Министерство науки и образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
"Московский институт электронной техники"
(МИЭТ)

Отчет по лабораторной работе № 7

Операционные системы

Выполнил: студент ПМ - 31

Мартынова Мария Олеговна

2023 г.

Задание 1

В этой лабораторной работе вам предстоит потрогать два клиент-серверных приложения. Первое использует проткол TCP, второй UDP. Вам необходимо:

Скомпилировать все четыре программы.

Вынести все константы (объявленные через #define) в аргументы командной строки.

Скомпилировать оба приложения через makefile.

lab7/src/tcpclient.c

```c
#define SADDR struct sockaddr
#define SIZE sizeof(struct sockaddr_in)

int main(int argc, char *argv[]) {
  //argv[1] - IP-адрес сервера, к которому будем подключаться
  //argv[2] - порт сервера, к которому будем подключаться
  //argv[3] - размер буфера, который будем посылать серверу
  int fd;
  int nread;
  struct sockaddr_in servaddr;
  if (argc < 4) { // проверка на правильное введение параметров
    printf("Too few arguments \n");
    exit(1);
  }

  int size = atoi(argv[3]); //перевод строки в целое число
  char* buf = malloc(sizeof(char) * size); //выделение динамической памяти буферу

  if ((fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) { //создаем сокет с потоковым взаимодействием
    perror("socket creating");
    exit(1);
  }

  memset(&servaddr, 0, SIZE);//заполнение объекта нулями
  servaddr.sin_family = AF_INET;//указание способа взаимодействия сервера (сокет, как объект ядра)

  if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0) { // записываем в объект IP-адрес сервера
    perror("bad address");
    exit(1);
  }

  servaddr.sin_port = htons(atoi(argv[2]));// записываем в объект порт сервера

  if (connect(fd, (SADDR *)&servaddr, SIZE) < 0) {//устанавливаем соединение с сервером
    perror("connect");
    exit(1);
  }

  write(1, "Input message to send\n", 22);//вывод в консоль сообщения
  while ((nread = read(0, buf, size)) > 0) { //считывание сообщения в буфер
    if (write(fd, buf, nread) < 0) {//отправка сообщения серверу
      perror("write");
      exit(1);
    }
  }

  close(fd);
  exit(0);
}
```

lab7/src/tcpserver.c

```c
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define SADDR struct sockaddr

int main(int argc, char *argv[]) {
  //argv[1] - порт сервера
  //argv[2] - размер принимаемого от клиента сообщения
  const size_t kSize = sizeof(struct sockaddr_in);

  if (argc < 3) {
    printf("Too few arguments \n");
    exit(1);
  }

  int size = atoi(argv[2]);

  int lfd, cfd;
  int nread;
  char* buf = malloc(sizeof(char) * size);
  struct sockaddr_in servaddr;
  struct sockaddr_in cliaddr;

  if ((lfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {//создание сокета сервера
    perror("socket");
    exit(1);
  }

  memset(&servaddr, 0, kSize);
  servaddr.sin_family = AF_INET;//вид взаимодействия (через сокет, как объект ядра)
  servaddr.sin_addr.s_addr = htonl(INADDR_ANY);//может работать с любым IP-адресом
  servaddr.sin_port = htons(atoi(argv[1]));//запись порта сервера

  if (bind(lfd, (SADDR *)&servaddr, kSize) < 0) {//ставим сокету свойства
    perror("bind");
    exit(1);
  }

  if (listen(lfd, 5) < 0) {//слушающий режим, ждем соединения
    perror("listen");
    exit(1);
  }

  while (1) {
    unsigned int clilen = kSize;

    if ((cfd = accept(lfd, (SADDR *)&cliaddr, &clilen)) < 0) {//принятие соединения сервера с клиентом
      perror("accept");
      exit(1);
    }
    printf("connection established\n");

    while ((nread = read(cfd, buf, size)) > 0) {//считывание в буфер сообщения от клиента
      write(1, &buf, nread);//вывод считанного сообщения в консоль
    }

    if (nread == -1) {
      perror("read");
      exit(1);
    }
    close(cfd);
  }
}
```

lab7/src/udpclient.c

```c
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>

#include <arpa/inet.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

#define SADDR struct sockaddr
#define SLEN sizeof(struct sockaddr_in)

int main(int argc, char **argv) {
  //argv[1] - IP-адрес сервера, к которому будем подключаться
  //argv[2] - порт сервера, к которому будем подключаться
  //argv[3] - размер буфера, который будем посылать серверу
  int sockfd, n;
  struct sockaddr_in servaddr;
  struct sockaddr_in cliaddr;

  if (argc != 4) {
    printf("usage: client <IPaddress of server>, <SERV_PORT>, <BUFSIZE>\n");
    exit(1);
  }

  int size = atoi(argv[3]);
  char* sendline = malloc(size);//массив, которым будем отправлять серверу
  char* recvline = malloc(size + 1);//массив, в который будем записывать ответ от сервера

  memset(&servaddr, 0, sizeof(servaddr));//заполнение объекта нулями
  servaddr.sin_family = AF_INET;//указание способа взаимодействия сервера (сокет, как объект ядра)
  servaddr.sin_port = htons(atoi(argv[2]));// записываем в объект порт сервера

  if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) < 0) {// записываем в объект IP-адрес сервера
    perror("inet_pton problem");
    exit(1);
  }
  if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) { //создание сокета клиента
    perror("socket problem");
    exit(1);
  }

  write(1, "Enter string\n", 13);//вывод сообщения в консоль

  while ((n = read(0, sendline, size)) > 0) {//считывание в sendline введеного сообщения
    if (sendto(sockfd, sendline, n, 0, (SADDR *)&servaddr, SLEN) == -1) {//отправка сообщения серверу с указанными свойствами
      perror("sendto problem");
      exit(1);
    }

    if (recvfrom(sockfd, recvline, size, 0, NULL, NULL) == -1) {//получение ответа от сервера и записывание в recvline
      perror("recvfrom problem");
      exit(1);
    }

    printf("REPLY FROM SERVER= %s\n", recvline);//вывод в консоль ответа от сервера
  }
  close(sockfd);
}
```

lab7/src/udpserver.c

```c
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

#define SADDR struct sockaddr
#define SLEN sizeof(struct sockaddr_in)

int main(int argc, char **argv) {
  //argv[1] - порт сервера
  //argv[2] - размер принимаемого от клиента сообщения
  int sockfd, n;
  struct sockaddr_in servaddr;
  struct sockaddr_in cliaddr;

  if (argc != 3) {
    printf("usage: client <SERV_PORT>, <BUFSIZE>\n");
    exit(1);
  }

  int size = atoi(argv[2]);
  char* mesg = malloc(size);//выделение памяти для записывания сообщения от клиента
  char ipadr[16];

  if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {//создание сокета сервера
    perror("socket problem");
    exit(1);
  }

  memset(&servaddr, 0, SLEN);
  servaddr.sin_family = AF_INET;//вид взаимодействия (через сокет, как объект ядра)
  servaddr.sin_addr.s_addr = htonl(INADDR_ANY);//может работать с любым IP-адресом
  servaddr.sin_port = htons(atoi(argv[1]));//запись порта сервера

  if (bind(sockfd, (SADDR *)&servaddr, SLEN) < 0) {//ставим сокету свойства
    perror("bind problem");
    exit(1);
  }
  printf("SERVER starts...\n");//пишется в консоль, что сервер запущен

  while (1) {
    unsigned int len = SLEN;

    if ((n = recvfrom(sockfd, mesg, size, 0, (SADDR *)&cliaddr, &len)) < 0) {//считывание пришедшего сообщения в mesg, записывание в cliaddr данные о клиенте
      perror("recvfrom");
      exit(1);
    }
    mesg[n] = 0;//конец сообщения

    printf("REQUEST %s      FROM %s : %d\n", mesg,
           inet_ntop(AF_INET, (void *)&cliaddr.sin_addr.s_addr, ipadr, 16),//IP-адрес клиента
           ntohs(cliaddr.sin_port));//порт клиента

    if (sendto(sockfd, mesg, n, 0, (SADDR *)&cliaddr, len) < 0) {//отправка сообщения обратно клиенту
      perror("sendto");
      exit(1);
    }
  }
}
```

lab7/src/makefile

```
CC=gcc
CFLAGS=-I.

tcp :
  $(CC) -o tcpclient tcpclient.c $(CFLAGS)
  $(CC) -o tcpserver tcpserver.c $(CFLAGS)

udp :
  $(CC) -o udpclient udpclient.c $(CFLAGS)
  $(CC) -o udpserver udpserver.c $(CFLAGS)
```

```
~/oslab2019$ cd lab7/src
~/.../lab7/src$ make tcp
gcc -o tcpclient tcpclient.c -I.
gcc -o tcpserver tcpserver.c -I.
~/.../lab7/src$ make udp
gcc -o udpclient udpclient.c -I.
gcc -o udpserver udpserver.c -I.
~/.../lab7/src$ ./udpserver 20001 100 &
[1] 58
~/.../lab7/src$ SERVER starts...
./udpclient 127.0.0.1 20001 100
Enter string
International Space Day - 21 May
REQUEST International Space Day - 21 May
      FROM 127.0.0.1 : 33829
REPLY FROM SERVER= International Space Day - 21 May
```

Задание 2

Ответить на следующие вопросы:

1. Что делают оба приложения?
2. Что произойдет, если tcpclient отправит сообщение незапущенному серверу?
3. Что произойдет, если udpclient отправит сообщение незапущенному серверу?
4. Что произойдет, если tcpclient отвалится во время работы с сервером?
5. Что произойдет, если udpclient отвалится во время работы с сервером?
6. Что произойдет, если udpclient отправит сообщение на несуществующий / выключенный сервер?
7. Что произойдет, если tcpclient отправит сообщение на несуществующий / выключенный сервер?
8. В чем отличия UDP и TCP протколов?

1. Обмен сообщениями, введенными пользователем

2. Системный вызов connect вернет ошибку о невозможности соединения

3. Клиент зависнет, потому что будет посылать данные, но не получив ответа, продолжит посылать данные

4. Сервер получит ошибку и прекратит взаимодействие

5. Сервер зависнет, потому что будет посылать/принимать данные, но не получив ответа, продолжит посылать данные

6. См. п.3

7. См. п.2

8. Рассмотрим транспортный слой и поймем, как именно доставляют информацию по компьютерной сети. Существует два основных варианта взаимодействия: **дейтаграммное и потоковое**.

Общее

Участники сетевого взаимодействия, работающие на одном компьютере, будут для идентификации использовать двухбайтные беззнаковые целые числа, называемые **номерами портов** или просто **портами**.

Про дейтаграмнное, UDP (user datagram protocol)

Данные, которые нужно отправить по сети, делятся на части определенного объема. Такие части будем называть **пакетами**. Содержимое такого пакета обычно называют **дейтаграммой**. Дейтаграммное взаимодействие выглядит так: берем некую порцию данных, надписываем адрес получателя и просим операционную систему позаботиться о пересылке.

Важно: отправленный **пакет может потеряться** или, наоборот, прийти в двух экземплярах, а пакеты, отправленные раньше других, могут прийти к получателю позже. Поэтому адресат в ответ на полученный пакет обязательно должен отправить подтверждение; в свою очередь, отправитель должен некоторое время ожидать получения подтверждения, а если такового не поступит, снова отправить ту же самую дейтаграмму.

Решение всех этих проблем возлагается на программиста.

Про потоковое, TCP (transmission control protocol)

С ним дела обстоят прямо противоположным образом. Сетевое соединение выглядит как обычный поток (иллюзия непрерывности данных, поток байтов, как в канале: с одного конца записывают, в другого конца после доставки считывают, и наоборот, т.е. двухсторонний поток). **Операционная система гарантирует**, что все байты, записанные в поток, будут затем доступны для чтения на другом конце потока, причем их порядок будет сохранен. При невозможности соблюдения этой гарантии соединение окажется разорвано, **о чем узнают оба партнера**.

Важно: в отличие от работы с дейтограммами, при потоковой работе необходимо сначала **установить соединение**, для чего один из будущих партнеров должен находиться в состоянии ожидания запроса на соединение, а второй должен этот запрос послать.