МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Вычислительной Техники

ОТЧЕТ

по лабораторной работе №11
по дисциплине «Организация процессов и программирование в среде Linux»
ТЕМА: «ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ ЧЕРЕЗ СОКЕТЫ»

Студент гр. 8308 _____ Тайсумов И.И.

Преподаватель _____ Разумовский Г.В.

Санкт-Петербург
2021

Цель работы.

Целью лабораторной работы является знакомство с организацией сокетов и системными функциями, обеспечивающими работу с сокетами.

Задание.

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.

Обработка результатов эксперимента.

Программы были разработаны и откомпилированы. После чего программы были запушена через два терминала. Результаты работы программ приведены на рисунках:

```
taisumov@taisumov-TM1703:~/CLionProjects/CPP_lab11_client$ ./main
sent: 10 0 5 5 1 6 3 13 2 9
received: 0 1 2 3 5 5 6 9 10 13
taisumov@taisumov-TM1703:~/CLionProjects/CPP_lab11_client$
taisumov@taisumov-TM1703:~/CLionProjects/CPP_lab11_client$ ./main
sent: 2 1 4 11 11 2 2 2 6 1
received: 1 1 2 2 2 2 4 6 11 11
taisumov@taisumov-TM1703:~/CLionProjects/CPP_lab11_client$ ./main
sent: 4 13 14 7 6 7 3 5 4 11
received: 3 4 4 5 6 7 7 11 13 14
taisumov@taisumov-TM1703:~/CLionProjects/CPP_lab11_client$ ./main
sent: 1 3 1 7 8 0 11 6 13 8
received: 0 1 1 3 6 7 8 8 11 13
taisumov@taisumov-TM1703:~/CLionProjects/CPP_lab11_client$ ./main
sent: 0 13 2 1 4 12 2 6 7 5
received: 0 1 2 2 4 5 6 7 12 13
taisumov@taisumov-TM1703:~/CLionProjects/CPP_lab11_client$
```

Рисунок 1. Работа клиента

```
taisumov@taisumov-TM1703:~/CLionProjects/CPP_lab11_server$ ./main
sorted: 0 1 2 3 5 5 6 9 10 13
sorted: 1 1 2 2 2 2 4 6 11 11
sorted: 3 4 4 5 6 7 7 11 13 14
sorted: 0 1 1 3 6 7 8 8 11 13
sorted: 0 1 2 2 4 5 6 7 12 13
TIME OUT, CLOSING PROGRAM...
taisumov@taisumov-TM1703:~/CLionProjects/CPP_lab11_server$
```

Рисунок 2. Работа сервера

```
taisumov@taisumov-TM1703:~/CLionProjects/CPP_lab11_client$ ./main
CONNECT ERROR!
taisumov@taisumov-TM1703:~/CLionProjects/CPP_lab11_client$
```

Рисунок 3. Работа клиента с незапущенным сервером

```
taisumov@taisumov-TM1703:~/CLionProjects/CPP_lab11_server$ ./main
TIME OUT, CLOSING PROGRAM...
taisumov@taisumov-TM1703:~/CLionProjects/CPP_lab11_server$
```

Рисунок 4. Работа сервера с незапущенным клиентом

Текст программы приведен в приложении.

Вывод.

При выполнении лабораторной работы изучены и использованы сокеты и системные функции, обеспечивающие работу с сокетами. Программа, разработанная в соответствии с заданием, работает корректно.

ПРИЛОЖЕНИЕ

client/main.cpp:

```
#include <iostream>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

using namespace std;

void printSortedArray(int *array) {
   for(int i = 0; i < 10; ++i)
      cout << array[i] << ' ';
   cout << endl;
}

int main() {
   int socket_fd = 1;

   // Открытие сокета
   // Эта операция выполняет построение сокета и возвращает его дескриптор.
   if((socket_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
      cout << "SOCKET CREATION ERROR!" << endl;
      return EXIT_FAILURE;
   }

   //struct sockaddr_in {
   //  short int sin_family;         // Тип домена – значение AF_INET
   //  unsigned short int sin_port;    // Номер порта в сетевом порядке байт,
   //                                  первые 1024 порта зарезервированы;
   //                                  если 0, то система самостоятельно выберет номер порта,
   //  struct in_addr sin_addr;       // IP-адрес в сетевом порядке байт
   //  unsigned char sin_zero[8];     // Дополнение до размера структуры sockaddr
   //};
   sockaddr_in socketAddress;
   socketAddress.sin_family= AF_INET;
   socketAddress.sin_addr.s_addr = INADDR_ANY;
   socketAddress.sin_port = htons(8080);

   time_t  endTimeToConnect,
         startTimeToConnect = time(nullptr);
   int isConnected;
   // за 5 секунд пытаемся подключиться
   do {
      isConnected = connect(socket_fd, (sockaddr*)&socketAddress, sizeof(socketAddress));
      endTimeToConnect = time(nullptr);
      if((endTimeToConnect - startTimeToConnect) > 4)
         break;
   }while(isConnected < 0);
```

```
    // если не получилось подключиться, выходим из программы
    if(isConnected == -1) {
        cout << "CONNECT ERROR!" << endl;
        return EXIT_FAILURE;
    }

    // создаем массив
    int* array = new(int[10]);
    srand(time(0));
    for(int i = 0; i < 10; ++i)
        array[i] = rand() % 15;

    // отправляем массив на сервер
    send(socket_fd, array, 40, 0);
    cout << "sent: ";
    printSortedArray(array);

    fd_set socket_set;

    // обнуляем наборы дескрипторов
    FD_ZERO(&socket_set);
    FD_SET(socket_fd, &socket_set);

    // задаем таймер в 1 секунду
    timeval timeOut = {1};

    // n – кол-во опрашиваемых дескрипторов сокетов(на единицу больше самого
    //большого номера описателей из всех наборов), если все FD_SETSIZE
    //
    // readfds, writefds, exceptfds наборы дескрипторов, которые следует проверять,
    //соответственно, на готовность к чтению, записи и на наличие
    //исключительных ситуаций.
    //
    // timeout- время ожидания (верхняя граница времени, которое пройдет перед
    //возвратом из select), если 0, то процесс будет приостановлен до тех пор, пока
    //один из сокетов не изменит свое состояние
    if(select(socket_fd+1, &socket_set, nullptr, nullptr, &timeOut) > 0) {

        recv(socket_fd, array, 40, 0);
        cout << "received: ";
        printSortedArray(array);

    }
    else {
        cout << "TIME OUT, CLOSING PROGRAM..." << endl;
    }

    close(socket_fd);
    return 0;
}
```

server/main.cpp:

```
#include <iostream>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <algorithm>

using namespace std;

void printSortedArray(int *array) {
   for(int i = 0; i < 10; ++i)
      cout << array[i] << ' ';
   cout << endl;
}

void processSorter(int connect_fd) {
   if(fork() == 0){
      int array[10];
      //sleep(5);
      recv(connect_fd, array, sizeof(array), 0);
      sort(array, array + 10);
      cout << "sorted: ";
      printSortedArray(array);

      send(connect_fd, array, sizeof(array) * 10, 0);

      close(connect_fd);
      exit(EXIT_SUCCESS);
   }
}

int main() {
   int socket_fd;
   // Открытие сокета
   // Эта операция выполняет построение сокета и возвращает его дескриптор.
   if((socket_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
      cout << "SOCKET CREATION ERROR!" << endl;
      return EXIT_FAILURE;
   }

   //struct sockaddr_in {
   //  short int sin_family;         // Тип домена – значение AF_INET
   //  unsigned short int sin_port;    // Номер порта в сетевом порядке байт,
   //                                  первые 1024 порта зарезервированы;
   //                                  если 0, то система самостоятельно выберет номер порта,
   //  struct in_addr sin_addr;       // IP-адрес в сетевом порядке байт
```

```
//  unsigned char sin_zero[8];     // Дополнение до размера структуры sockaddr
//};
sockaddr_in socketAddress;
socketAddress.sin_family= AF_INET;
socketAddress.sin_addr.s_addr = INADDR_ANY;
socketAddress.sin_port = htons(8080);

//int bind(int sockfd, struct sockaddr *addr, int addrlen);
//sockfd – дескриптор сокета
//addr – указатель на структуру с адресом
//addrlen = sizeof(addr) – длина структуры
if(bind(socket_fd, (sockaddr*)&socketAddress, sizeof(sockaddr))) {
   return EXIT_FAILURE;
}

//перевод сокета в пассивное (слушающее) состояние и создание очередей для
//порождаемых при установлении соединения присоединенных сокетов, находящихся в
//состоянии не полностью установленного соединения и полностью установленного
//соединения.
if(listen(socket_fd, 1) < 0) {
   return EXIT_FAILURE;
}

// Задаём время ожидания
timeval timeOut = {5};

for(;;) {
   fd_set socket_set;

   //обнуляем наборы дескрипторов
   FD_ZERO(&socket_set);
   FD_SET(socket_fd, &socket_set);

   // n – кол-во опрашиваемых дескрипторов сокетов(на единицу больше самого
   //большого номера описателей из всех наборов), если все FD_SETSIZE
   //
   // readfds, writefds, exceptfds наборы дескрипторов, которые следует проверять,
   //соответственно, на готовность к чтению, записи и на наличие
   //исключительных ситуаций.
   //
   // timeout- время ожидания (верхняя граница времени, которое пройдет перед
   //возвратом из select), если 0, то процесс будет приостановлен до тех пор, пока
   //один из сокетов не изменит свое состояние
   if(select(socket_fd+1, &socket_set, nullptr, nullptr, &timeOut) > 0) {
      int connect_fd,
         sockSize = sizeof(socketAddress);

      //accept используется сервером, ориентированным на
      //установление связи путем виртуального соединения, для приема полностью
      //установленного соединения .
      if((connect_fd = accept(socket_fd, (sockaddr*)&socketAddress, (socklen_t *)&sockSize)) < 0) {
```

```
            return EXIT_FAILURE;
        }
        processSorter(connect_fd);
    }
    else {
        cout << "TIME OUT, CLOSING PROGRAM..." << endl;
        break;
    }
    timeOut.tv_sec = 10;

  }
  close(socket_fd);
  return 0;
}
```