COE 768
UDP Server Implementation

In this lab, we will study the characteristics of UDP and the implementation of UDP server.

I. UDP System calls

UDP is another transport protocol commonly used by network applications. Unlike TCP, applications based on UDP can send data without the establishment of transport-layer connection.

System calls used by the UDP server

A UDP server only needs three system calls to prepare the UDP socket for communication.

```
s = socket(AF_INET, SOCK_DGRAM, 0);
bind(s, (struct sockaddr *)&sin, sizeof(sin));
listen(s,5);
```

The socket system call indicates that the transport service is UDP (SOCK_DGRAM). It does not need to call *accept* for it does not need to deal with connection request. UDP server calls *recvfrom* to wait for data from the client.

```
recvfrom(s, buf, sizeof(buf), 0, (struct sockaddr *)&fsin, &alen);
```

recvfrom is same as *read* with the exception that *recvfrom* also returns the address of the sender (the client) stored in the argument fsin. When the server needs to send data back to the client, it will call *sendto*

```
sendto(s, buf, msglen, 0, (struct sockaddr *)&fsin, sizeof(fsin));
```

The argument fsin contains the destination (client) address. This address, of course, came from *recvfrom*.

System calls used by the UDP client

The UDP client makes essentially the same system calls as TCP client to prepare the socket.

```
s = socket(AF_INET, SOCK_DGRAM, 0);
connect(s, (struct sockaddr *)&sin, sizeof(sin));
```

However, in this case, *connect* does not trigger a TCP connection. Instead, it associates the socket to the destination address (stored in the argument sin). Because of this association, the client does not need to use *sendto* when it sends data to the server; instead it can use

write. Similarly, when the client wants to wait for data from the server, it can call *read* instead of *recvfrom* for the client already knew the address of the server. Figure 1 illustrates the system calls and the packet exchanges made by the server and the client.

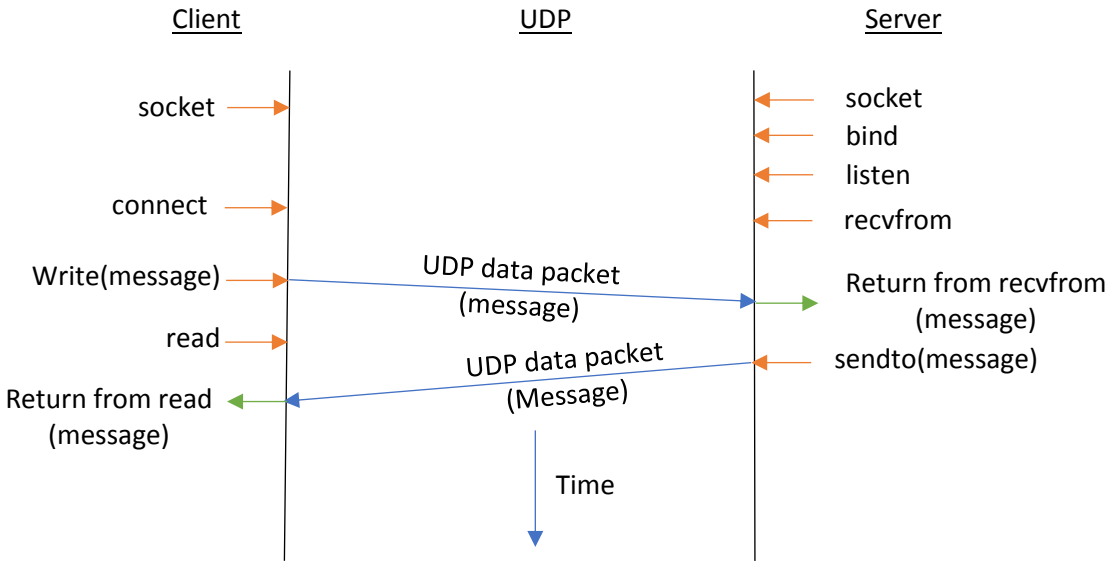


Figure 1

II. Time Service

In this part, we will study a simple UDP service called Time Service.

1. Download the time_server.c and time_client.c from the course D2L site (Content-Lab-Assignments-Socket Programs). Compile time_server.c on VM1 and time_client.c on VM2:

   $./gcc -o time_server time_server.c -lnsl
   $./gcc -o time_client time_client.c -lnsl

2. Setup the server on VM1:

   $./time_server [port_number]

3. Start the Wireshark and enable capture.

4. Start a client on VM2:

   time_client [IP_address_VM1] [port_number]}

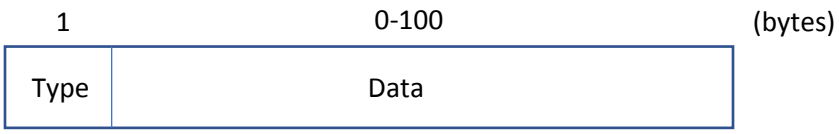
5. You will notice that datetime information is displayed in the client terminal. The service cycle ends with the exit of the client.

6. Stop the Wireshark capture. The captured data should show that only 2 UDP packets were exchanged. There are no connection establishment nor termination packets.

7. By examining the captured data, you probably find that the first message was sent by the client. This message contained some random message that was ignored by the server. What is the function of this message?

III. Program Assignment: File Download Service based on UDP

You are required to write an UDP file download application. In this implementation, a simple data structure is imposed in the protocol data unit (PDU) exchanged between the client and server. The PDU has the following structure:



The data field contains data. The maximum size is 100 bytes. The "type" field is 1 byte. It specifies the PDU type. There are 4 PDU types:

- Type = D -- DATA PDU;
- Type = F -- FINAL PDU;
- Type = E -- ERROR PDU;
- Type = C -- FILENAME PDU.

The DATA PDU is used by the server to send file data to the client; the data field contains file data. The FINAL PDU carries the last batch of file data of a file and is also used to signal the end of a file download. The ERROR PDU is used to report an error. The FILENAME PDU is sent by the client to specify a file that the client wants to download from the server. Figure 2 illustrates packet exchanges of a successful file download.

Similar to the File Download application of Lab 4, the client saves the downloaded data in the local file. On the other hand, error messages are displayed in the Terminal.

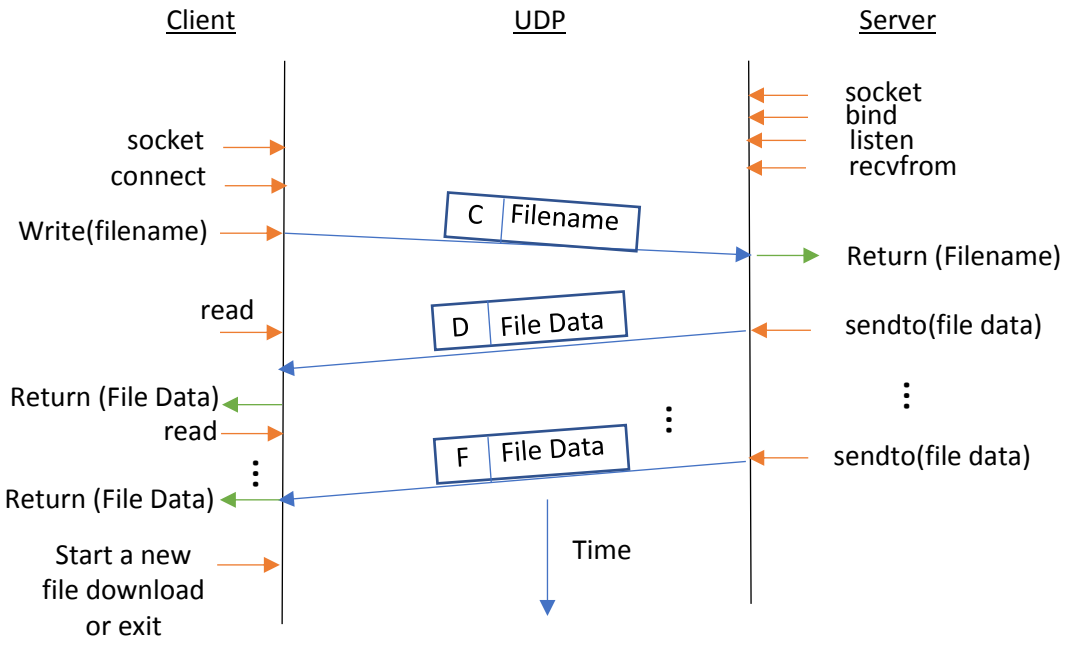


Figure 2

Additional instructions and requirements of the programs

1. The client program should allow the user download multiple files. Consequently, the client program should provide a user interface such that the user can select to download file or quit the program.

2. The PDU should be defined as a pdu structure:

```
struct pdu {
        char type;
        char data[100];
};
```

Imposing such a structure allows the program to readily access a specific field. For example, when the client prepares and sends a FILENAME PDU, it can use the following code:

```
struct pdu spud;
spdu.type = 'C';                    //Set the type to FILENAME PDU
n=read(0, spdu.data, 100);          //Read the filename entered by the user
spdu.data[n-1] = '\0';              //Make the filename a character string
write(sd, &spdu, n+1);                      //Send the PDU to the server
```

You can also use *scanf* to read data from the terminal. In this case, the last three lines can be replaced by

```
scanf("%s", spdu.data);
write(sd, &spud, strlen(spdu.data)+1);
```

3. In this implementation, you may need to predetermine the file size. Knowing the file size let the server know if the batch of data just read from the file is the last batch of data. If it is, the data should be sent in the Final PDU. To predetermine the file size, you can use *lstate* system call.

What you needed to show to your TA

1. Show the completion of steps 1 to 6.

2. Answer the question in step 7.

3. The Time service can run on both UDP and TCP transport protocols. Discuss which transport protocol is more appropriate.

4. Discuss if the Time server should be designed as a concurrent server or non-concurrent (iterative) server

5. Demonstrate your programs in part III. The file size of the file to be downloaded must be great than 100 bytes. The client program should also be able to handle multiple downloads.

6. Go over your programs in part III, locate and explain the parts of the code that deal with the following implementations:

    - How the server transfers the complete file to the client.
    - How the client receives the complete file.
    - How the client handles the error message.