Network Assignment 7

Name :- Debargha Mukherjee Roll :- 001910501067

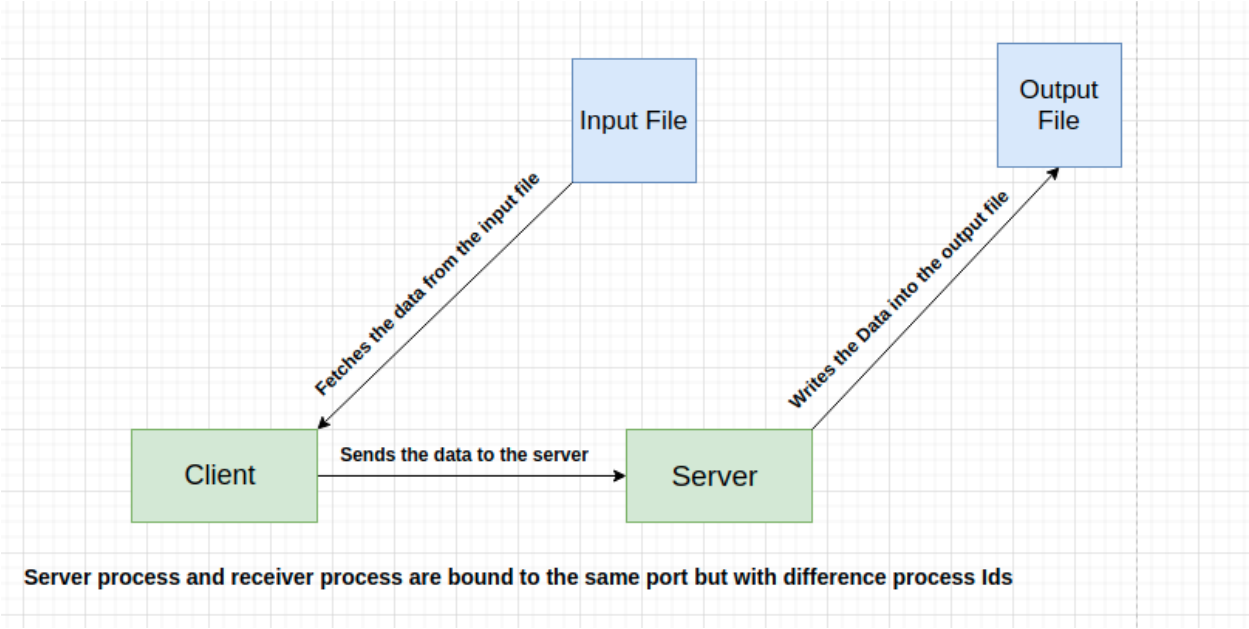
Problem Statement :- Implement any two protocols using TCP/UDP Socket as suitable.

1. BOOTP
2. FTP
3. DHCP
4. BGP
5. RIP

Solution

FTP design using TCP Sockets

System Architecture Design



Components of the system

Client :- client is responsible for reading the contents from the input file and sends the data to the server following the Transmission Control Protocol, i.e, by first establishing a connection with the server and then transmitting the contents of the file.The client

process mainly communicates with the server process via a single port which can be considered as the end point for process communication.

Server :- It runs on a specific port with a specific process id and is responsible for receiving the data from the client process. Finally it will write the contents received from the client process in the receive.txt file

FTP :- FTP stands for File transfer protocol. FTP is a standard internet protocol provided by TCP/IP used for transmitting the files from one host to another. It is mainly used for transferring the web page files from their creator to the computer that acts as a server for other computers on the internet .It is also used for downloading the files to computers from other servers.

Code Snippets to show the server and client process

server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#define SIZE 1024

void write_file(int sockfd){
 int n;
 FILE *fp;
 char *filename = "recv.txt";
 char buffer[SIZE];

 fp = fopen(filename, "w");
 while (1) {
   n = recv(sockfd, buffer, SIZE, 0);
   if (n <= 0){
     break;
     return;
   }
   fprintf(fp, "%s", buffer);
   bzero(buffer, SIZE);
 }
 return;
}
```

```c
int main(){
 char *ip = "127.0.0.1";
 int port = 8080;
 int e;

 int sockfd, new_sock;
 struct sockaddr_in server_addr, new_addr;
 socklen_t addr_size;
 char buffer[SIZE];

 sockfd = socket(AF_INET, SOCK_STREAM, 0);
 if(sockfd < 0) {
   perror("[-]Error in socket");
   exit(1);
 }
 printf("[+]Server socket created successfully.\n");

 server_addr.sin_family = AF_INET;
 server_addr.sin_port = port;
 server_addr.sin_addr.s_addr = inet_addr(ip);

 e = bind(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr));
 if(e < 0) {
   perror("[-]Error in bind");
   exit(1);
 }
 printf("[+]Binding successful.\n");

 if(listen(sockfd, 10) == 0){
   printf("[+]Listening....\n");
 }else{
   perror("[-]Error in listening");
   exit(1);
 }

 addr_size = sizeof(new_addr);
 new_sock = accept(sockfd, (struct sockaddr*)&new_addr, &addr_size);
 write_file(new_sock);
 printf("[+]Data written in the file successfully.\n");
```

```
 return 0;
}
```

client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#define SIZE 1024

void send_file(FILE *fp, int sockfd){
 int n;
 char data[SIZE] = {0};

 while(fgets(data, SIZE, fp) != NULL) {
   if (send(sockfd, data, sizeof(data), 0) == -1) {
     perror("[-]Error in sending file.");
     exit(1);
   }
   bzero(data, SIZE);
 }
}

int main(){
 char *ip = "127.0.0.1";
 int port = 8080;
 int e;

 int sockfd;
 struct sockaddr_in server_addr;
 FILE *fp;

 char *filename;
 printf ("Enter the File Name :- ");
 scanf ("%s", filename);
```

```c
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if(sockfd < 0) {
  perror("[-]Error in socket");
  exit(1);
}
printf("[+]Server socket created successfully.\n");

server_addr.sin_family = AF_INET;
server_addr.sin_port = port;
server_addr.sin_addr.s_addr = inet_addr(ip);

e = connect(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr));
if(e == -1) {
  perror("[-]Error in socket");
  exit(1);
}
printf("[+]Connected to Server.\n");

fp = fopen(filename, "r");
if (fp == NULL) {
  perror("[-]Error in reading file.");
  exit(1);
}

send_file(fp, sockfd);
printf("[+]File data sent successfully.\n");

printf("[+]Closing the connection.\n");
close(sockfd);

return 0;
}
```

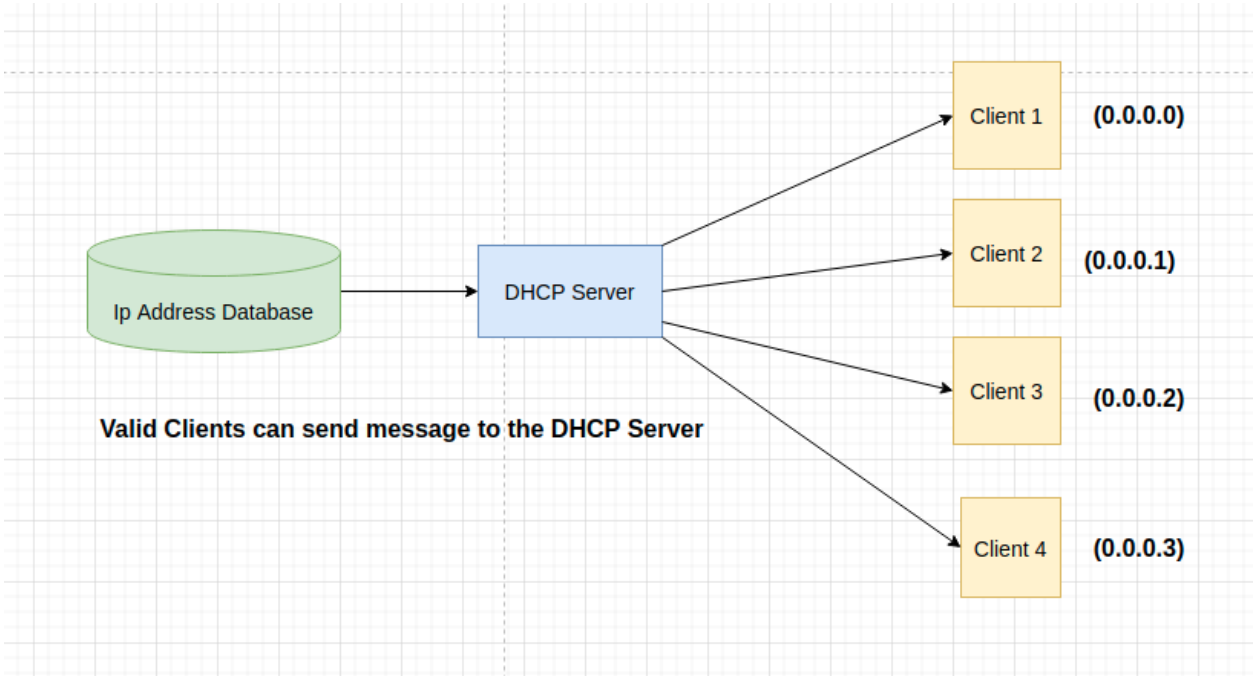
Output Snapshots for FTP

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
(base) debargha@debargha-Vostro-3480:~/BCSE_Network_Assignment/network_assignment7/FTP$ ./serve.c
bash: ./serve.c: No such file or directory
(base) debargha@debargha-Vostro-3480:~/BCSE_Network_Assignment/network_assignment7/FTP$ ./server
[+]Server socket created successfully.
[+]Binding successfull.
[+]Listening....
[+]Data written in the file successfully.
(base) debargha@debargha-Vostro-3480:~/BCSE_Network_Assignment/network_assignment7/FTP$
```

```
bash
(base) debargha@debargha-Vostro-3480:~/BCSE_Network_Assignment/network_assignment7/FTP$ ./cient
bash: ./cient: No such file or directory
(base) debargha@debargha-Vostro-3480:~/BCSE_Network_Assignment/network_assignment7/FTP$ ./client
Enter the File Name :- send.txt
[+]Server socket created successfully.
[+]Connected to Server.
[+]File data sent successfully.
[+]Closing the connection.
(base) debargha@debargha-Vostro-3480:~/BCSE_Network_Assignment/network_assignment7/FTP$
```

DHCP design using UDP

System Design Architecture



System Design Components

Client :- The client process will be created in a particular port with a specific process id and if it's within range of maximum number of clients that can be accomodated, it will be assigned with an ip address by the DHCP server from the ip address database.

Server :- The server process is responsible for allocating an ip address to all valid clients. If a client disconnects itself then it's ip will be freed and will be available to be assigned for another valid client.

Code Snippets

Server.py

```
import socket
import os



#AF_INET used for IPv4
#SOCK_DGRAM used for UDP protocol
s = socket.socket(socket.AF_INET , socket.SOCK_DGRAM )
#binding IP and port

print (os.getpid())

available_ip_queue = []
available_ip_queue.append('0.0.0.0')
available_ip_queue.append('0.0.0.1')
available_ip_queue.append('0.0.0.2')

connected_clients = []
s.bind(('127.0.0.1',12345))

print("Server started ...")
print("Waiting for Client response...")
#receiving data from client



#dictionary to handle all the clients
clients = {}
import errno
```

```
import os

def pid_exists(pid):
   """Check whether pid exists in the current process table.
   UNIX only.
   """
   if pid < 0:
       return False
   if pid == 0:
       # According to "man 2 kill" PID 0 refers to every process
       # in the process group of the calling process.
       # On certain systems 0 is a valid PID but we have no way
       # to know that in a portable fashion.
       raise ValueError('invalid PID 0')
   try:
       os.kill(pid, 0)
   except OSError as err:
       if err.errno == errno.ESRCH:
           # ESRCH == No such process
           return False
       elif err.errno == errno.EPERM:
           # EPERM clearly means there's a process to deny access to
           return True
       else:
           # According to "man 2 kill" possible error values are
           # (EINVAL, EPERM, ESRCH)
           raise
   else:
       return True

def find_pid(inode):

   # get a list of all files and directories in /proc
   procFiles = os.listdir("/proc/")

   # remove the pid of the current python process
   procFiles.remove(str(os.getpid()))

   # set up a list object to store valid pids
   pids = []
```

```
   for f in procFiles:
       try:
           # convert the filename to an integer and back, saving the
result to a list
           integer = int(f)
           pids.append(str(integer))
       except ValueError:
           # if the filename doesn't convert to an integer, it's not a
pid, and we don't care about it
           pass

   for pid in pids:
       # check the fd directory for socket information
       fds = os.listdir("/proc/%s/fd/" % pid)
       for fd in fds:
           # save the pid for sockets matching our inode
           if ('socket:[%d]' % inode) == os.readlink("/proc/%s/fd/%s" %
(pid, fd)):
               return pid
while True:
   data, addr = s.recvfrom(1024)
   dormant clients = []
   """
   for key, value in clients.items():
       if pid_exists(key[1]) == False:
           available_ip_queue.append(key[0])
           dormant_clients.append(key[1])
   print (dormant_clients)
   for client in dormant_clients:
       available_ip_queue.append(client)
       clients.pop(client, None)
   """
   if addr in clients.keys():
       s.sendto('acknowledgement msg'.encode(), addr)
       print (str (data.decode()) + ' Received from client with ip :- ' +
str(clients[addr]))
   else:
       msg = ''
       if len(available_ip_queue) == 0:
```

```
            msg = 'Sorry client cannot be provided with ip :- '
        else:
            msg = 'hello new client your assigned ip is :- ' +
available_ip_queue[0]
            clients[addr] = available_ip_queue[0]
            available_ip_queue.pop(0)
        s.sendto(msg.encode ('utf-8'), addr)
```

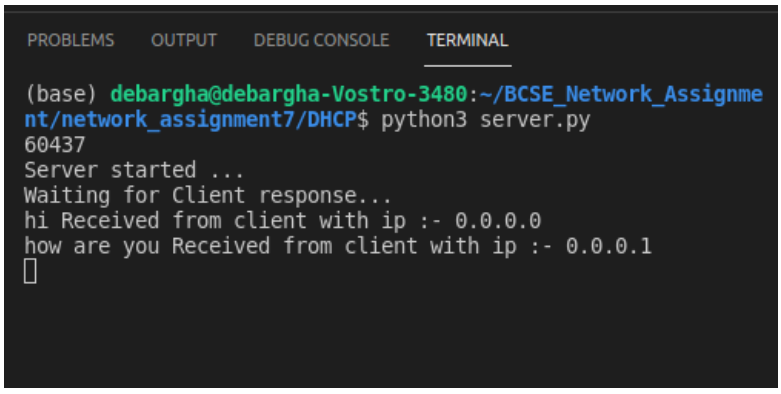
Client.py

```
import socket
import os
#client program

s = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
process_id = str(os.getpid())

count = 0
while True:
      #ip ,port = input("Enter server ip address and port number
:\n").split()
      m = input("Enter data to send server: ")
      res = s.sendto(m.encode('utf-8'), ('127.0.0.1',12345))
      data, addr = s.recvfrom(1024)
      if count == 0:
          print (str(data.decode()))
          count = 1
```

Output Snapshots



```
(base) debargha@debargha-Vostro-3480:~/BCSE_Network_Assignment/network_assignment7/DHCP$ python3 client.py
60455
Enter data to send server: h
hello new client your assigned ip is :- 0.0.0.0
60455
Enter data to send server: hi
60455
Enter data to send server:
```

```
python3
(base) debargha@debargha-Vostro-3480:~/BCSE_Network_Assignment/network_assignment7/DHCP$ python3 client.py
60522
Enter data to send server: hello
hello new client your assigned ip is :- 0.0.0.1
60522
Enter data to send server: how are you
60522
Enter data to send server:
```