Autumn Semester 2012
Lab-2

Part 1

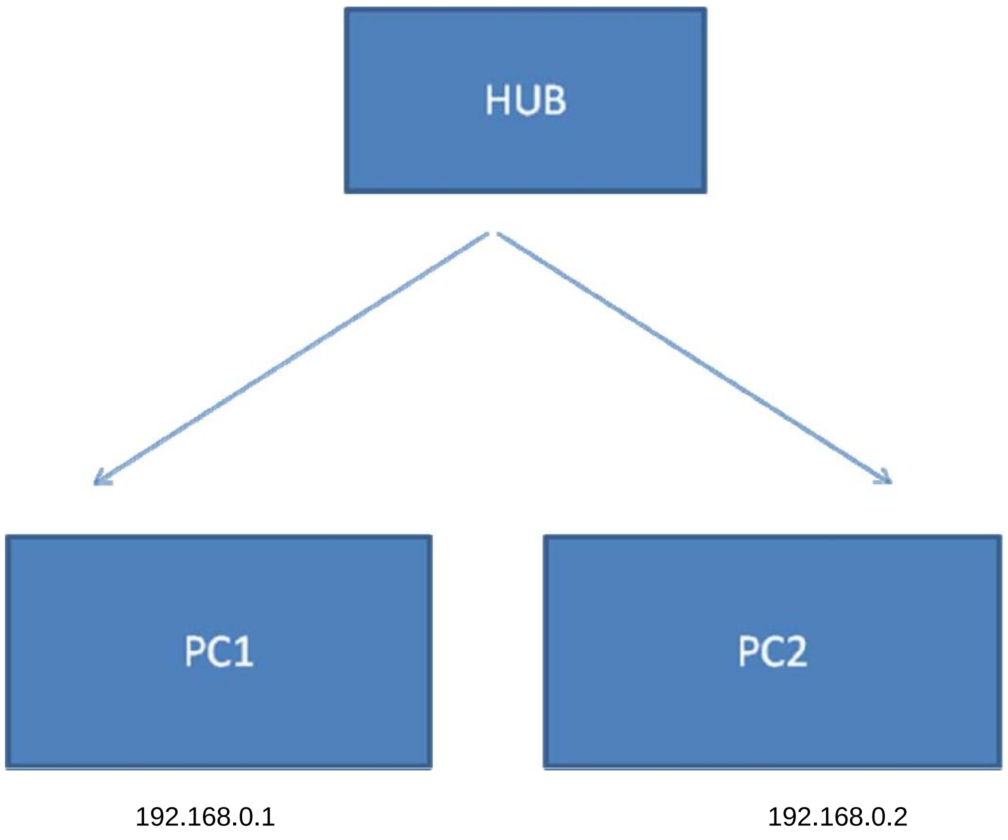
Aim:

Design and Implement an Online Quiz Application using TCP/IP Socket Programming.

Resources:

1.	PCs	2Nos.
2.	NICs	2Nos.
3.	Hub	1No.
4.	Data cables	2No.

Configuration:



Exercise-1:

- Study the application and underlying protocol implemented using TCP Sockets.
- Compile and test the performance of the given implementation.
- Source code for TCP **client** and **server**.

Exercise-2:

- Design the protocol for the same application using UDP Sockets.

Theory

TCP:

TCP is a transport layer protocol used by applications that require guaranteed delivery. It is a sliding window protocol that provides handling for both timeouts and retransmissions.

TCP establishes a full duplex virtual connection between two endpoints. Each endpoint is defined by an IP address and a TCP port number. The operation of TCP is implemented as a finite state machine.

The byte stream is transferred in segments. The window size determines the number of bytes of data that can be sent before an acknowledgement from the receiver is necessary.

UDP:

The User Datagram Protocol offers only a minimal transport service -- non-guaranteed datagram delivery -- and gives applications direct access to the datagram service of the IP layer. UDP is used by applications that do not require the level of service of TCP or that wish to use communications services (e.g., multicast or broadcast delivery) not available from TCP.

UDP is almost a null protocol; the only services it provides over IP are checksumming of data and multiplexing by port number. Therefore, an application program running over UDP must deal directly with end-to-end communication problems that a connection-oriented protocol would have handled -- e.g., retransmission for reliable delivery, packetization and reassembly, flow control, congestion avoidance, etc., when these are required. The fairly complex coupling between IP and TCP will be mirrored in the coupling between UDP and many applications using UDP.

Socket:

In computer networking, an Internet socket or network socket is an endpoint of a bidirectional inter process communication flow across an internet protocol based computer network such as the internet.

The term internet socket is also used as a name for an application programming interface (API) for the TCP/IP protocol stack, usually provided by the operating system. Internet sockets constitute a mechanism for delivering incoming data packets to the appropriate application process or thread, based on a combination of local and remote IP address and port numbers. Each socket is mapped by the operating system to a communicating application process or thread.

A socket address is the combination of an IP address (the location of the computer) and a port (which is mapped to the application program process) into a single identity, much like one end of a telephone connection is the combination of a phone number and a particular extension.

Two types of internet sockets:

1). Stream sockets: Stream sockets are reliable two-way connected communication streams. If you output two items into the socket in the order “1, 2”, they will arrive in the order “1, 2” at the opposite end. They will also be error-free.

2). Datagram sockets: Datagram sockets are sometimes called “connectionless sockets”.

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.

Socket API (Some relevant interfaces) For Connection Oriented (TCP-based)

getaddrinfo();

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

```
int getaddrinfo(const char *node, // e.g. "www.example.com" or
                IP const char *service, // e.g. "http" or port number
                const struct addrinfo *hints,
                struct addrinfo **res);
```

socket():

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

```
int socket(int domain, int type, int protocol);
```

bind():

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

```
int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
```

connect():

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

```
int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```

listen():

```
int listen(int sockfd, int backlog);
```

accept():

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

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

send()

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

```
ssize_t send(int socket, const void *buffer, size_t length, int flags);
```

The send() function shall initiate transmission of a message from the specified socket to its peer. The send() function shall send a message only when the socket is connected (including when the peer of a connectionless socket has been set via connect()).

recv()

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

```
ssize_t recv(int socket, void *buffer, size_t length, int flags);
```

The recv() function shall receive a message from a connection-mode or connectionless-mode socket. It is normally used with connected sockets because it does not permit the application to retrieve the source address of received data.

Socket API for Connectionless communication (UDP-based)

sendto()

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

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

The sendto() function shall send a message through a connection-mode or connectionless-mode socket. If the socket is connectionless-mode, the message shall be sent to the address specified by dest_addr. If the socket is connection-mode, dest_addr shall be ignored.

recvfrom()

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

```
ssize_t recvfrom(int socket, void *buffer, size_t length, int flags, struct sockaddr
                                                *address, socklen_t *address_len);
```

The recvfrom() function receives a message from a connection-mode or connectionless-mode socket. It is normally used with connectionless-mode sockets because it permits the application to retrieve the source address of received data.

Usage of ctime():

```
char * ctime ( const time_t * ptr_time );
```

Parameters: Pointer ptr_time to a time_t object that contains a calendar time.

Return Value:

The function returns a C string containing the date and time information. The string is followed by a new-line character ('\n') Converts the time_t object pointed by timer to a C string containing a human-readable version of the corresponding local time and date.

The functions ctime and as ctime share the array which holds this string. If either one of these functions is called, the content of the array is overwritten.

Explanation:

The string that is returned will have the following format: **Www Mmm dd hh:mm:ss yyyy**

Www = which day of the week.

Mmm = month in letters.

dd = the day of the month.

hh:mm:ss = the time in hour, minutes, seconds.

yyyy = the year.

Source code example of ctime():

```
#include <stdio.h>
#include <time.h>

int main ()
{
        time_t time_raw_format;

        time ( &time_raw_format );
        printf ( "The current local time: %s", ctime(&time_raw_format));

        return 0;
}
```

time()

Syntax:

```
#include <time.h>
time_t time( time_t *time );
```

Description:

The function time() returns the current time, or -1 if there is an error. If the argument time is given, then the current time is stored in time.

Lab02_tcpclient.c

```
#include<sys/ioctl.h>
#include<arpa/inet.h>
#include<stdio.h>
#include<stdlib.h>
#include<net/if_arp.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<string.h>

#define FNAME file1
#define PORT 3447
#define BUFSIZE 128

int
main(int argc, char **argv)
{
	int sockfd, fd, n, size,count=0,fd1,i;
	long int size1,size2;
	char buf[BUFSIZE],  fname[50],buf1[BUFSIZE],content[30],uname[30];
	struct sockaddr_in servaddr;

	if (argc != 2) {
	printf("Usage: %s server_address", argv[0]);
	exit(1);
	}


	if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
		exit(1);

	bzero(&servaddr, sizeof(servaddr));
	servaddr.sin_family = AF_INET;
	servaddr.sin_port   = htons(PORT);
	if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0)
		exit(1);



	if (connect(sockfd, (struct sockaddr*) &servaddr, sizeof(servaddr))
< 0)
		exit(1);
	printf("connection established\n");

	printf("Enter the User-name : ");
	scanf("%s",uname);
	 strcpy(fname,"questions");
	send(sockfd,fname,sizeof(fname),0);
	 send(sockfd,uname,sizeof(uname),0);


	//read(sockfd, size1, sizeof(size1));

	//printf("Size of the File is : %d ",size1);

	fd=open(fname,O_WRONLY|O_CREAT,S_IRWXU);
```

```
      while ( (n = read(sockfd, buf, BUFSIZE-1)) > 0)
      {
            buf[n] = '\0';
            printf("%s\n",buf);
            write(fd,buf,n);
            if( n < BUFSIZE-2)
                  break;
      }

//.........................................................


    if(fork())
    {
        //printf("\nEnter the data to be send type exit for stop:\n");
        scanf("%s",content);

        while(strcmp(content,"exit")!=0)
        {
            send(sockfd,content,30,0);
            scanf("%s",content);
        }
        send(sockfd,"exit",5,0);
    }
    else
    {
        i=recv(sockfd,content,30,0);

        while(strcmp(content,"exit")!=0)
        {
            printf("\nServer: %s\n",content);
            i=recv(sockfd,content,30,0);
        }
        send(sockfd,"exit",5,0);
    }
//........................................................

      close(sockfd);
      close(fd);
            close(fd1);

      exit(0);
}
```

Lab02_tcpserver.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <fcntl.h>


#define FNAME file1
#define PORT 3447
#define BUFSIZE 128
#define LISTENQ 5

int main(int argc, char **argv)
{
      int   listenfd, connfd, fd, pid, n, size,i,count=10;
//    FILE *fp;
      struct sockaddr_in    servaddr;
      char buf[BUFSIZE],fname[50],content[30],content1[30],uname[30];

      struct stat stat_buf;


      listenfd = socket(AF_INET, SOCK_STREAM, 0);

      bzero(&servaddr, sizeof(servaddr));
      servaddr.sin_family      = AF_INET;
      servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
      servaddr.sin_port        = htons(PORT);

      bind(listenfd, (struct sockaddr *) &servaddr, sizeof(servaddr));

      listen(listenfd, LISTENQ);
            printf("listening\n");

      for ( ; ; )
      {
            connfd = accept(listenfd, (struct sockaddr *) NULL, NULL);
            pid=fork();
            if(pid==0)
            {
                  printf("Handling connection request\n");
                  recv(connfd,fname,50,0);
                          recv(connfd,uname,30,0);
                  printf("File name is %s :  ", fname);
                          printf("user name is:%s\n",uname);
                  fd=open(fname,O_RDONLY,S_IRUSR);
                  fstat(fd, &stat_buf);
                  size = stat_buf.st_size;
                  printf(" size is %d\n", size);
                  printf("\nopened file\n");
                  while ( (n = read(fd, buf, BUFSIZE-1)) > 0)
                  {
                        buf[n] = '\0';
                        //printf("%s\n",buf);
                        write(connfd,buf,n);
                  }
```

```
                printf("file transfer completed \n");

                         //.........................................

                        /*if(fork())
                        {
//printf("\nEnter the data to be send type exit for stop:\n");
                          scanf("%s",content);
                   while(strcmp(content,"exit")!=0)
                {
                      send(connfd,content,30,0);
                      scanf("%s",content);
                }

                  send(connfd,"exit",5,0);
                        }
                else*/
                i = recv(connfd,content,30,0);
                //while(strcmp(content,"result")!=0)
                //{
                printf("\nClient: %s\n",content);
                strcpy(content1,content);
                //i=recv(connfd,content,30,0);
                //}
                printf("\nMarks sent to client:");
    // write(connfd,&count,sizeof(count));
//int A = 5;
//const char* pBytesOfA = (const char*)&A;
//int lengthOfBytes = sizeof(A);
//send(connfd,pBytesOfA,lengthOfBytes,0);
                        strcpy(content,uname);
                if(content1[0]=='b'&&content1[1]=='b')
                        strcat(content," score is : 2");
                else if(content1[0]=='b'||content1[1]=='b')
                        strcat(content," score is : 1");
                else
                        strcat(content," score is : 0");
                send(connfd,content,30,0);

                           //.........................................
                close(connfd);
                close(fd);
                exit(0);
          }
                printf("now child process killed");
     }
}
```

This Question will be send to client after connection established

......................................Questions................................

1. find the next number in the series

   1,1,2,3,5,_

   A) 7

   B) 8

   C) 9

   D) I don't know

2. find the velue of "sqrt(144)"

A) 10

B) 12

C) question was not clear

D) I don't know

......................................EOF Questions........................

Note: please enter the answers in sequence. don't give any space between answers(enter in a sequence).

After that type "result" on client side then you will get your score card.

get your score card.

Example for Exercise-2 based on UDP (Connectionless communication)

UDPFiletransfer_server.c

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

#define BUF_LEN 10000
#define PORT 11000

void err(char *s)
{
    perror(s);
    exit(1);
}

int main(int argc, char *argv[])
{
  struct sockaddr_in my_addr, cli_addr; // structure to specify the socket
address(IP address + port + family)
  int sockfd;                          // socket desciptor
  int slen = sizeof(cli_addr);              // length of the address
structure
  char buffer[BUF_LEN];                     // buffer to store data



  // create a socket and store its descriptor for using later

if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
    err("socket");
  else
    printf("\n Server : Socket creation successful");

  // assign 0 to the address structure to flush it

  bzero(&my_addr, sizeof(my_addr));
  my_addr.sin_family = AF_INET;  //assign family
  my_addr.sin_port = htons(PORT); //assign port number
  my_addr.sin_addr.s_addr = htonl(INADDR_ANY);   // accept data from any
address

  // bind the socket to that ip address, so that clients can send data to
the ip
address and it gets delivered to the socket
  if(bind(sockfd, (struct sockaddr*) &my_addr, sizeof(my_addr)) == -1)
    err("bind");
  else
    printf("\n Server : bind successful");

  while(1)
  {

 // receive data from the client on the specified socket(IP Address + port
number)

 if(recvfrom(sockfd, buffer, BUF_LEN, 0, (struct sockaddr*)&cli_addr,
&slen) == -1)
      err("receiving error");

      // open the file specified
      int fp = open(buffer, O_RDONLY);
      bzero(buffer, sizeof(buffer));

      // read the first BUF_LEN bytes of the file
      read(fp, buffer, sizeof(buffer));

      // send the file data to the client address obtained in recvfrom

if(sendto(sockfd, buffer, BUF_LEN, 0, (struct sockaddr*)&cli_addr, slen) ==
-1)
      err("Client : sending of file data failed");
  }

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

UDPFiletransfer_client.c

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

#define BUF_LEN 10000
#define SERVER_PORT 11000
#define CLIENT_PORT 11001

void err(char *s)
{
    perror(s);
    exit(1);
}

int main(int argc, char *argv[])
{
  struct sockaddr_in my_addr, serv_addr;    // structure to specify the
socket address(IP address + port + family)
  int sockfd, slen = sizeof(serv_addr);           // socket desciptor and
length of sockaddr_in
  char buffer[BUF_LEN];                     // buffer to store data

  if(argc != 2) // check if ip address is specified or not, name of the op
file also counts as argument
  {
    printf("Not enough arguments");
    exit(0);
  }

  // Create a udp socket of family AF_INET. SOCK_DGRAM specify datagram
socket
  if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
    err("Client : socket creation failed");
  else
    printf("\n Client : Socket creation successful");

  //initialize client_addr struct with 0s
  bzero(&my_addr, sizeof(my_addr));
  my_addr.sin_family = AF_INET;  // assign family to server address
structure
  my_addr.sin_port = htons(CLIENT_PORT);    // assign port to sockaddr_in,
same as specified in server
  my_addr.sin_addr.s_addr = htonl(INADDR_ANY);   // accept data from any
address
```

```
  // bind the socket to that ip address, so that clients can send data to
the ip a address and it gets delivered to the socket
  if(bind(sockfd, (struct sockaddr*) &my_addr, sizeof(my_addr)) == -1)
    err("bind");
  else
    printf("\n Client : bind successful");

  // initialize serv_addr struct with 0s
  bzero(&serv_addr, sizeof(serv_addr));
  serv_addr.sin_family = AF_INET;     // assign family to server address
structure
  serv_addr.sin_port = htons(SERVER_PORT);  // assign port to sockaddr_in,
same as specified in server
  // s_addr is of long type so convert our IP address from decimal notation
to long using inet_aton
  if(inet_aton(argv[1], &serv_addr.sin_addr) == 0)
  {
      err("address assignment failed");
  }

  // to keep the program alive we have an infinite loop
  while(1)
  {
      printf("\n Enter name of the file to get(not greater than 2KB) : ");
      scanf("%[^\n]", buffer);
      getchar(); // to accomodate the enter

      // send the file name to the server address using socket created

if(sendto(sockfd, buffer, BUF_LEN, 0, (struct sockaddr*)&serv_addr, slen)
== -1)
     err("Client : sending of file name failed");

      // receive data from the server on the specified socket(IP Address +
port number)

if(recvfrom(sockfd, buffer, BUF_LEN, 0, (struct sockaddr*)&serv_addr,
&slen) == -1)
     err("receiving error");

      // write the data to a file or display it on screen.
      printf("Received from %s:%d\nData : %s\n\n",
inet_ntoa(serv_addr.sin_addr), ntohs(serv_addr.sin_port), buffer);
  }

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