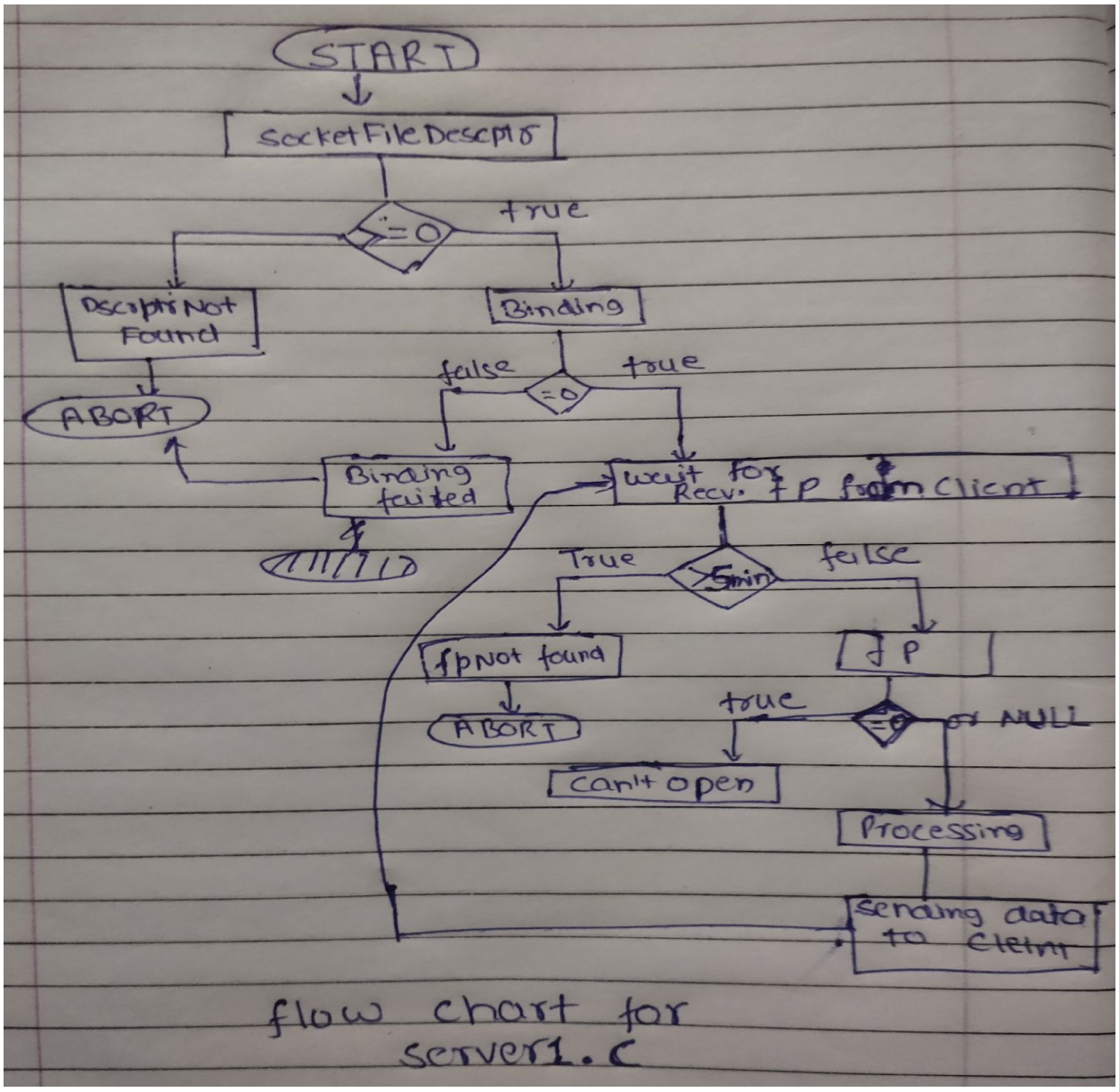
COMPUTER NETWORK
LAB ASSIGNMENT-2

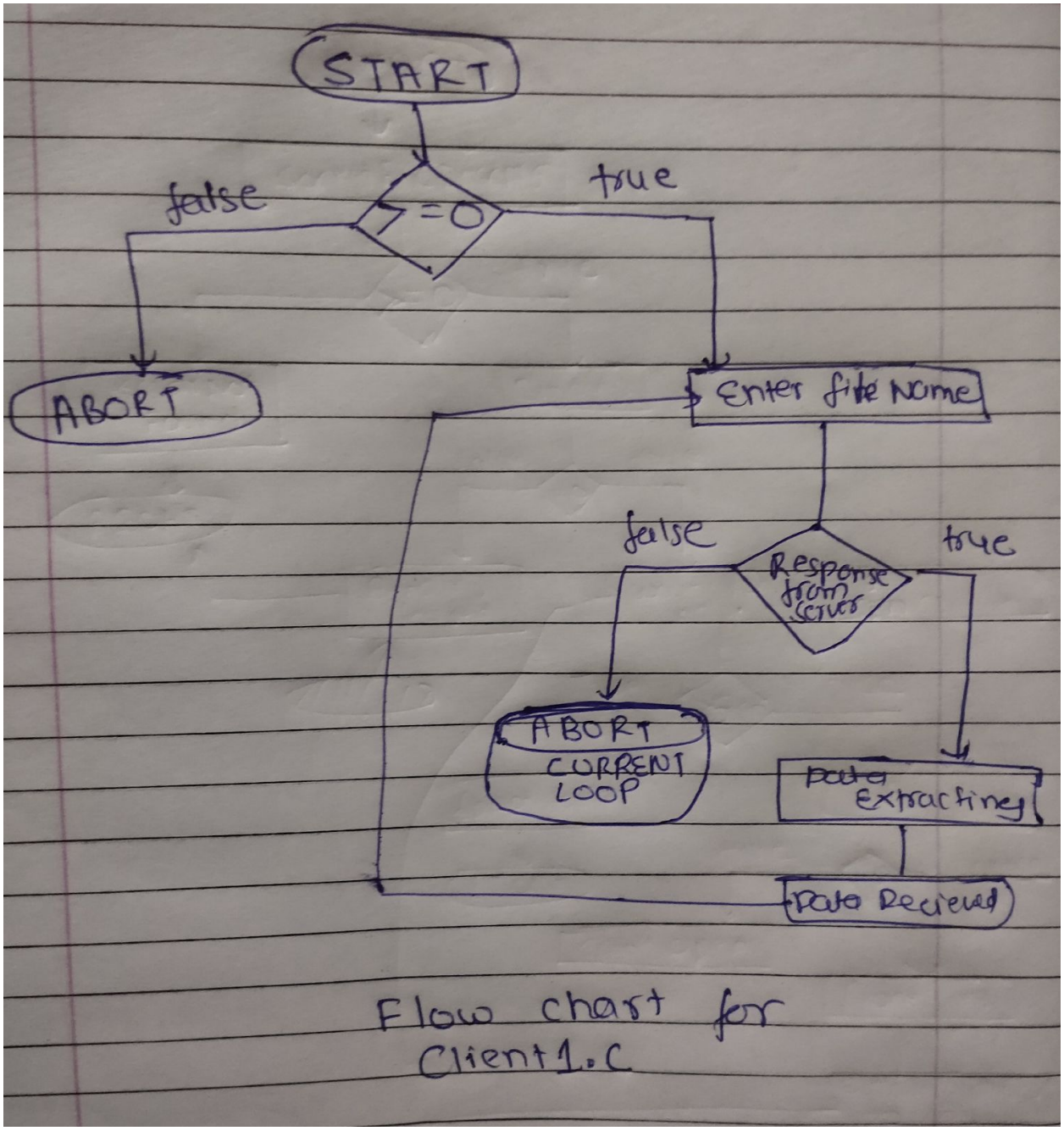
Submitted by (iit2018187) Abhishek Gupta

1. ***Write a client-server program that provides text and voice chat facility using datagram socket. Your application allows a user on one machine to talk to a user on another machine. Your application should provide a non blocking chat facility to the users. This means, the user can send its message at any time without waiting for a reply from the other side. (Hint: Use select() system call).***

Ans:

Flow chart:





EXPLAINING FUNCTION:

1.bufferClear

```
void bufferClear(char* b)
{
        int i;
        for (i = 0; i < 32; i++)
                b[i] = '\0';
}
```

It clears the buffer received from client or server for the next file.

2. **Cipher**

```
char Cipher(char ch)
{
        return ch ^ cipherKey;
}
```

It encrypted the data while sending to the client and on the other hand in client it used as decryption.

3. **sendFile**

It checks the file received from the client. If the null file pointer receives then it sends the message to the client " give me the correct name of file".

We use some helper or inbuilt function to create this chat application.

1. **socket(AF_INET, SOCK_DGRAM, IP_PROTOCOL);**
   It creates the socket by taking arguments like
   AF_INET : addresses from internet
   SOCK_DGRAM: type of socket it stands for datagram socket.
   IP_PROTOCOL: which protocol will be used in that network, we can use 0 which reference for the same.

2.bind(socket File Descriptor, (struct sockaddr*)&addr_con, sizeof(addr_con))

After creation of the socket, bind function binds the socket to the address and port number specified in addr(custom data structure). In the example code, we bind the server to the localhost, hence we use INADDR_ANY to specify the IP address.

It returns int value. 0 if binding is successful.
Mainly it binds the client to the network via their ip address and port.
socket File Descriptor: it represents the current descriptor of the socket.
Addr_con: it contains current ip_adress and port of connection.

3. recvfrom(socket File Descriptor, net_buf, 32, flagrc, (struct sockaddr*)&addr_con, length Of Address)

It returns an int value which represents the number of bytes received from the client.
net_buf: buffer which stores filename received from client.
32 : size of buffer
Flagrc: bool value which represents the flag value of receiving and sending permission.

EXPLAINING LIBRARIES USED:

Programme runs on the linux platform on the terminal with gcc installed.

Terminal must be in root mode.
We are only using gcc libraries to run this chat application.

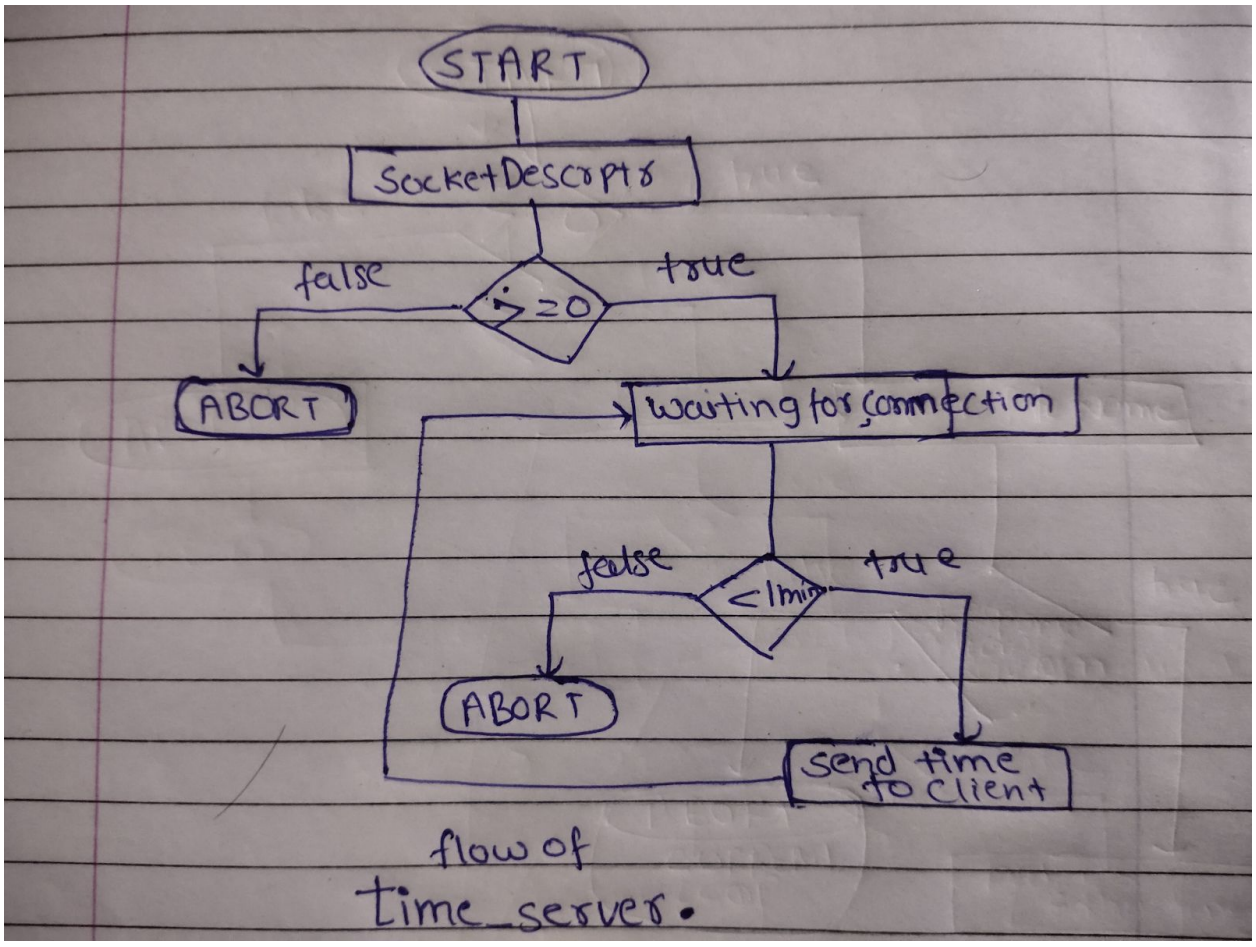
ALGORITHM:

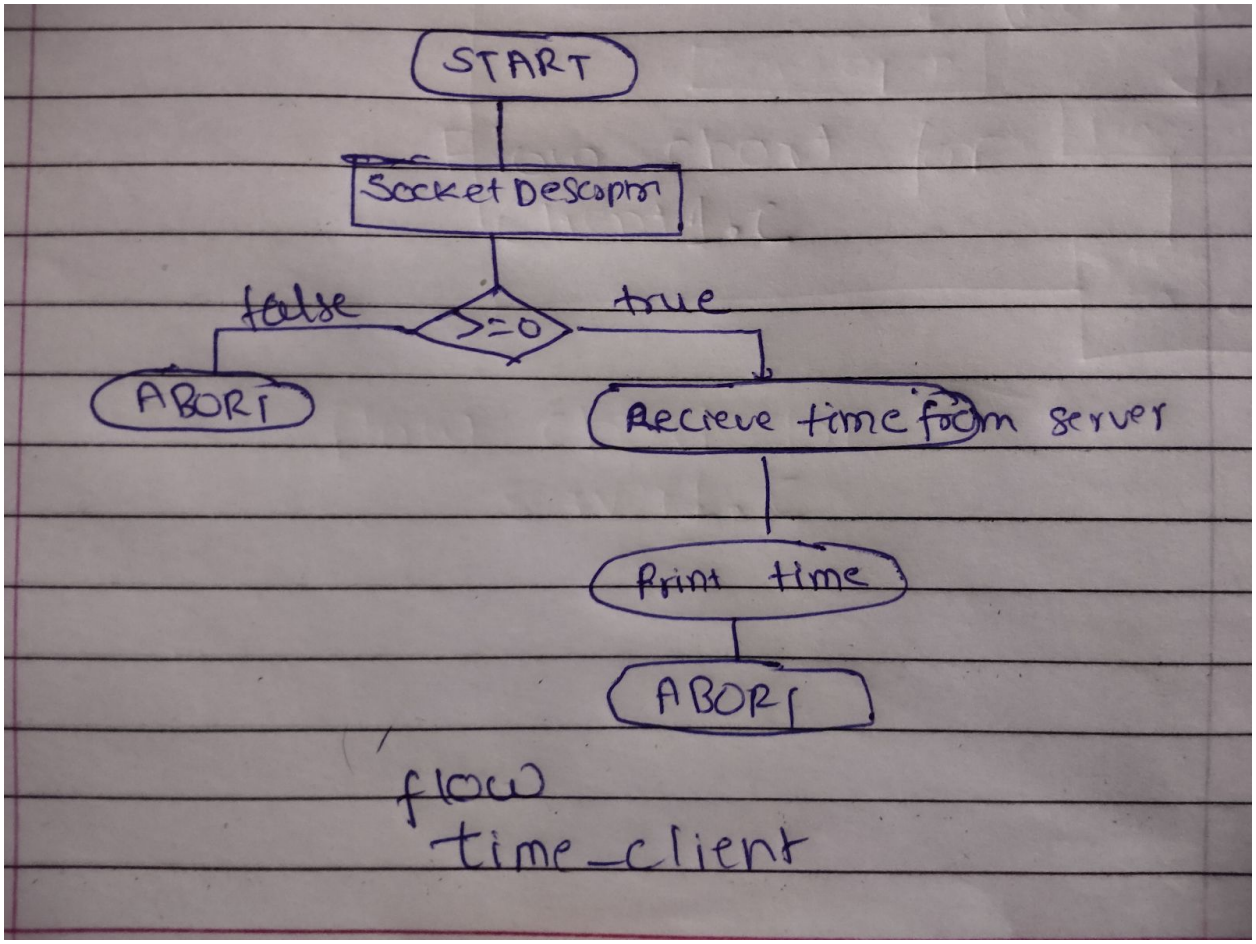
- We only use while loop in client and server program for continuation.
- For encrypting we xor value with char ‘S’ and for decrypting using the same.
- We send and receive data in a packet of 32 bytes.

2. ***Write a TCP client-server system to allow client programs to get the system date and time from the server. When a client connects to the server, the server gets the local time on the machine and sends it to the client. The client displays the date and time on the screen, and terminates. The server should be an iterative server.***

Ans:

Flowchart





EXPLAINING FUNCTION:

1. **listen(socketDescripter, 10);**

The listen() function marks a connection-mode socket.listen() attempts to continue to function rationally when there are no available descriptors. It accepts connections until the queue is emptied.

2. **accept(socketDescripter, (struct sockaddr*)NULL, NULL);**

The accept() function accepts a connection on a socket. An incoming connection is acknowledged and associated with an immediately created socket.

3. **ctime(&ticks)**

This function accepts single parameter *time_ptr*. It is used to set a time_t object that contains a time value.

EXPLAINING LIBRARIES USED:

Programme runs on the linux platform on the terminal with gcc installed.
Terminal must be in root mode.
We are only using gcc libraries to run this chat application.

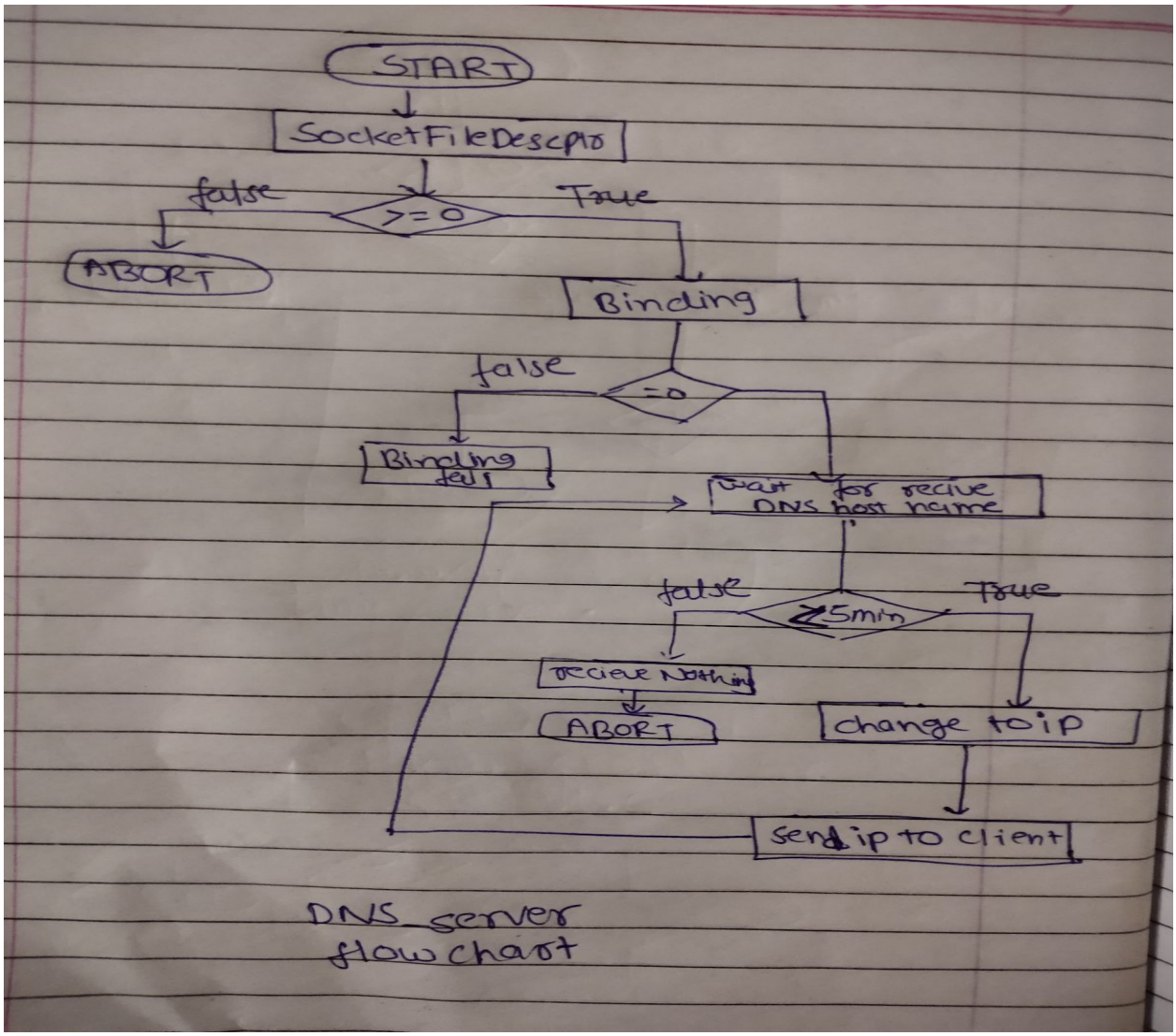
ALGORITHM:

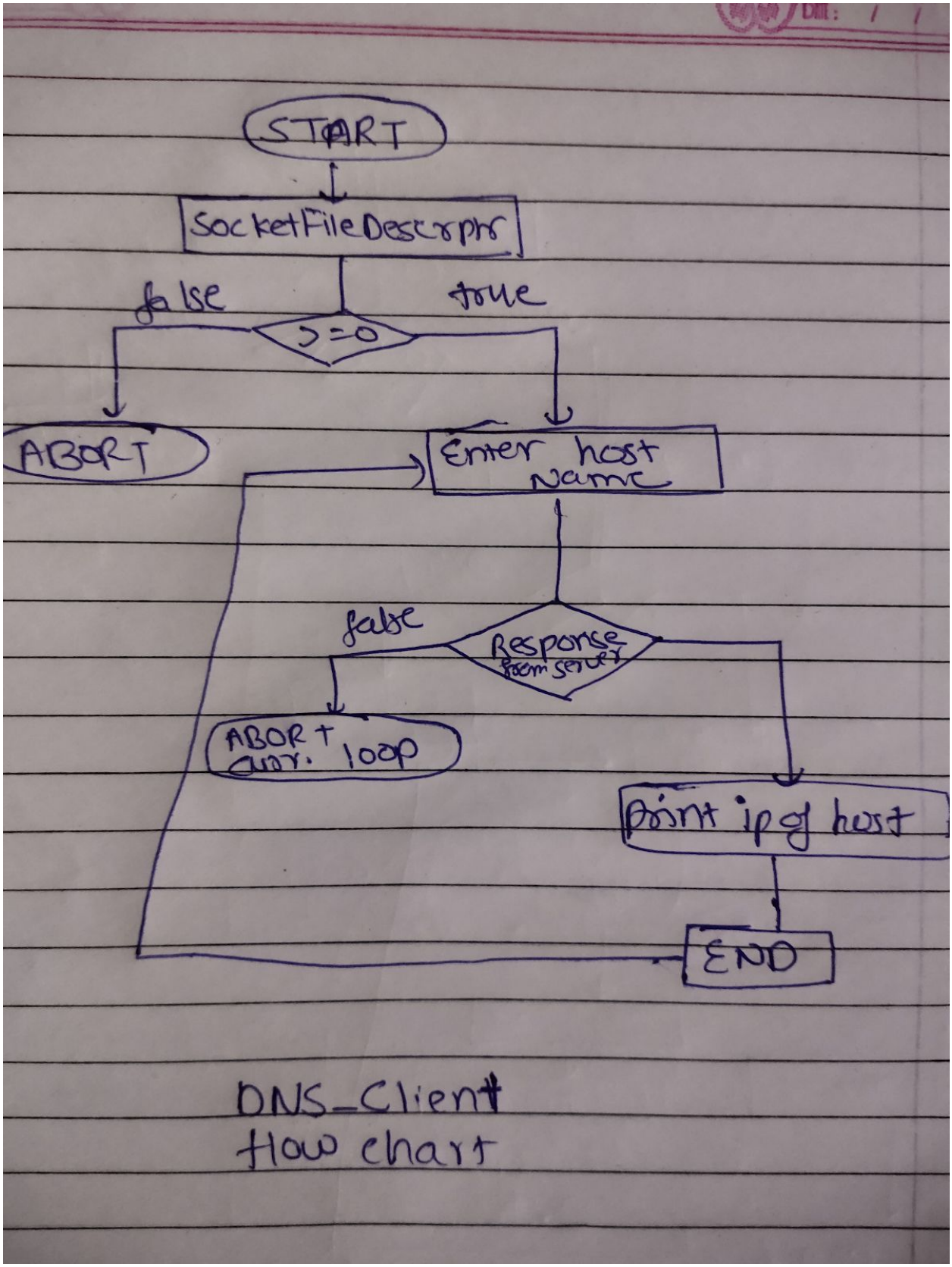
- We only use the while loop in server program for iterative server.
- ctime function for local time.

3.*Write a simple UDP iterative server and client to convert a given DNS name (for example, www.google.com) into its IP address(es). The client will read the DNS name as a string from the user and send it to the server. The server will convert it to one or more IP addresses and return it back to the client. The client will then print ALL the addresses returned, and exit. For basic UDP socket communication, see the sample program given. To get the IP address corresponding to a DNS name, use the function gethostbyname(). Look up the description of the function from the man page and the tutorial on the webpage.*

Ans:

Flow chart:





EXPLAINING FUNCTION:

I used one additional function host_to_ip in the server function to resolve hosts to their corresponding ip.

1.hostname_to_ip

```
int hostname_to_ip(char * hostname , char* ip)
{
	struct hostent *he;
	struct in_addr **addr_list;
	int i;

	if ( (he = gethostbyname( hostname ) ) == NULL)
	{
		herror("gethostbyname");
		return 1;
	}

	addr_list = (struct in_addr **) he->h_addr_list;

	for(i = 0; addr_list[i] != NULL; i++)
	{
		//Return the first one;
		strcpy(ip , inet_ntoa(*addr_list[i]) );
		//strcat(ip," \n");
		return 0;
	}

	return 1;
}
```

It uses **gethostbyname(hostname)** which returns hostent and their corresponding address lists.

EXPLAINING LIBRARIES USED:

Again no external libraries used.

I just use the linux platform with gcc libraries.

And one extra header file netdb.h which includes the function **gethostbyname .**

ALGORITHM:

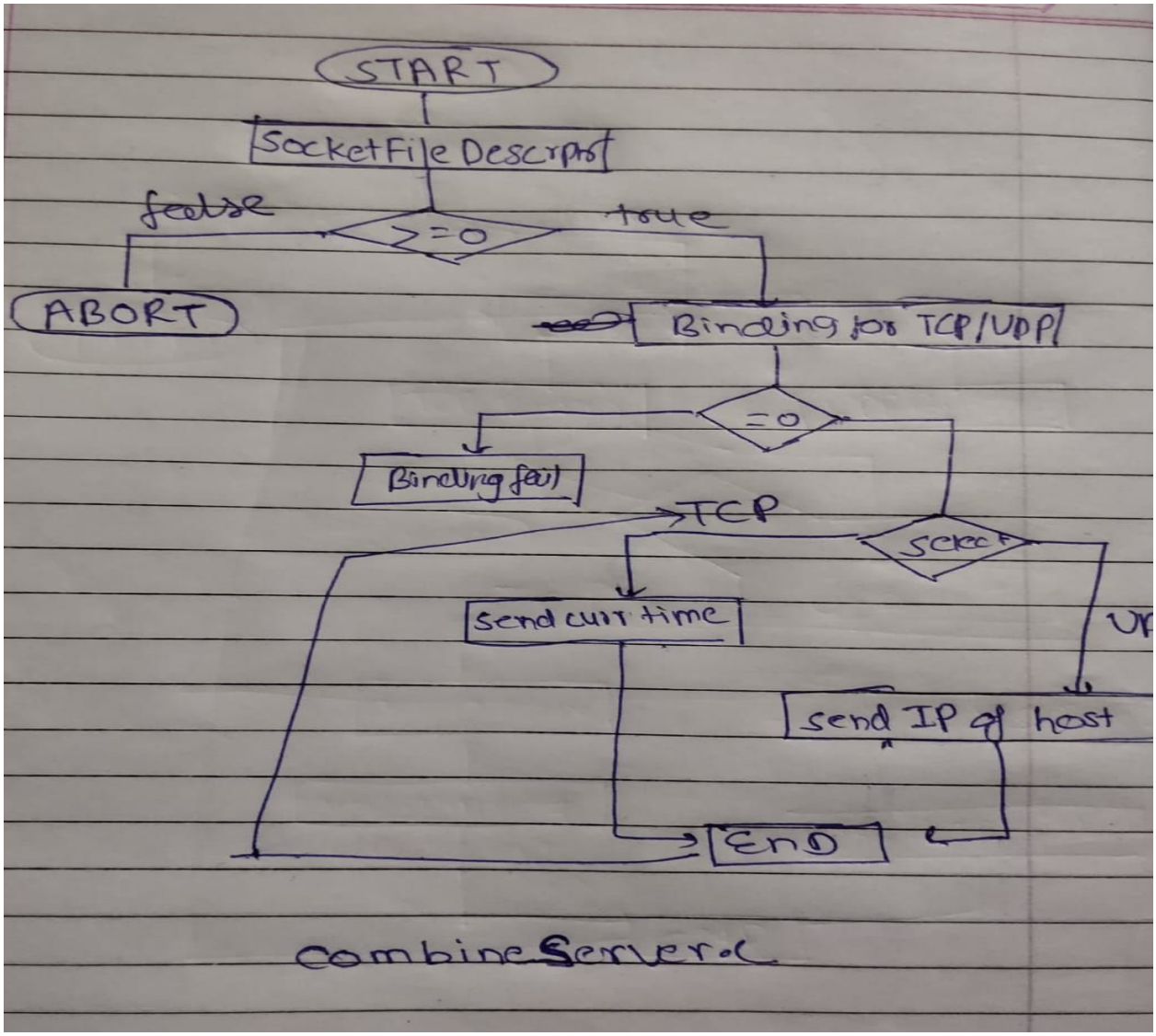
Algorithms are as simple as question 1. Instead of extracting file data we just print the ip address of the host.

4.Now suppose that the same server will act both as the time server in Problem 1 and the DNS server in Problem 2. Thus, some clients will request over the UDP socket for name-to-IP conversion, and some will connect over a TCP socket for the time. Thus, the server now needs to open both a TCP socket and a UDP socket, and accept request from any one (using the accept() + read()/send() call for TCP, and recvfrom() call for UDP), whichever comes first. Use the select() call to make the server wait for any one of the two connections, and handle whichever comes first. All handlings are iterative.

Submit one C file: combined_server.c

Ans:

Flow chart:



EXPLAINING FUNCTION:

1.select(maxfdp1, &rset, NULL, NULL, NULL);

select() and pselect() allow a program to monitor multiple file descriptors, waiting until one or more of the file descriptors become "ready" for some class of I/O operation .

Four macros are provided to manipulate the sets.

- FD_ZERO() clears a set.
- FD_SET() and
- FD_CLR() respectively add and remove a given file descriptor from a set.
- FD_ISSET() tests to see if a file descriptor is part of the set;

EXPLAINING LIBRARIES USED:

No additional libraries used in that question

ALGORITHM:

Flow chart explains most of the things.

I used while loop in program for iterative server

Select function used for selecting descriptor and waiting until more of the file descriptors become “ready”.

If udp is ready then the server receives the hostname and sends the ip address to udp client.

If tcp is ready then the server accepts connection and sends the current local time of the computer

**********END**********