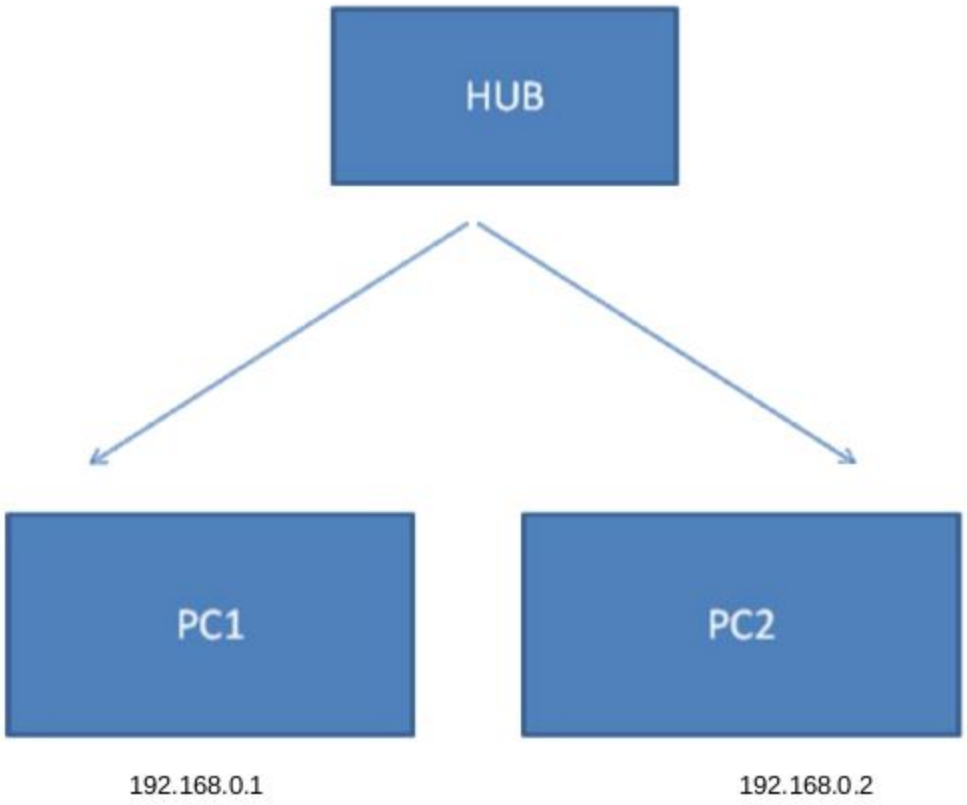
Computer Networks, IT-304
Lab 3-UDP

Topology:

Resources:

1. PC	2
2. NICs	2
3. Hub	1
4. Data cables	2



The User Datagram Protocol:

The TCP/IP protocol suite provides two transport protocols, the User Datagram Protocol (UDP) and the Transmission Control Protocol(TCP).There are some fundamental differences between applications written using TCP versus those that use UDP.

UDP is a connectionless, unreliable, datagram protocol, quite unlike the connection oriented, reliable byte stream provided by TCP. UDP is less complex and easier to understand.

The characteristics of UDP are given below:

End to end: UDP can identify a specific process running on a computer.
Connectionless: UDP follows the connectionless paradigm (see below).
Message oriented : Processes using UDP send and receive individual messages called segments.
Best effort: UDP offers the same best effort delivery as IP.

The Connectionless Paradigm:

UDP uses a connectionless communication setup. A process using UDP does not need to establish a connection before sending data and when two processes stop communicating there are no additional, control messages. Communication consists only of the data segments themselves.

Message Oriented Interface:

UDP provides a message oriented interface. Each message is sent as a single UDP segment.A computer may send UDP packets without first establishing a connection to the recipient. A UDP datagram is carried in a single IP packet and is hence limited to a maximum payload of 65,507 bytes for IPv4 and 65,527 bytes for IPv6. The transmission of large IP packets usually requires IP fragmentation. Fragmentation decreases communication reliability and efficiency and should therefore be avoided.

Application designers are generally aware that UDP does not provide any reliability, e.g., it does not retransmit any lost packets. Often, this is a main reason to consider UDP as a transport. Applications that do require reliable message delivery therefore need to implement appropriate protocol mechanisms in their applications.

UDP's best effort service does not protect against datagram duplication, i.e., an application may receive multiple copies of the same UDP datagram. Application designers therefore need to verify that their application gracefully handles datagram duplication and may need to implement mechanisms to detect duplicates.

The Internet may also significantly delay some packets with respect to others, e.g., due to routing transients, intermittent connectivity, or mobility. This can cause reordering, where UDP datagrams arrive at the receiver in an order different from the transmission order. Applications that require ordered delivery must restore datagram ordering themselves.

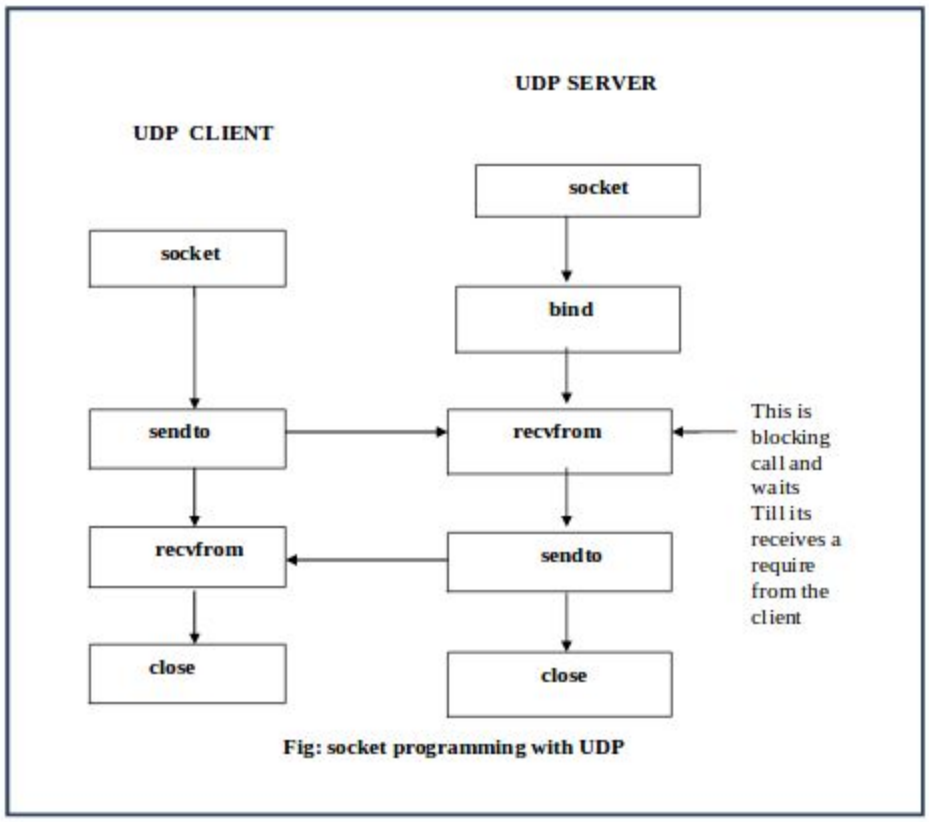


Fig: socket programming with UDP

There are a few steps involved in using sockets:

1. Create the socket
2. Identify the socket (name it)
3. On the server, wait for a message
4. On the client, send a message
5. Send a response back to the client (optional)
6. Close the socket

Important points:

-We can create UDP socket by specifying the second argument to socket function as SOCK_DGRAM.
-The client does not establish a connection with the server.
-The client just sends a datagram to the server using the sendto function , which requires the address of the destination as a parameter.
-Similarly, the server does not accept a connection from a client. Instead, the server just calls the recvfrom function, which waits until data arrives from some client.

Since we do not have a connection, the messages have to be addressed to their destination. Instead of using a *write* system call, we will use ***sendto***, which allows us to specify the destination. The address is identified through the sockaddr structure, the same way as it is in *bind* and as we did when using *connect* for TCP sockets.

The sendto call has the following syntax:

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

int sendto(int socket, const void *buffer, size_t length, int flags, const struct sockaddr
*dest_addr, socklen_t dest_len)
```

With TCP sockets, a server would set up a socket for listening via a *listen* system call and then call *accept* to wait for a connection. UDP is connectionless. A server can immediately listen for messages once it has a socket. We use the ***recvfrom*** system call to wait for an incoming datagram on a specific transport address (IP address and port number).

The *recvfrom* call has the following syntax:

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

int recvfrom(int socket, void *restrict buffer, size_t length, int flags, struct sockaddr *restrict
src_addr, socklen_t *restrict *src_len)
```

Read : https://www.cs.rutgers.edu/~pxk/417/notes/sockets/udp.html

Checksum:

A checksum or hash sum is a fixed-size computed from an arbitrary block of digital data for the purpose of detecting accidental errors that may have been introduced during its transmission or storage. The integrity of the data can be checked at any later time by re-computing the checksum and comparing it with the stored one. If the checksums match, the data were almost certainly not altered (either intentionally or unintentionally).

CRC:

A cyclic redundancy check (CRC) is an error-detecting code designed to detect accidental changes to raw computer data, and is commonly used in digital networks and storage devices such as hard disk drives.

Exercise -1:

Design UDP Client and Server application to reverse the given input sentence.
[Hint] sample code (**udpclient.c,udpserver.c**) is given in folder.

Exercise-2:

Design the FTP application using UDP Sockets.
Read:
https://www.codeproject.com/Questions/646310/Server-client-file-transfer-using-UDP-in-C

Since UDP Socket does not take care of reliability, your application protocol must include sequence no, acknowledgement no and retransmission.

Raw_udp.c and udp4.c code is given for reference. You can use necessary part from this code.

Follow below guidelines.

Create function for each task.

Packetization :
File will need to be divided into blocks.build a packet of size 1024 B.
Your block will consist of header and data part.Header contains following fields.

Define pseudo header
Sequence no: 1B
Acknowledgement no : 1B
Data length: 2B

So actual data in block will be block size- header size.
Data: 1000 B

In last block of file if data is not 1000 B , You need to do padding. If last block has data of 500 B, data length will be 500.

From server side include acknowledgment no and necessary fields

Selective repeat:

To handle retransmission , out of order packets implement selective repeat.
In selective repeat implement below task.
Buffer management: Build a buffer (window) of 8 packets in sender side and receiver side.

Implement timer function.When timeout happens retransmit unacknowledged packet.
receiver will need to send an acknowledgment for each successfully received datagram(block) and manage out of order packets in buffer.
To perform this experiment you need to know about raw UDP socket. So read following links.

Read

Raw sockets: http://www.cs.binghamton.edu/~steflik/cs455/rawip.txt
http://opensourceforu.com/2015/03/a-guide-to-using-raw-sockets/

UDP checksum : http://www4.ncsu.edu/~mlsichit/Teaching/407/Resources/udpChecksum.html

Advanced TCP/IP - THE RAW SOCKET PROGRAM EXAMPLES
Page 7 to 9: http://www.tenouk.com/download/pdf/Module43.pdf

Submission:

Submit a zip file containing all the codes the naming convention should be as following:

1. zip file name must be lab3UDP_groupid
2. File which contains code must have name ExerciseNumber_Client/ server