Лабораторная работа 7

Лабораторная работа №7

Задание 1

Необходимые знания

1. Компиляция программ на языке C.
2. Аргументы командной строки.
3. Протокол TCP.
4. Протокол UDP.

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.

tcpclient.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/types.h>
#include <unistd.h>

#define BUFSIZE 100
#define SADDR struct sockaddr
#define SIZE sizeof(struct sockaddr_in)

int main(int argc, char *argv[]) {
  int fd;
  int nread;
  //char buf[BUFSIZE];
  struct sockaddr_in servaddr;
```

```
  if (argc < 4) {
     printf("Usage: %s <IP address> <port> <buffer_size>\n", argv[0]);
    exit(1);
  }

  int buffer_size = atoi(argv[3]);
  char buf[buffer_size];

  if ((fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("socket creating");
    exit(1);
  }

  memset(&servaddr, 0, SIZE);
  servaddr.sin_family = AF_INET;

  if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0) {
    perror("bad address");
    exit(1);
  }

  servaddr.sin_port = htons(atoi(argv[2]));

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

  write(1, "Input message to send\n", 22);
  while ((nread = read(0, buf, buffer_size)) > 0) {
    if (write(fd, buf, nread) < 0) {
      perror("write");
      exit(1);
    }
  }

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

tcpserver.c:

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

#include <sys/socket.h>
```

```c
#include <sys/types.h>
#include <unistd.h>

#define SERV_PORT 10050
#define BUFSIZE 100
#define SADDR struct sockaddr

int main(int argc, char *argv[]) {
  if (argc < 3) {
    printf("Usage: %s <port> <buffer_size>\n", argv[0]);
    exit(1);
  }

  const size_t kSize = sizeof(struct sockaddr_in);

  int lfd, cfd;
  int nread;
  //char buf[BUFSIZE];
  int buffer_size = atoi(argv[2]);
  char buf[buffer_size];
  struct sockaddr_in servaddr;
  struct sockaddr_in cliaddr;

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

  memset(&servaddr, 0, kSize);
  servaddr.sin_family = AF_INET;
  servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
  servaddr.sin_port = htons(atoi(argv[1]));

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

  if (listen(lfd, 5) < 0) {
    perror("listen");
    exit(1);
  }

  while (1) {
    unsigned int clilen = kSize;

    if ((cfd = accept(lfd, (SADDR *)&cliaddr, &clilen)) < 0) {
      perror("accept");
      exit(1);
    }
    printf("connection established\n");
```

```
    while ((nread = read(cfd, buf, buffer_size)) > 0) {
      write(1, &buf, nread);
    }

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

udpclient.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 SERV_PORT 20001
#define BUFSIZE 1024
#define SADDR struct sockaddr
#define SLEN sizeof(struct sockaddr_in)

int main(int argc, char *argv[]) {
  int sockfd, n;
  //char sendline[BUFSIZE], recvline[BUFSIZE + 1];
  struct sockaddr_in servaddr;
  //struct sockaddr_in cliaddr;

  if (argc != 4) {
    printf("usage: %s <IPaddress of server> <port> <buffer_size>\n", argv[0]);
    exit(1);
  }

  int buffer_size = atoi(argv[3]);
  char sendline[buffer_size], recvline[buffer_size + 1];

  memset(&servaddr, 0, sizeof(servaddr));
  servaddr.sin_family = AF_INET;
  servaddr.sin_port = htons(atoi(argv[2]));
```

```
  if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) < 0) {
    perror("inet_pton problem");
    exit(1);
  }
  if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
    perror("socket problem");
    exit(1);
  }

  write(1, "Enter string\n", 13);

  while ((n = read(0, sendline, buffer_size)) > 0) {
    if (sendto(sockfd, sendline, n, 0, (SADDR *)&servaddr, SLEN) == -1) {
      perror("sendto problem");
      exit(1);
    }

    if (recvfrom(sockfd, recvline, buffer_size, 0, NULL, NULL) == -1) {
      perror("recvfrom problem");
      exit(1);
    }

    printf("REPLY FROM SERVER= %s\n", recvline);
  }
  close(sockfd);
}
```

udpserver.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 SERV_PORT 20001
#define BUFSIZE 1024
#define SADDR struct sockaddr
#define SLEN sizeof(struct sockaddr_in)

int main(int argc, char *argv[]) {
```

```c
  if (argc < 3) {
    printf("Usage: %s <port> <buffer_size>\n", argv[0]);
    exit(1);
  }

  int sockfd, n;
  //char mesg[BUFSIZE], ipadr[16];
  int buffer_size = atoi(argv[2]);
  char mesg[buffer_size], ipadr[16];
  struct sockaddr_in servaddr;
  struct sockaddr_in cliaddr;

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

  memset(&servaddr, 0, SLEN);
  servaddr.sin_family = AF_INET;
  servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
  servaddr.sin_port = htons(atoi(argv[1]));

  if (bind(sockfd, (SADDR *)&servaddr, SLEN) < 0) {
    perror("bind problem");
    exit(1);
  }
  printf("SERVER starts...\n");

  while (1) {
    unsigned int len = SLEN;

    if ((n = recvfrom(sockfd, mesg, buffer_size, 0, (SADDR *)&cliaddr, &len)) < 0) {
      perror("recvfrom");
      exit(1);
    }
    mesg[n] = 0;

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

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

makefile:

```
CC = gcc
CFLAGS = -Wall -g

all: tcpclient tcpserver udpclient udpserver

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

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

udpclient: udpclient.c
    $(CC) $(CFLAGS) -o udpclient udpclient.c

udpserver: udpserver.c
    $(CC) $(CFLAGS) -o udpserver udpserver.c

clean:
    rm -f tcpclient tcpserver udpclient udpserver
```

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?

```
⊗ @markast555 →/workspaces/os_lab_2019/lab7/src (master) $ ./tcpclient 127.0.0.1 10050 100
  connect: Connection refused
⊗ @markast555 →/workspaces/os_lab_2019/lab7/src (master) $ ./tcpclient 127.0.0.1 10050 100
  Input message to send
  Hello server
  Goodbye
  Hey?
  Sad
```

```
@markast555 ➜/workspaces/os_lab_2019/lab7/src (master) $ ./tcpserver 10050 100
connection established
Hello server
Goodbye
^C
```

```
@markast555 ➜/workspaces/os_lab_2019/lab7/src (master) $ ./tcpclient 127.0.0.1 10050 100
Input message to send
qwerty
^C
@markast555 ➜/workspaces/os_lab_2019/lab7/src (master) $ ./tcpserver 10050 100
connection established
qwerty
```

```
@markast555 ➜/workspaces/os_lab_2019/lab7/src (master) $ ./udpclient 127.0.0.1 20001 1024
Enter string
Hey?
Sad
:(
@markast555 ➜/workspaces/os_lab_2019/lab7/src (master) $ ./tcpserver 10050 100
connection established
qwerty
^C
@markast555 ➜/workspaces/os_lab_2019/lab7/src (master) $
```

```
@markast555 ➜/workspaces/os_lab_2019/lab7/src (master) $ ./udpclient 127.0.0.1 20001 1024
Enter string
Hello
REPLY FROM SERVER= Hello

Hey?
Sad
:(
@markast555 ➜/workspaces/os_lab_2019/lab7/src (master) $ ./udpserver 20001 1024
SERVER starts...
REQUEST Hello
      FROM 127.0.0.1 : 49543
^C
```

```
@markast555 ➜/workspaces/os_lab_2019/lab7/src (master) $ ./udpclient 127.0.0.1 20001 1024
Enter string
Boo
REPLY FROM SERVER= Boo
^C
@markast555 ➜/workspaces/os_lab_2019/lab7/src (master) $ ./udpserver 20001 1024
SERVER starts...
REQUEST Boo
      FROM 127.0.0.1 : 49811
```

1. Что делают оба приложения?

Эти приложения реализуют клиент-серверную архитектуру с использованием протоколов TCP и UDP.

2. Что произойдет, если tcpclient отправит сообщение незапущенному серверу?

Соединение с сервером не установится.

3. Что произойдет, если udpclient отправит сообщение незапущенному серверу?

Сообщение не будет доставлено.

4. Что произойдет, если tcpclient отвалится во время работы с сервером?

Ничего не произойдёт.

5. Что произойдет, если udpclient отвалится во время работы с сервером?

Ничего не произойдёт.

6. Что произойдет, если udpclient отправит сообщение на несуществующий / выключенный сервер?

Сообщение не будет доставлено.

7. Что произойдет, если tcpclient отправит сообщение на несуществующий / выключенный сервер?

Соединение с сервером не установится.

8. В чем отличия UDP и TCP протколов?

Соединение:

TCP: Ориентированный на соединение. Устанавливается надежное соединение перед передачей данных.

UDP: Без соединения. Данные отправляются без предварительного установления соединения.

Надежность:

- **TCP**: Гарантирует доставку данных, проверяет целостность и порядок. Если данные потеряны, они будут повторно отправлены.
- **UDP**: Не гарантирует доставку, порядок или целостность данных. Нет механизма повторной отправки.

Скорость:

- **TCP**: Более медленный из-за дополнительных проверок и установления соединения.
- **UDP**: Более быстрый, так как не требует установления соединения и дополнительных проверок.

Использование:

- **TCP**: Используется для приложений, требующих надежности, таких как веб-браузеры и электронная почта.
- **UDP**: Используется для приложений, где скорость важнее надежности, таких как потоковое видео и онлайн-игры.