PB_05_Kushagra Suryawanshi

CN Lab Assignment 7

Title: UDP Socket

Aim: Write a C program for wired network using UDP socket to perform any one of the following operations

a. String Conversion from Upper Case to Lower Case.

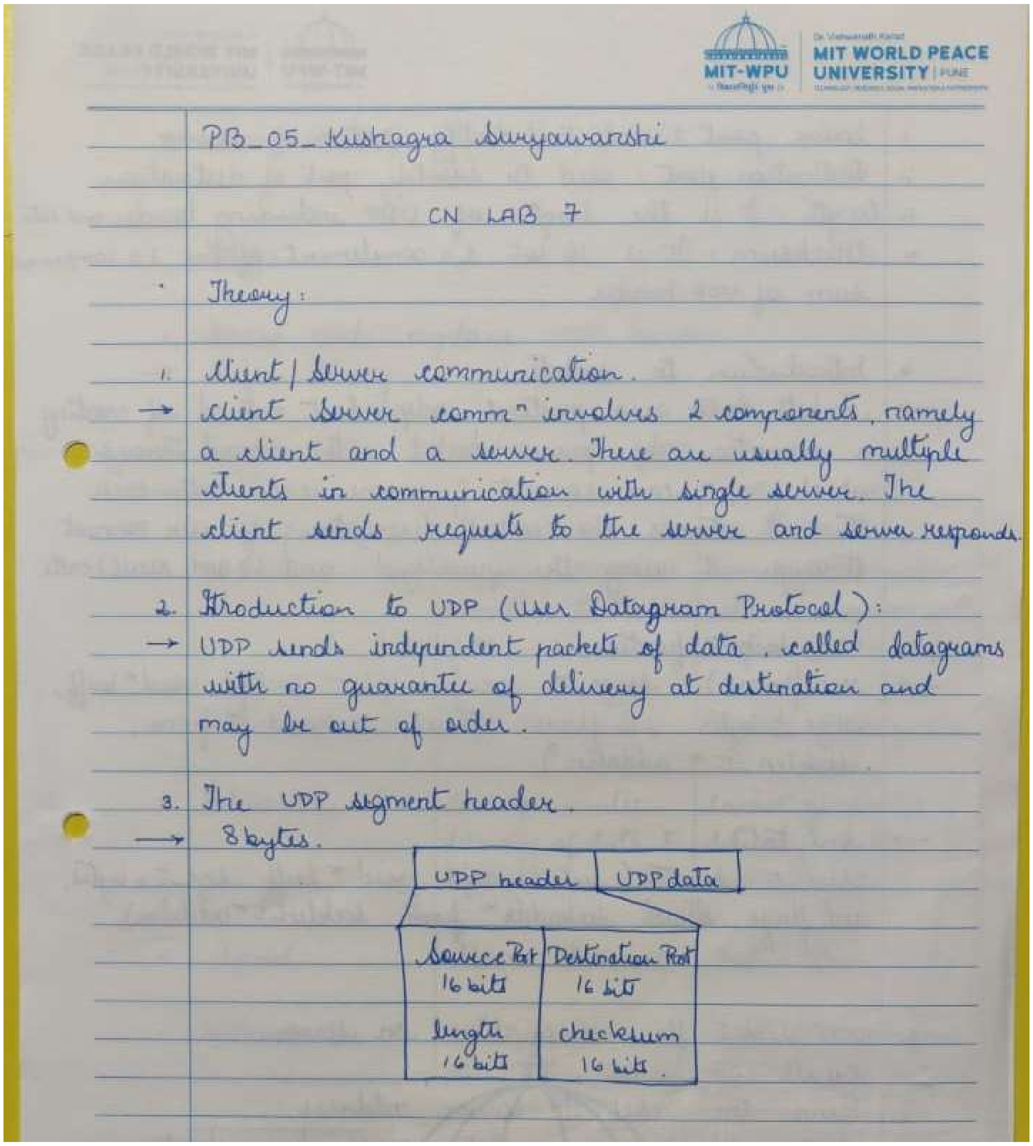
Objectives:

1.To understand concept of socket programming using UDP.

Steps to implement:

1. Write c code for client and server on an editor.
2. Compile and create an exe file for client and server.
3. Use ‘. /Server1.exe <ipaddress>’ to execute the server script and ‘./Client1.exe <ipaddress>’ to execute client script (USE TWO SEPARATE TERMINALS).
4. Send uppercase string through client side. Server will return lowercase conversion

Theory:



PB_05_Kushagra Suryawanshi

CN LAB 7

• Theory:

1. Client / Server communication.

→ client Server comm^n involves 2 components, namely a client and a server. There are usually multiple clients in communication with single server. The client sends requests to the server and server responds.

2. Itroduction to UDP (User Datagram Protocol):

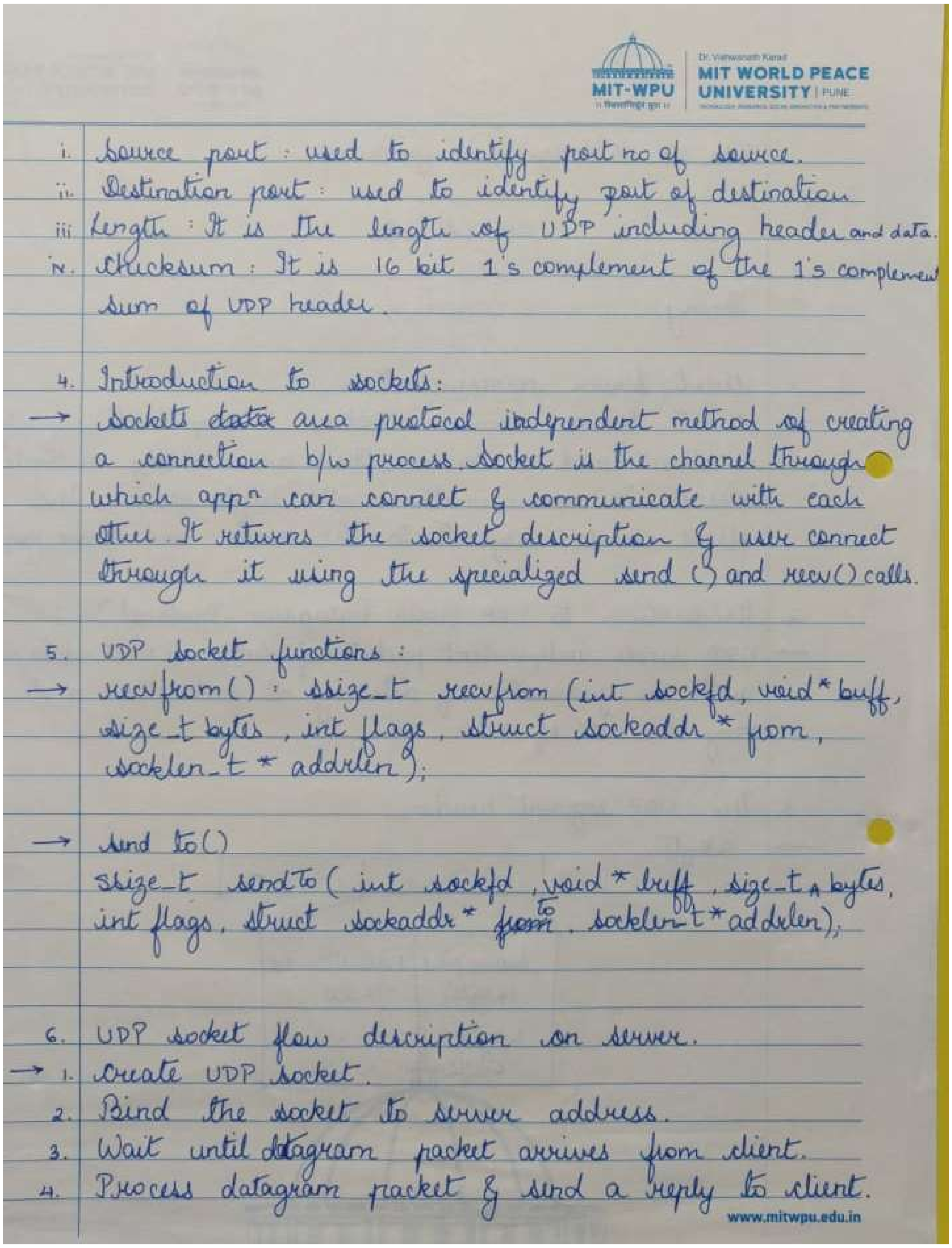
→ UDP sends independent packets of data, called datagrams with no guarantee of delivery at destination and may be out of order.

3. The UDP segment header.

→ 8 bytes.

UDP header	UDP data

Source Port 16 bits	Destination Port 16 bits
length 16 bits	checksum 16 bits.



MIT-WPU | Dr. Vishwanath Karad MIT WORLD PEACE UNIVERSITY | PUNE

i. Source port : used to identify port no of source.

ii. Destination port : used to identify port of destination

iii Length : It is the length of UDP including header and data.

iv. Checksum : It is 16 bit 1's complement of the 1's complement sum of UDP header.

4. Introduction to sockets:

→ Sockets ~~data~~ area protocol independent method of creating a connection b/w process. Socket is the channel through which appⁿ can connect & communicate with each other. It returns the socket description & user connect through it using the specialized send () and recv() calls.

5. UDP socket functions :

→ recvfrom() : ssize_t recvfrom (int sockfd, void* buff, size_t bytes, int flags, struct sockaddr* from, socklen_t * addrlen);

→ send to()

ssize_t sendto (int sockfd, void * buff, size_t bytes, int flags, struct sockaddr* ~~from~~ to, socklen_t * addrlen);

6. UDP socket flow description on server.

→ 1. Create UDP socket.

2. Bind the socket to server address.

3. Wait until datagram packet arrives from client.

4. Process datagram packet & send a reply to client.

www.mitwpu.edu.in

Server Algorithm:

1. Create UDP socket.
2. Bind the socket to server address.
3. Wait until datagram packet arrives from client.
4. Process the datagram packet and send a reply to client.
5. Go back to Step 3.

Client Algorithm:

1. Create UDP socket.
2. Send message to server.
3. Wait until response from server is received.
4. Process reply and go back to step 2, if necessary.
5. Close socket descriptor and exit.

Server code:

```
#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 BUFLEN 512

#define PORT 9930

void err (char *str)

{

    perror (str);

    exit (1);

}



int main(void)

{

    struct sockaddr_in my_addr, cli_addr;

    int sockfd, i,connfd;

    socklen_t slen=sizeof (cli_addr) ;

    char buf [BUFLEN], buf1[BUFLEN];

    if ((sockfd = socket (AF_INET, SOCK_DGRAM, IPPROTO_UDP))==-1)
```

```
{
    err ("socket");
}
else
{
    printf ("Server: Socket() successful\n");
}


bzero (&my_addr,sizeof(my_addr));
my_addr.sin_family= AF_INET;
my_addr.sin_port = htons (PORT);
my_addr.sin_addr.s_addr = htonl (INADDR_ANY);
if (bind (sockfd,(struct sockaddr*)&my_addr,sizeof(my_addr))==-1)
    err ("bind");
else
    printf ("Server: bind() successful\n");


while (1)
{
if (recvfrom (sockfd, buf, BUFLEN, 0, (struct sockaddr*) &cli_addr, &slen)==-1)
                                {
                                            err ("recvfrom()");
                                }
    printf ("Received packet from %s: %d\nData: %s\n\n",inet_ntoa (cli_addr.sin_addr), ntohs
(cli_addr.sin_port), buf);
```

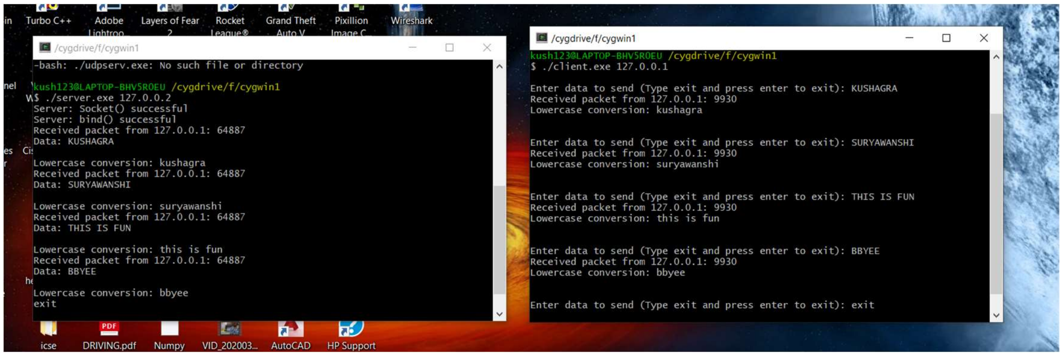
```
                        printf("Lowercase conversion: ");
                        buf[BUFLEN] = puts(strlwr(buf));
                        if(sendto (sockfd,buf, BUFLEN, 0, (struct sockaddr*) &cli_addr, slen)== -1)
                        {
                                    err("sendto()");
                        }
    }
return 0;
}
```

Client code:

```
#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 BUFLEN 512
#define PORT 9930

void err (char *str)
{
    perror (str);
    exit (1);

}


int main(int argc, char** argv)

{

    struct sockaddr_in serv_addr;

    int sockfd, i, slen=sizeof (serv_addr);

    char buf [BUFLEN];

    if (argc !=2)

    {

        printf("Usage: %s <Server-IP>\n",argv[0]);

                        exit(0);

    }

    if ((sockfd = socket (AF_INET, SOCK_DGRAM, IPPROTO_UDP))==-1)

        err ("socket");


    bzero (&serv_addr, sizeof (serv_addr));

            serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    serv_addr.sin_family= AF_INET;

    serv_addr.sin_port = htons(PORT);

    if (inet_aton (argv[1], &serv_addr.sin_addr) ==0)

    {

        fprintf(stderr, "inet_aton () failed\n");

        exit (1);
    }

    while (1)

    {



        printf ("\nEnter data to send (Type exit and press enter to exit): ");

        scanf("%[^\n]", buf);

                        getchar();

        if (strcmp (buf, "exit") == 0)

                        {

                                    exit(0);

        }

        if (sendto (sockfd, buf, BUFLEN, 0,(struct sockaddr*) &serv_addr, slen)==-1)

                        {

                                    err ("sendto ()") ;

        }

        if (recvfrom(sockfd, buf, BUFLEN, 0, (struct sockaddr*)&serv_addr, &slen)==-1)

                        {

                                    err ("client recvfrom()");

        }

        printf ("Received packet from %s: %d\nLowercase conversion: %s\n\n",
inet_ntoa(serv_addr.sin_addr), ntohs(serv_addr.sin_port), buf);

    }

            close (sockfd);

            return 0;
}
```

OUTPUT SCREENSHOTS:

SERVER SIDE and CLIENT SIDE:



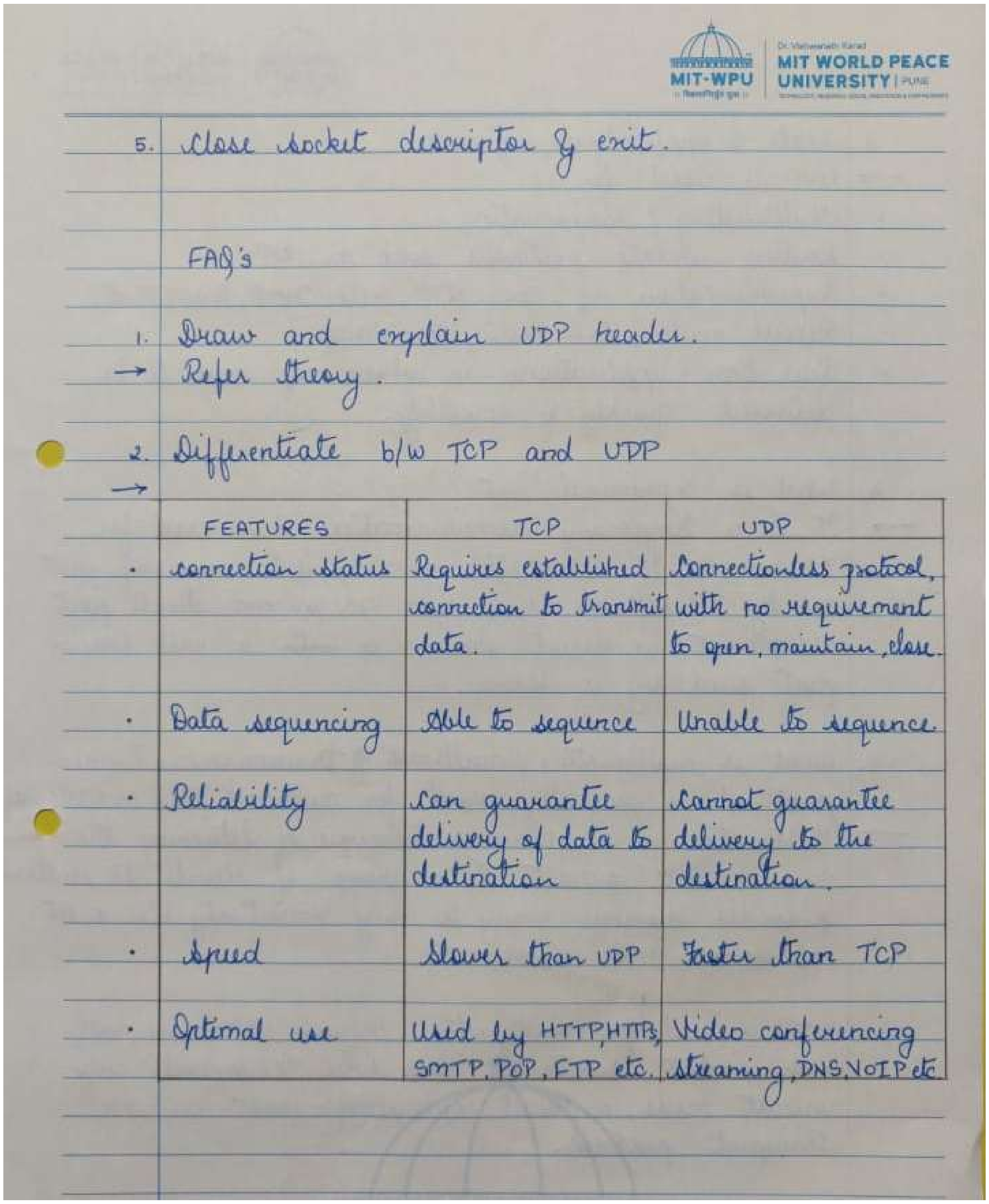
```
-bash: ./udpserv.exe: No such file or directory

kush123@LAPTOP-BHV5ROEU /cygdrive/f/cygwin1
$ ./server.exe 127.0.0.2
Server: Socket() successful
Server: bind() successful
Received packet from 127.0.0.1: 64887
Data: KUSHAGRA

Lowercase conversion: kushagra
Received packet from 127.0.0.1: 64887
Data: SURYAWANSHI

Lowercase conversion: suryawanshi
Received packet from 127.0.0.1: 64887
Data: THIS IS FUN

Lowercase conversion: this is fun
Received packet from 127.0.0.1: 64887
Data: BBYEE

Lowercase conversion: bbyee
exit
```

```
kush123@LAPTOP-BHV5ROEU /cygdrive/f/cygwin1
$ ./client.exe 127.0.0.1

Enter data to send (Type exit and press enter to exit): KUSHAGRA
Received packet from 127.0.0.1: 9930
Lowercase conversion: kushagra


Enter data to send (Type exit and press enter to exit): SURYAWANSHI
Received packet from 127.0.0.1: 9930
Lowercase conversion: suryawanshi


Enter data to send (Type exit and press enter to exit): THIS IS FUN
Received packet from 127.0.0.1: 9930
Lowercase conversion: this is fun


Enter data to send (Type exit and press enter to exit): BBYEE
Received packet from 127.0.0.1: 9930
Lowercase conversion: bbyee


Enter data to send (Type exit and press enter to exit): exit
```

Students Observation:

- To use Linux libraries in Windows OS, Cygwin terminal should be installed.
- Client and server scripts should be executed in different terminals.

FAQS:



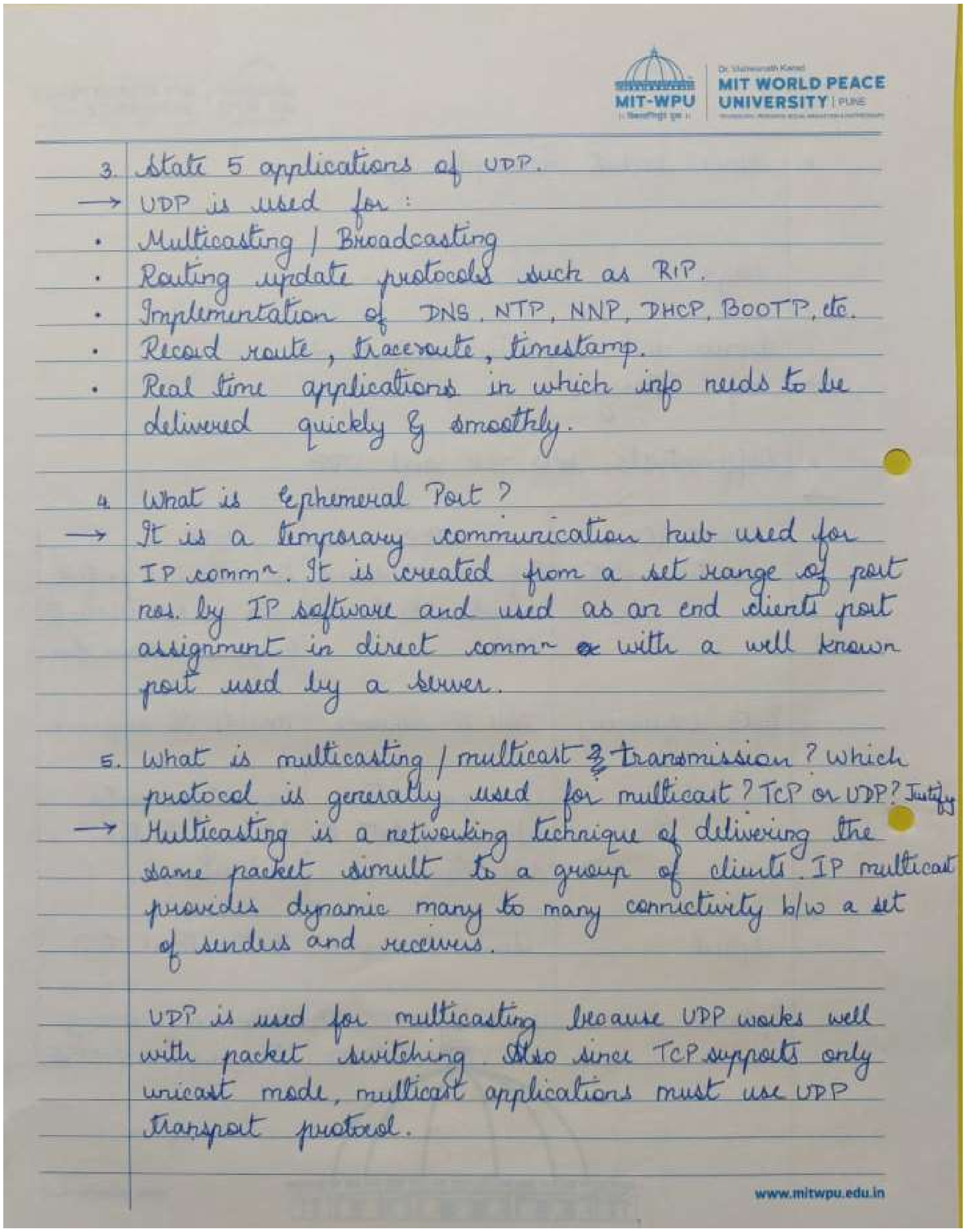
5. Close socket descriptor & exit.

FAQ's

1. Draw and explain UDP header.
→ Refer theory.

2. Differentiate b/w TCP and UDP
→

	FEATURES	TCP	UDP
·	connection status	Requires established connection to transmit data.	Connectionless protocol, with no requirement to open, maintain, close.
·	Data sequencing	Able to sequence	Unable to sequence
·	Reliability	can guarantee delivery of data to destination	cannot guarantee delivery to the destination.
·	Speed	Slower than UDP	Faster than TCP
·	Optimal use	Used by HTTP,HTTPS, SMTP, POP, FTP etc.	Video conferencing streaming, DNS, VoIP etc



3. State 5 applications of UDP.

→ UDP is used for :

- Multicasting / Broadcasting
- Routing update protocols such as RIP.
- Implementation of DNS, NTP, NNP, DHCP, BOOTP, etc.
- Record route, traceroute, timestamp.
- Real time applications in which info needs to be delivered quickly & smoothly.

4. What is Ephemeral Port ?

→ It is a temporary communication hub used for IP comm^n. It is created from a set range of port nos. by IP software and used as an end clients port assignment in direct comm^n with a well known port used by a server.

5. What is multicasting / multicast transmission ? Which protocol is generally used for multicast ? TCP or UDP ? Justify

→ Multicasting is a networking technique of delivering the same packet simult to a group of clients. IP multicast provides dynamic many to many connectivity b/w a set of senders and receivers.

UDP is used for multicasting because UDP works well with packet switching. Also since TCP supports only unicast mode, multicast applications must use UDP transport protocol.

Conclusion:

Thus, we studied and implemented a string conversion application using UDP socket programming.