REPORT
Lab Assignment 2
Course: Computer Networks (CNE532)

Question 1:

Theory

UDP : In UDP, the client does not form a connection with the server like in TCP and instead just sends a datagram. Similarly, the server need not accept a connection and just waits for datagrams to arrive. Datagrams upon arrival contain the address of the sender which the server uses to send data to the correct client.

select() : select() allows a program to monitor multiple file descriptors,wait until one or more of the file descriptors become "ready" for some class of I/O operation (e.g., input possible). A file descriptor is considered ready if it is possible to perform a corresponding I/O operation (e.g., read, or a sufficiently small write) without blocking.
Using select changes made are users need not wait for message arrival for data to be sent. Non blocking text chat is made possible.

UDP Server :

1. Create UDP socket.
2. Bind the socket to the server address.
3. One time ping received from a client.
4. Can send or receive data simultaneously without blockage.

UDP Client :

1. Create UDP socket.
2. Ping the server.
3. Send and receive messages from the server without blocking.

Code Explanation

Functions

Returns socket file descriptor.Creates an unbound socket in the specified domain.

- *int socket(int domain, int type, int protocol)*

Assigns address to the unbound socket.

- *int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)*

Send a message on socket

- *sendto(int sockfd, const void *buf, size_t len, int flags,const struct sockaddr *dest_addr socklen_t addrlen)*

Receive a message from the socket.

- *recvfrom(int sockfd, void *buf, size_t len, int flags, sockaddr *src_addr, socklen_t *addrlen)*

Close a file descriptor
- int close(int fd)
Select() which monitores activity on socketfd.
*- int select(int nfds, fd_set *readfds, fd_set *writefds,fd_set *exceptfds, struct timeval *timeout);*
Clears file descriptor in the set
*- void FD_CLR(int fd, fd_set *set);*
Checks if file descriptor is SET
*- int FD_ISSET(int fd, fd_set *set);*
SET the file descriptor
*- void FD_SET(int fd, fd_set *set);*

Logic

- Set File Descriptor when server is idle i.e. free to send messages

```
FD_SET(0,&nset);
```

- Set File Descriptor at which server listens for incoming messages.

```
FD_SET(sockfd,&nset);
```

- Select File Descriptor based on activity on file descriptors.

```
int ret=select(10,&nset,0,0,0);
```

- Catch Error in Select

```
if (ret==-1)
{
        printf("SELECT Error\n");
        close(sockfd);
        return 0;
}
```

- Checks if 0 is set means the server can send messages.

else if (FD_ISSET(0,&nset))

- Checks if sockfd is set means server catches activity on socket (incoming message).

else if (FD_ISSET(sockfd,&nset))

Question 2:

Theory

TCP_ : A single physical connector can serve multiple connections. Each side of a socket connection uses its own port number, which does not change during the life of that connection. The port number and IP address together uniquely identify an endpoint. Together, two endpoints are considered a 'socket'.

TCP Server –

- using create(), Create TCP socket.
- using bind(), Bind the socket to server address.
- using listen(), put the server socket in a passive mode, where it waits for the client to approach the server to make a connection
- using accept(), At this point, connection is established between client and server, and they are ready to transfer data.

- Receive Request from client.
- Send Response to client
- Close clientfd

TCP Client –

- Create TCP socket.
- connect the newly created client socket to the server.
- Request Server for time
- Close socketfd

Code Explanation

Functions

- SOCKET : *Returns socket file descriptor.Creates an unbound socket in the specified domain.*
  - int socket(int domain, int type, int protocol)
- BIND : *Assigns address to the unbound socket.*
  *- int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)*
- CLOSE : *Close a file descriptor*
  - int close(int fd)
- serv_addr.sin_family=AF_INET *It contains the code for the address family and is always AF_INET, indicating the internet domain.*
- serv_addr.sin_addr.s_addr=INADDR_ANY*; here sin_addr has one member s_addr which is used to hold the IP address of the machine.*
- serv_addr.sin_port=htons(portno)*; here we hav to store port number into he sin_port member and this takes on the network bute order.*
- *listen() – call allows a process to listen on socket for communication. usage: listen(socket_fd, no_of_waiting_connections); so it takes in a socket file descriptor and the no. of connections waiting while the process is handling a particular connection.*
- accept*() – a system call that causes the process to block until the client connects to the server. it returns a new descriptor int accept(sockfd,pointer_to_address_of client)*
- read(fd,buffer,4096*); this reads the contents of the file into the buffer. In socket programming, all communications happens using the buffer both at client and server side. With the completion of this read , the contents of the file is residing in the buffer and is ready to be sent to the client.*
- write(newsockfd,buffer,4096)*; this is the final command which writes the contents of the buffer(which has the file content) into the socket using the newsockfd which finally delivers it to the client process.*
- connect*() – is a function used by the client to establish a connection to the server. It takes 3 arguments: usage: connect(sockfd, host_to_which_itconnects, sizeof_addr); Eg: connect(sockfd,(struct sockaddr*)&serv_addr,sizeof(Serv_addr))<0);*

Logic

Time Generate

- Include package to find time

  #include<time.h>

- Declare time_t variable to store time value

  time_t ticks;

- Initialise variable as Null in respect to time object

    ticks = time(NULL);

- Find time using ctime() and store it in variable ‘snd’.

    snprintf(snd, sizeof(snd), "%.24s\r\n", ctime(&ticks));

Question 3:

Theory

UDP : In UDP, the client does not form a connection with the server like in TCP and instead just sends a datagram. Similarly, the server need not accept a connection and just waits for datagrams to arrive. Datagrams upon arrival contain the address of the sender which the server uses to send data to the correct client.

UDP Server :

1. Create UDP socket.
2. Bind the socket to server address.
3. Wait until client sends HOST name.
4. Find IP of concerned host.
5. Send reply to client.
6. Go back to Step 3.

UDP Client :

1. Create UDP socket.
2. Get HOST name as input from user.
3. Send message to the server.
4. Wait until a response from the server is received.
5. Print the response.
6. Close socket descriptor and exit.

Code Explanation

Functions

Returns socket file descriptor.Creates an unbound socket in the specified domain.

- *int socket(int domain, int type, int protocol)*

Assigns address to the unbound socket.

- *int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)*

Send a message on socket

- *sendto(int sockfd, const void *buf, size_t len, int flags,const struct sockaddr *dest_addr socklen_t addrlen)*

Receive a message from the socket.

- *recvfrom(int sockfd, void *buf, size_t len, int flags, sockaddr *src_addr, socklen_t *addrlen)*

Close a file descriptor

- int close(int fd)

gethostbyname() : The gethostbyname function retrieves host information corresponding to a host name from a host database.

*- struct hostent * gethostbyname (const char *name)*

inet_ntoa() : *The inet_ntoa function converts an (Ipv4) Internet network address into an ASCII string in Internet standard dotted-decimal format.*

*- inet_ntoa(*addr_list[i])*

Logic

- HOST TO IP

```
int hostname_to_ip(char * hostname , char* ip)
{
   struct hostent *he;
   struct in_addr **addr_list;
   int i;
  TO Retrieve HOST name
   if ( (he = gethostbyname( hostname ) ) == NULL)
   {
Getting HOST info
      herror("gethostbyname");
      return 1;
   }
   addr_list = (struct in_addr **)he->h_addr_list;
Iterating address list
  for(i = 0; addr_list[i] != NULL; i++)
   {
To convert an Internet network address into ASCII string
      strcpy(ip , inet_ntoa(*addr_list[i]) );
      return 0;
   }
   return 1;
}
```

Question 4:

Theory

TCP_ : A single physical connector can serve multiple connections. Each side of a socket connection uses its own port number, which does not change during the life of that connection. The port number and IP address together uniquely identify an endpoint. Together, two endpoints are considered a 'socket'.

UDP : In UDP, the client does not form a connection with the server like in TCP and instead just sends a datagram. Similarly, the server need not accept a connection and

just waits for datagrams to arrive. Datagrams upon arrival contain the address of the sender which the server uses to send data to the correct client.

select() : select() allows a program to monitor multiple file descriptors,wait until one or more of the file descriptors become "ready" for some class of I/O operation (e.g., input possible). A file descriptor is considered ready if it is possible to perform a corresponding I/O operation (e.g., read, or a sufficiently small write) without blocking.
Using select changes made are users need not wait for message arrival for data to be sent. Non blocking text chat is made possible.

Server:

1. Create TCP i.e Listening socket
2. Create a UDP socket
3. Bind both sockets to the server address.
4. Initialize a descriptor set for select and calculate maximum of 2 descriptor for which we will wait
5. Call select and get the ready descriptor(TCP or UDP)
6. Handle new connection if ready descriptor is of TCP OR receive data gram if ready descriptor is of UDP

TCP Client:

1. Create a TCP socket.
2. Call connect to establish connection with server
3. When the connection is accepted write message to server
4. Read response of Server
5. Close socket descriptor and exit.

UDP Client:

1. Create UDP socket.
2. Send a message to the server.
3. Wait until a response from the server is received.
4. Close socket descriptor and exit.

Code Explanation

Functions

- SOCKET : *Returns socket file descriptor.Creates an unbound socket in the specified domain.*
  - int socket(int domain, int type, int protocol)
- BIND : *Assigns address to the unbound socket.*
  *- int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)*
- SENDTO *: Send a message on socket*
  *- sendto(int sockfd, const void *buf, size_t len, int flags,const struct sockaddr *dest_addr socklen_t addrlen)*
- RECVFROM *:Receive a message from the socket.*
  *- recvfrom(int sockfd, void *buf, size_t len, int flags, sockaddr *src_addr, socklen_t *addrlen)*

- CLOSE : *Close a file descriptor*
  - int close(int fd)
- serv_addr.sin_family=AF_INET *It contains the code for the address family and is always AF_INET, indicating the internet domain.*
- serv_addr.sin_addr.s_addr=INADDR_ANY*; here sin_addr has one member s_addr which is used to hold the IP address of the machine.*
- serv_addr.sin_port=htons(portno)*; here we hav to store port number into he sin_port member and this takes on the network bute order.*
- *listen() – call allows a process to listen on socket for communication. usage: listen(socket_fd, no_of_waiting_connections); so it takes in a socket file descriptor and the no. of connections waiting while the process is handling a particular connection.*
- accept*() – a system call that causes the process to block until the client connects to the server. it returns a new descriptor int accept(sockfd,pointer_to_address_of client)*
- bzero(buffer,4096); *n=read(newsockfd,buffer,4096); this initializes the buffer to zero and then reads the content from the socket (using the newsockfd) into the buffer.*
  fd=open(buffer,O_RDONLY) opens the file requested by the client in the read only mode.
- read(fd,buffer,4096*); this reads the contents of the file into the buffer. In socket programming, all communications happens using the buffer both at client and server side. With the completion of this read , the contents of the file is residing in the buffer and is ready to be sent to the client.*
- write(newsockfd,buffer,4096)*; this is the final command which writes the contents of the buffer(which has the file content) into the socket using the newsockfd which finally delivers it to the client process.*
- connect*() – is a function used by the client to establish a connection to the server. It takes 3 arguments: usage: connect(sockfd, host_to_which_itconnects, sizeof_addr); Eg: connect(sockfd,(struct sockaddr*)&serv_addr,sizeof(Serv_addr))<0);*
- Select*() which monitores activity on socketfd.*
  *- int select(int nfds, fd_set *readfds, fd_set *writefds,fd_set *exceptfds, struct timeval *timeout);*
- FD_CLR *:Clears file descriptor in the set*
  *- void FD_CLR(int fd, fd_set *set);*
- FD_ISSET *: Checks if file descriptor is SET*
  *- int FD_ISSET(int fd, fd_set *set);*
- FD_SET *sets the file descriptor*
  *- void FD_SET(int fd, fd_set *set);*

Logic

- Set File Descriptor when server is idle i.e. free to send messages
  FD_SET(tcpfd, &fdset);
- Set File Descriptor at which server listens for incoming messages.
  FD_SET(udpfd, &fdset);
- Select File Descriptor based on activity on file descriptors.
  sel = select(maxfdp1, &fdset, NULL, NULL, NULL);
- Catch Error in Select

```
if (sel==-1)
{
        printf("SELECT Error\n");
        return 0;
}
```

- Checks if tcpfd is set means the server will receive messages from TCP.

```
if (FD_ISSET(tcpfd, &fdset)) {
        len = sizeof(client_addr);
        Accepting Connection and Processing Request
        clientfd = accept(tcpfd, (struct sockaddr*)&client_addr, &len);
        if ((childpid = fork()) == 0) {
                write(clientfd, snd, sizeof(snd));
                printf("TCP request completed\n");
                }
        close(clientfd);
}
```

- Checks if tcpfd is set means the server will receive messages from UDP.

```
if (FD_ISSET(udpfd, &fdset)) {
        n = recvfrom(udpfd, buffer, sizeof(buffer), 0, (struct sockaddr*)&client_addr, &len);
        Convert HOST to IP
        int signa = hostname_to_ip(hostname , ip);
        sendto(udpfd, ip, sizeof(ip), 0,(struct sockaddr*)&client_addr, sizeof(client_addr));
        printf("UDP request completed\n");
}
```