i)

);

// Соединение сокета

if (0 > connect(socketFileDescriptor,

(struct sockaddr \*) &name,

(socklen\_t) sizeof(name))

)

{

perror("connect"); // !!! заменить на fprintf(stderr, "Connect error\n");

exit(1);

}

char message[256];

message[0] = '\0';

int flag;

char exit[256] = "exit";

printf("Для выхода наберите \"exit\"\n");

do

{

// Отсылаем 1 сообщение на сервер

do

{

flag = 1;

printf("Введите массив через пробел (по окончании поставьте \".\"): ");

fgets(message, 256, stdin);

message[strlen(message) - 1] = 0;

if (0 == strcmp(message, exit))

{

WriteText(socketFileDescriptor, message);

return 0;

}

// Проверка на упорядоченность введенного массива

char\* buf\_message;

strcpy(buf\_message, message);

char\* buffer = strtok(buf\_message, " ");

int previous;

int next;

while (buffer != NULL)

{

previous = atoi(buffer);

buffer = strtok(NULL, " ");

if (buffer == NULL)

break;

next = atoi(buffer);

if (previous > next)

{

printf("Вы ввели неупорядоченный массив\n");

flag = 0;

}

}

} while (flag == 0);

WriteText(socketFileDescriptor, message);

// Отсылаем 2 сообшение на сервер

printf("Введите ключ (элемент), который необходимо найти в массиве: ");

fgets(message, 256, stdin);

message[strlen(message) - 1] = 0;

if (0 == strcmp(message, exit))

{

WriteText(socketFileDescriptor, message);

return 0;

}

WriteText(socketFileDescriptor, message);

printf("===============================\n");

} while (1);

// Разъединение и закрытие сокета

shutdown(socketFileDescriptor, SHUT\_WR);

close(socketFileDescriptor);

return 0;

}

**3 Примеры работы программ**

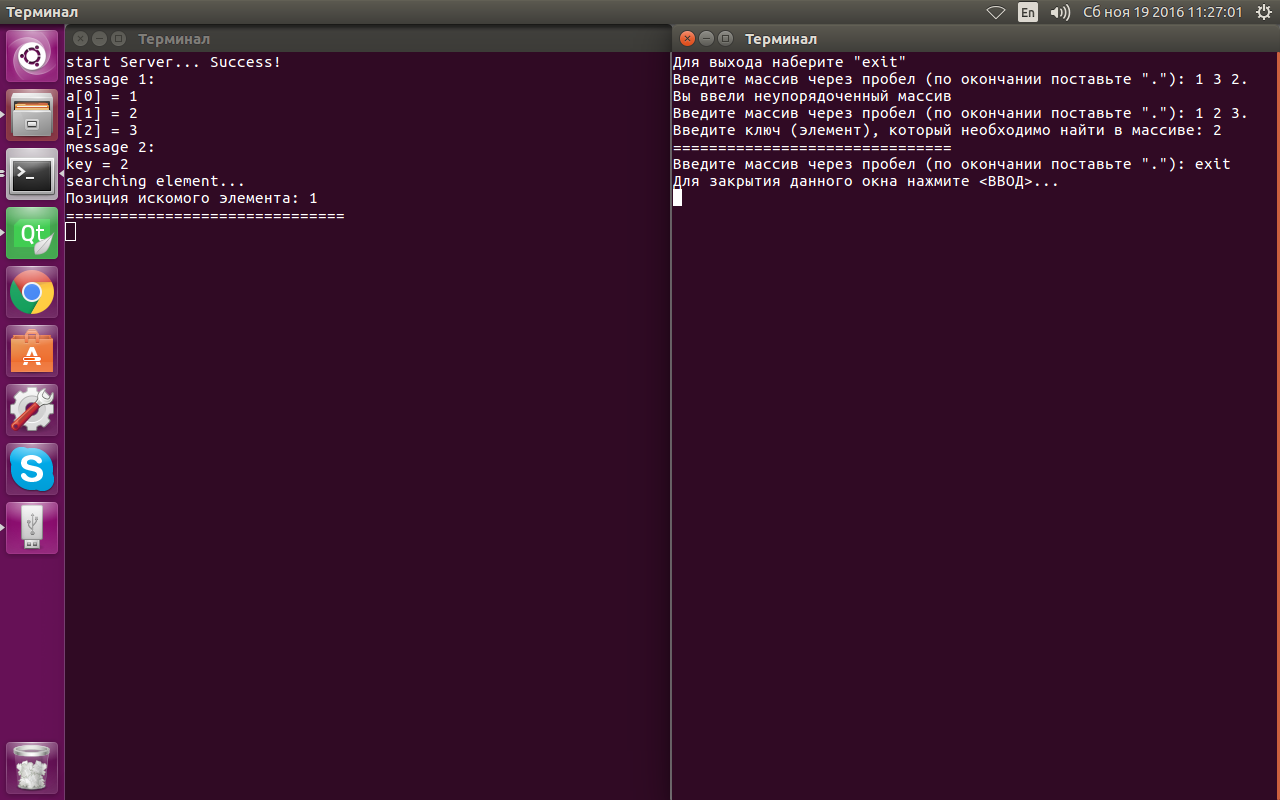
****

Рисунок 1 – Клиент и Сервер