

SUBMITTED BY:

ANUSHKA DIXIT [19BCE0577]

Tushar S Agrawal [19bce0559] Kaustubh Agrawal [19bce2274]

CSE 2005: Operating systems [j component]

Project Review

Server-Client File Transfer System using socket programming and text-based interface

# CONTENTS

# Introduction

# Approach

# Working (Graph)

# Algorithm

# Functions Used

# Code

# Output

# References

# INTRODUCTION

The file transfer system created is a server-client based model that allows server to access and process data stored on the client. When a user accesses a file on the client, the client sends the server a copy of the file, which is cached on the server’s computer while the data is being processed and is then returned to the server.

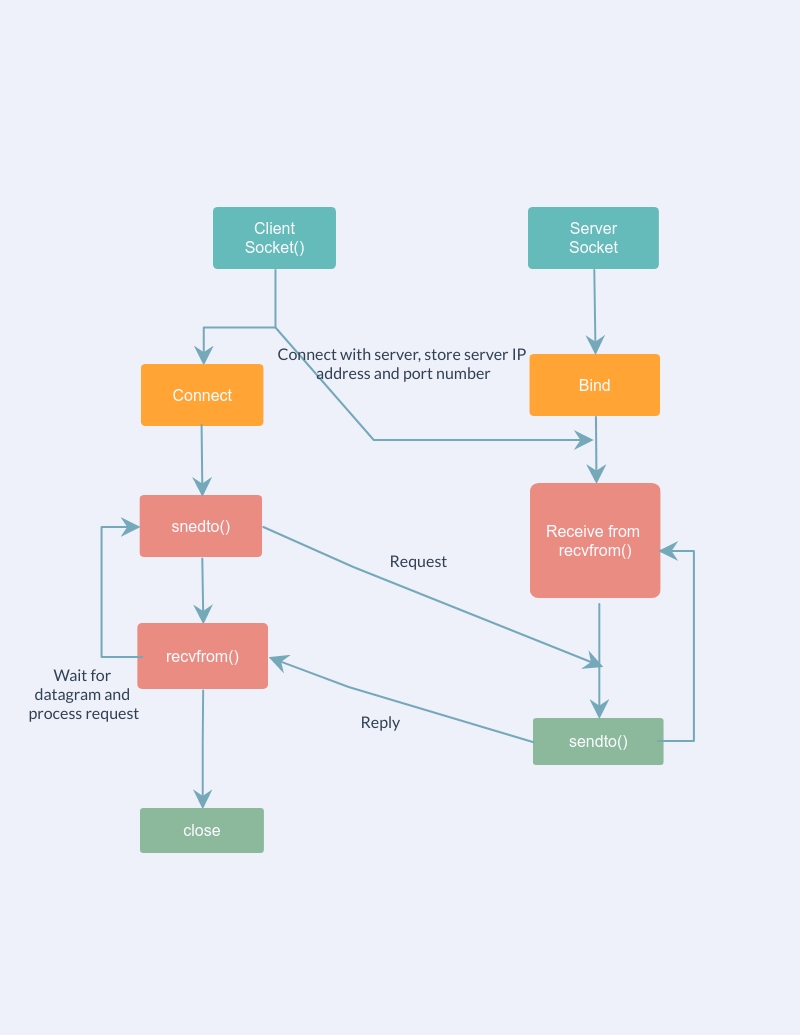
Server usually passively waits for and responds to clients. Hence it is the passive socket. The client on the other hand, initiates the communication and hence must know the address and the port of the server. It is the active socket. File transferring systems typically use file or database replication (distributing copies of data on multiple servers) to protect against data access failures. This is mainly due to the fact that UDP is not very reliable.

APPROACH

Here, the model uses one client and server implemented using the UDP protocol. The User Datagram Protocol, or UDP, is a communication protocol used across the Internet for especially time-sensitive transmissions such as video playback or [DNS](https://www.cloudflare.com/learning/dns/what-is-dns/) lookups. It speeds up communications by not formally establishing a connection before data is transferred. This allows data to be transferred very quickly.

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 sender which the server uses to send data to the correct client. For establishing connection between the two ends, socket is used. Sockets allow communication between two different processes on the same or different machines and is used in the UNIX client-server application framework.

# WORKING

The working of the application can be explained in simple steps using the following graph:

When both client and server side is available on the same computer, connect() function is used. Bind function is used to bind the server while sendto() and recvfrom() functions are used to send and receive files. After the end of the program, the socket is closed.

# ALGORITHM

1. The server starts and waits for file name.
2. The client sends a file name.
3. The server receives file name.
4. Server checks for the file.
5. If the file is present server starts reading file and continues encrypted until file end is reached.
6. End is marked by EOF (End of File).
7. File is received as buffers until EOF is received. Then it is decrypted.
8. If not present, a file not found error is sent.

# FUCNTIONS USED

The necessary functions for this application are:

1. socket()

Syntax: int socket(int domain, int type, int protocol)

socket() creates an endpoint for communication and returns a file descriptor that refers to that endpoint. The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.

**Arguments :**  
domain – Specifies the communication domain   
type – Type of socket to be created  
protocol – Protocol to be used by socket.

1. bind()

Syntax: int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen)

Assigns address to the unbound socket.

**Arguments :**  
sockfd – File descriptor of socket to be binded  
addr – Structure in which address to be binded to is specified  
addrlen – Size of addr structure

1. sendto()

Syntax: ssize\_t sendto(int sockfd, const void \*buf, size\_t len, int flags, const struct sockaddr \*dest\_addr, socklen\_t addrlen)

Sends a message on the socket

**Arguments :**  
sockfd – File descriptor of socket  
buf – Application buffer containing the data to be sent  
len – Size of buf application buffer  
flags – Bitwise OR of flags to modify socket behaviour  
dest\_addr – Structure containing address of destination  
addrlen – Size of dest\_addr structure

1. recvfrom()

Synatx: ssize\_t recvfrom(int sockfd, void \*buf, size\_t len, int flag, struct sockaddr \*src\_addr, socklen\_t \*addrlen)

Receives a message from the socket.

**Arguments :**  
sockfd – File descriptor of socket  
buf – Application buffer in which to receive data  
len – Size of buf application buffer  
flags – Bitwise OR of flags to modify socket behaviour  
src\_addr – Structure containing source address is returned  
addrlen – Variable in which size of src\_addr structure is returned

1. close

Synatx: int close(int fd)

Closes a file descriptor

**Arguments :** fd – File descriptor

1. connect():

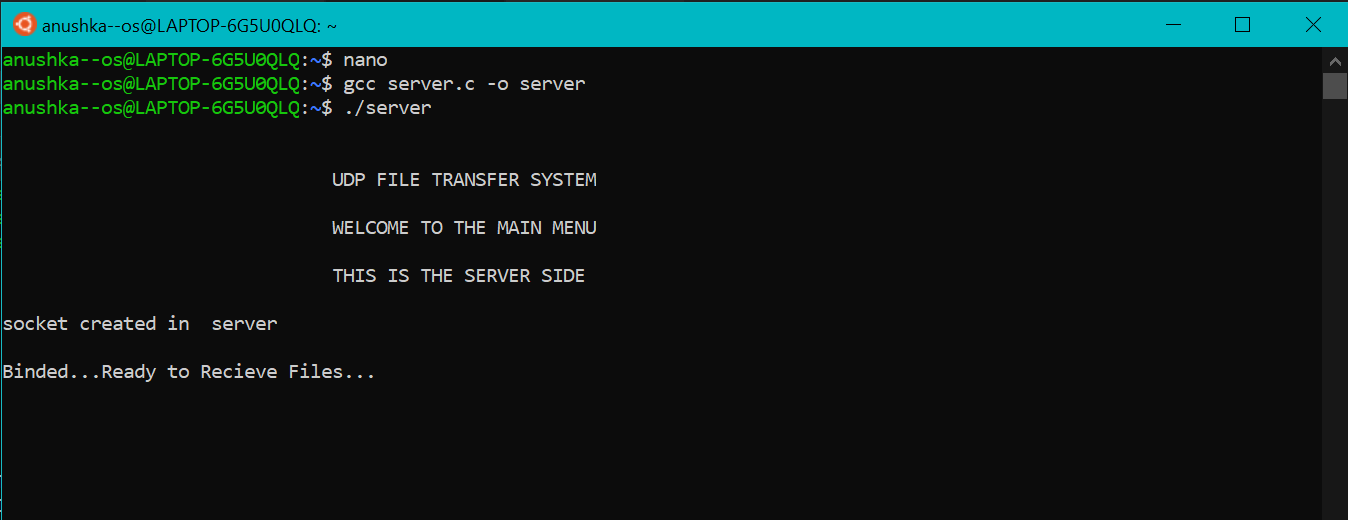
Sytanx: int connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

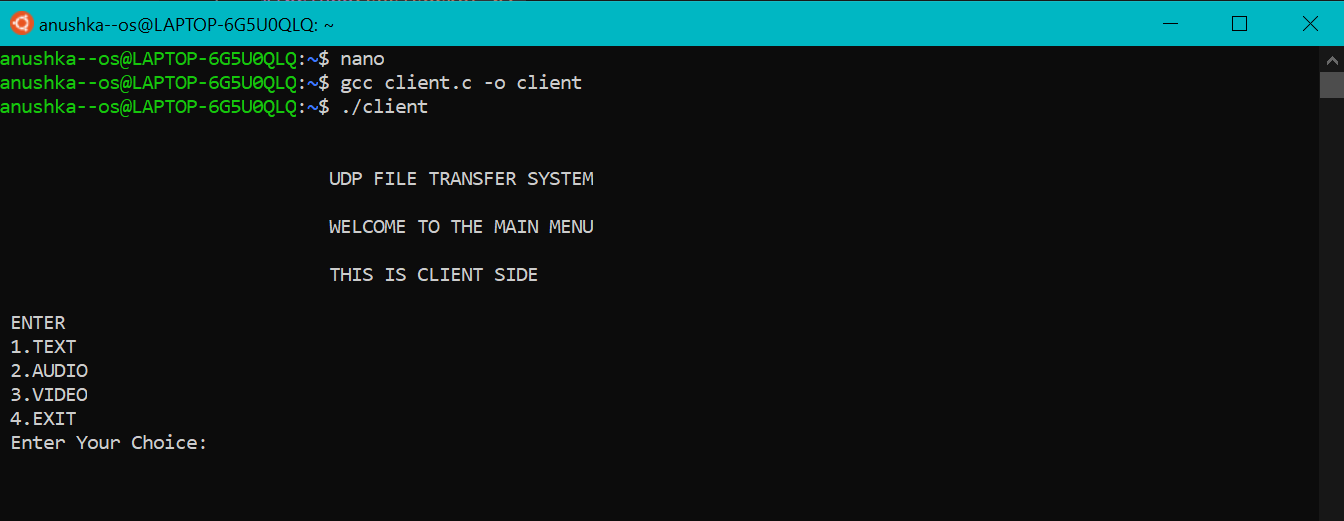
The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server’s address and port is specified in addr.

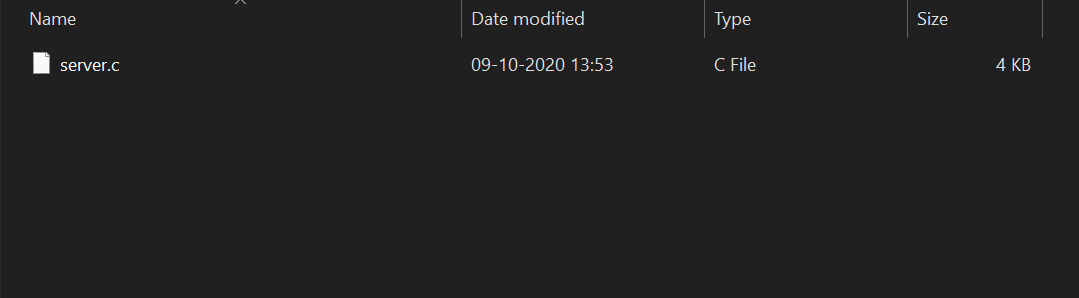
# CODE

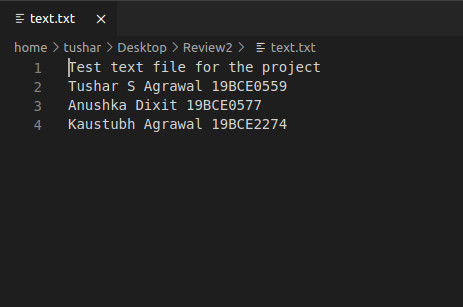
1. **CLIENT SIDE: client.c**
2. #include <stdio.h>
3. #include <stdlib.h>
4. #include <unistd.h>
5. #include <string.h>
6. #include <sys/types.h>
7. #include <sys/socket.h>
8. #include <arpa/inet.h>
9. #include <netinet/in.h>
10. #define PORT 8000
11. #define remote\_port 8001
12. int main()
13. {
14. int fd;
15. char datagram[512], fileName[100], afileName[100], vfileName[100], file\_buffer[100], c, caufile[7000], aufile[7000000], vfile[1000000];
16. char Bye[4] = "Bye";
17. struct sockaddr\_in servaddr;
18. struct sockaddr\_in remote\_add;
19. int remote\_len, z, x, remote\_sock;
20. // Creating socket
21. if ((fd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0)
22. {
23. perror("Error!! Socket was not Created....Exiting");
24. exit(EXIT\_FAILURE);
25. }
26. printf("\n\n\t\t\t      UDP FILE TRANSFER SYSTEM");
27. printf("\n\n\t\t\t      WELCOME TO THE MAIN MENU");
28. printf("\n\n\t\t\t      THIS IS CLIENT SIDE");
29. memset(&servaddr, 0, sizeof(servaddr));
30. bzero(&servaddr, sizeof(servaddr));
31. servaddr.sin\_family = AF\_INET;
32. servaddr.sin\_port = htons(PORT);
33. servaddr.sin\_addr.s\_addr = INADDR\_ANY;
34. int len = sizeof(servaddr);
35. int choice;
36. Loop:
37. while (choice != 5)
38. {
39. printf("\n\n ENTER \n 1.TEXT \n 2.AUDIO \n 3.VIDEO\n 4.Chat\n 5.EXIT\n Enter Your Choice: ");
40. scanf("%d", &choice);
41. char num = choice;
42. sendto(fd, &num, sizeof(num), 0, (struct sockaddr \*)&servaddr, sizeof(struct sockaddr));
43. switch (choice)
44. {
45. case 1:
46. printf(" Enter text file name to send: \n");
47. scanf("%s", fileName);
48. sendto(fd, fileName, strlen(fileName), 0, (struct sockaddr \*)&servaddr, sizeof(struct sockaddr));
49. FILE \*fp;
50. fp = fopen(fileName, "r");
51. if (fp)
52. {
53. printf(" Reading file contents...\n");
54. fseek(fp, 0, SEEK\_END);
55. size\_t file\_size = ftell(fp);
56. fseek(fp, 0, SEEK\_SET);
57. if (fread(file\_buffer, file\_size, 1, fp) <= 0)
58. {
59. printf(" Error!! Empty File or Error reading the File...\n");
60. exit(1);
61. }
62. }
63. else
64. {
65. printf(" Error!! Cannot open the file\n");
66. exit(0);
67. }
68. printf(" FILE CONTENTS to Send :\n%s\n", file\_buffer);
69. if (sendto(fd, file\_buffer, strlen(file\_buffer), 0, (struct sockaddr \*)&servaddr, sizeof(struct sockaddr)) < 0)
70. {
71. printf("Error!! File wad not Sent\n");
72. }
73. else
74. {
75. printf("File Sent Sucessfully\n");
76. z = recvfrom(fd, datagram, 512, 0, (struct sockaddr \*)&servaddr, &len);
77. printf("Acknowledgement from server: %s", datagram);
78. }
79. fclose(fp);
80. break;
81. case 2:
82. printf("Enter audio file name to send : \n");
83. scanf("%s", afileName);
84. sendto(fd, afileName, strlen(afileName), 0, (struct sockaddr \*)&servaddr, sizeof(struct sockaddr));
85. FILE \*afp;
86. afp = fopen(afileName, "r");
87. fseek(afp, 0, SEEK\_END);
88. size\_t afsize = ftell(afp);
89. fseek(afp, 0, SEEK\_SET);
90. if (afp)
91. {
92. printf("Reading file contents.\n");
93. if (fread(aufile, afsize, 1, afp) <= 0)
94. {
95. printf("Error!! Empty file or Error reading the File\n");
96. exit(1);
97. }
98. }
99. else
100. {
101. printf("Could not read audio file.\n");
102. exit(0);
103. }
104. if (sendto(fd, aufile, afsize, 0, (struct sockaddr \*)&servaddr, sizeof(struct sockaddr)) < 0)
105. {
106. printf("Error!! Unable to send the file\n");
107. }
108. else
109. {
110. printf(" File Sent Sucessfully\n");
111. z = recvfrom(fd, datagram, 512, 0, (struct sockaddr \*)&servaddr, &len);
112. printf("Acknowledgement from server: %s", datagram);
113. }
114. fclose(afp);
115. break;
116. case 3:
117. printf("Enter video file name to send : \n");
118. scanf("%s", vfileName);
119. sendto(fd, vfileName, strlen(vfileName), 0, (struct sockaddr \*)&servaddr, sizeof(struct sockaddr));
120. FILE \*vfp;
121. vfp = fopen(vfileName, "r");
122. fseek(vfp, 0, SEEK\_END);
123. size\_t vfsize = ftell(vfp);
124. fseek(vfp, 0, SEEK\_SET);
125. if (vfp)
126. {
127. if (fread(vfile, 1, vfsize, vfp) <= 0)
128. printf("Error!! Empty file or Error reading the File\n");
129. }
130. else
131. {
132. printf("Could not read audio file.\n");
133. exit(0);
134. }
135. if (sendto(fd, vfile, vfsize, 0, (struct sockaddr \*)&servaddr, sizeof(struct sockaddr)) < 0)
136. printf("Error!! File was not sent\n");
137. else
138. {
139. printf("File Sent Sucessfully \n");
140. z = recvfrom(fd, datagram, 512, 0, (struct sockaddr \*)&servaddr, &len);
141. printf("Acknowledgement from server: %s", datagram);
142. }
143. fclose(vfp);
144. break;
145. case 5:
146. close(fd);
147. break;
148. case 4:
149. printf("Client Initiated Chat ...\nType 'Bye' to exit from chat");
150. fflush(stdin);
151. fgets(datagram, 512, stdin);
152. while (1)
153. {
154. printf("\nYou(Send A message): ");
155. bzero(datagram, 512);
156. fgets(datagram, 512, stdin);
157. x = sendto(fd, datagram, strlen(datagram), 0, (struct sockaddr \*)&servaddr, sizeof(struct sockaddr));
158. if (x != -1)
159. {
160. printf("Message sent. ");
161. }
162. if (strcmp(datagram, Bye) == 0)
163. {
164. printf("Closing chat..\n");
165. break;
166. }
167. printf("Waiting for response.........\n");
168. bzero(datagram, 512);
169. // z = recvfrom(fd, datagram, 512, 0, (struct sockaddr \*)&remote\_add, &remote\_len);
170. z = recvfrom(fd, datagram, 512, 0, (struct sockaddr \*)&servaddr, &len);
171. datagram[z] = 0;
172. printf("\nServer: %s", datagram);
173. if (strcmp(datagram, Bye) == 0)
174. {
175. printf("Closing chat..\n");
176. break;
177. }
178. }
179. break;
180. default:
181. printf("Wrong Input... Try Again...\n\n");
182. break;
183. }
184. }
185. }
186. **SERVER SIDE: server.c**
187. #include <sys/socket.h>
188. #include <arpa/inet.h>
189. #include <stdio.h>
190. #include <unistd.h>
191. #include <fcntl.h>
192. #include <sys/types.h>
193. #include <string.h>
194. #include <stdlib.h>
195. #define PORT 8000
196. #define maxlen 70000
197. #define mlen 100000
198. #define remote\_port 8001
199. int main()
200. {
201. printf("\n\n\t\t\t      UDP FILE TRANSFER SYSTEM");
202. printf("\n\n\t\t\t      WELCOME TO THE MAIN MENU");
203. printf("\n\n\t\t\t      THIS IS THE SERVER SIDE");
204. char fileName[100];
205. char filebuffer[2000], caufile[maxlen];
206. char \*vfilep;
207. int aufile[700000], vfile[mlen];
208. int sd, connfd, len, remote\_sock, x, z, remote\_len;
209. char datagram[512];
210. char Bye[4] = "Bye";
211. for (int i = 0; i <= 100; i++)
212. {
213. fileName[i] = '\0';
214. }
215. struct sockaddr\_in servaddr, cliaddr;
216. struct sockaddr\_in remote\_add;
217. sd = socket(AF\_INET, SOCK\_DGRAM, 0);
218. if (sd == -1)
219. {
220. printf("\n\n socket not created in server\n");
221. exit(0);
222. }
223. else
224. printf("\n\nsocket created in  server\n");
225. bzero(&servaddr, sizeof(servaddr));
226. servaddr.sin\_family = AF\_INET;
227. servaddr.sin\_addr.s\_addr = INADDR\_ANY;
228. servaddr.sin\_port = htons(PORT);
229. memset(&(servaddr.sin\_zero), '\0', 8);
230. if (bind(sd, (struct sockaddr \*)&servaddr, sizeof(servaddr)) != 0)
231. printf(" \nNot binded\n");
232. else
233. printf(" \nBinded...Ready to Recieve Files...\n");
234. len = sizeof(cliaddr);
235. int choice = 1;
236. Recreated:
237. while (1)
238. {
239. char num;
240. recvfrom(sd, &num, sizeof(num), 0, (struct sockaddr \*)&cliaddr, &len);
241. choice = num;
242. switch (choice)
243. {
244. case 1:
245. recvfrom(sd, fileName, 1024, 0, (struct sockaddr \*)&cliaddr, &len);
246. printf(" NAME OF TEXT FILE RECEIVED: %s\n", fileName);
247. FILE \*fp;
248. printf(" Contents in the RECIEVED TEXT file: \n");
249. recvfrom(sd, filebuffer, 1024, 0, (struct sockaddr \*)&cliaddr, &len);
250. printf("%s\n", filebuffer);
251. int fsize = strlen(filebuffer);
252. fp = fopen(fileName, "w");
253. if (fp)
254. {
255. fwrite(filebuffer, fsize, 1, fp);
256. printf(" File received successfully\n");
257. strcpy(datagram, "Got Your Message\n");
258. x = sendto(sd, datagram, strlen(datagram), 0, (struct sockaddr \*)&cliaddr, sizeof(struct sockaddr));
259. if (x != -1)
260. {
261. puts("Message sent...\n");
262. }
263. }
264. else
265. printf(" Error!! Unable to Create Output File\n");
266. memset(fileName, '\0', sizeof(fileName));
267. fclose(fp);
268. break;
269. case 2:
270. recvfrom(sd, fileName, 1024, 0, (struct sockaddr \*)&cliaddr, &len);
271. printf(" NAME OF AUDIO FILE RECEIVED: %s\n", fileName);
272. FILE \*afp;
273. int numbytes;
274. afp = fopen(fileName, "w");
275. size\_t afsize;
276. afsize = recvfrom(sd, aufile, 700000, 0, (struct sockaddr \*)&cliaddr, &len);
277. if (afp)
278. {
279. fwrite(aufile, afsize, 1, afp);
280. printf(" File received successfully\n");
281. strcpy(datagram, "Got Your Audio File\n");
282. x = sendto(sd, datagram, strlen(datagram), 0, (struct sockaddr \*)&cliaddr, sizeof(struct sockaddr));
283. if (x != -1)
284. {
285. puts("Message sent...\n");
286. }
287. }
288. else
289. printf(" Error!! Unable to Open Output File\n");
290. memset(fileName, '\0', sizeof(fileName));
291. fclose(afp);
292. break;
293. case 3:
294. recvfrom(sd, fileName, 1024, 0, (struct sockaddr \*)&cliaddr, &len);
295. printf(" VIDEO FILE NAME RECEIVED: %s\n", fileName);
296. FILE \*vfp;
297. vfp = fopen(fileName, "w");
298. size\_t vfsize;
299. vfsize = recvfrom(sd, vfile, 100000, 0, (struct sockaddr \*)&cliaddr, &len);
300. if (vfp)
301. {
302. fwrite(vfile, vfsize, 1, vfp);
303. printf(" File received successfully\n");
304. strcpy(datagram, "Got Your Video File\n");
305. x = sendto(sd, datagram, strlen(datagram), 0, (struct sockaddr \*)&cliaddr, sizeof(struct sockaddr));
306. if (x != -1)
307. {
308. puts("Message sent...\n");
309. }
310. }
311. else
312. printf(" Error!! Unable to Open Output File\n");
313. fclose(vfp);
314. break;
315. case 4:
316. while (1)
317. {
318. bzero(datagram, 512);
319. /\* Here we are waiting for recieving a message from remote machine \*/
320. printf("Waiting for response.........\n");
321. z = recvfrom(sd, datagram, 512, 0, (struct sockaddr \*)&cliaddr, &len);
322. datagram[z] = 0;
323. if (strcmp(datagram, Bye) == 0)
324. {
325. printf("Client: Bye\nClosing chat...\n");
326. printf("Bye\n");
327. goto Recreated;
328. }
329. printf("Client: %s", datagram);
330. printf("\nYou(Send A message): ");
331. bzero(datagram, 512);
332. fgets(datagram, 512, stdin);
333. x = sendto(sd, datagram, strlen(datagram), 0, (struct sockaddr \*)&cliaddr, sizeof(struct sockaddr));
334. if (x != -1)
335. {
336. puts("Message sent...\n");
337. }
338. if (strcmp(datagram, Bye) == 0)
339. {
340. printf("Closing Chat....\n");
341. goto Recreated;
342. }
343. }
344. case 5:
345. close(sd);
346. break;
347. }
348. }
349. return (0);
350. }

# OUTPUT

**Server start**

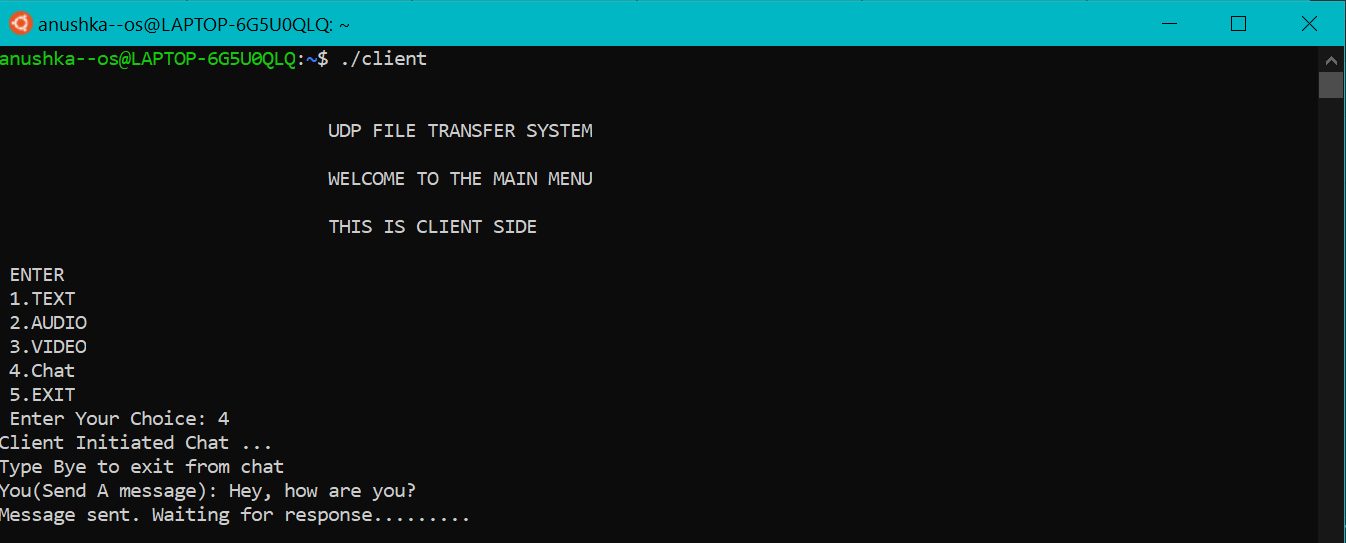
**Client starts**

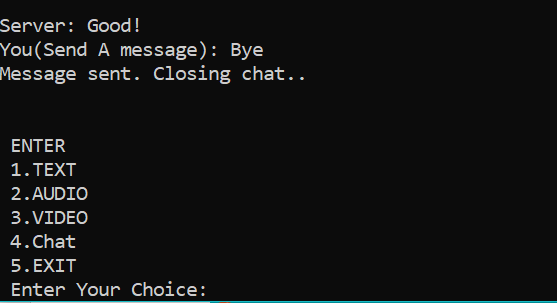
****

**Text File**

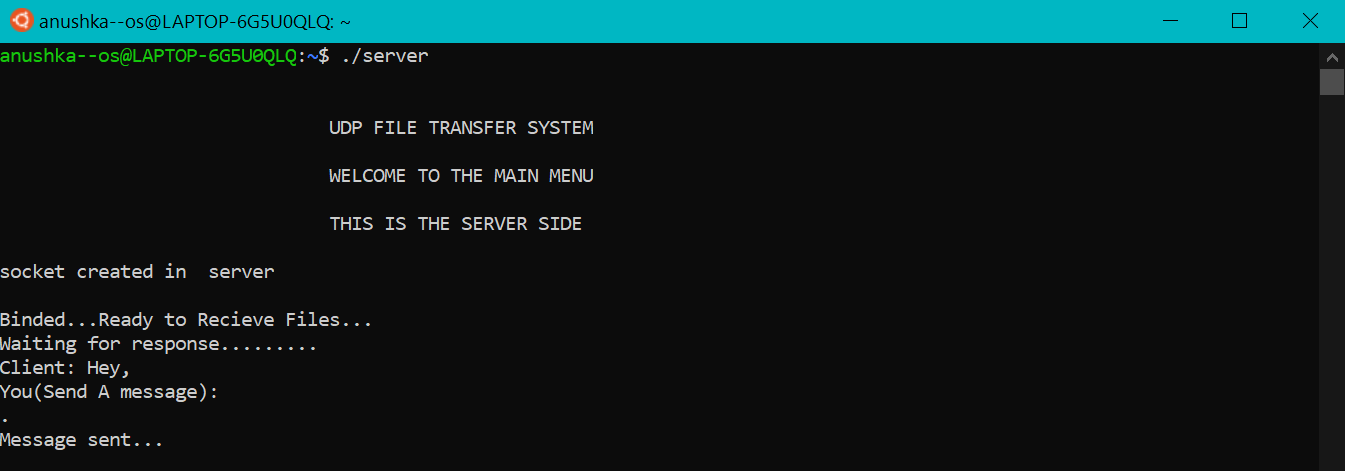
# Chat Application

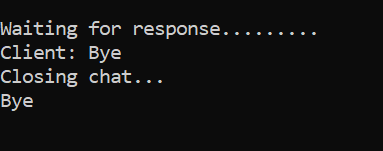
**Client end’s output**





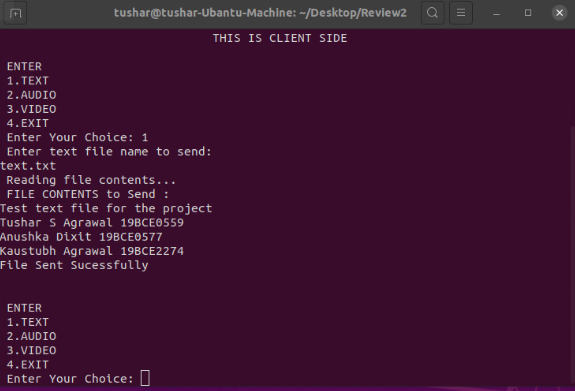
**Server end’s output**

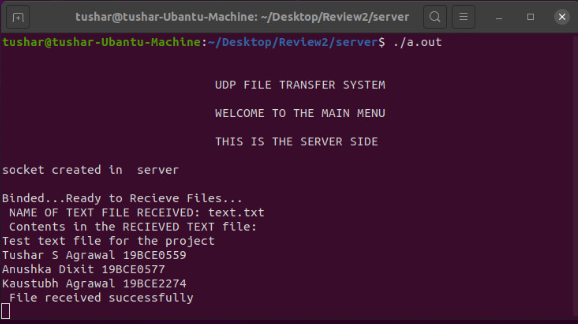




# 2. Sending Text Files

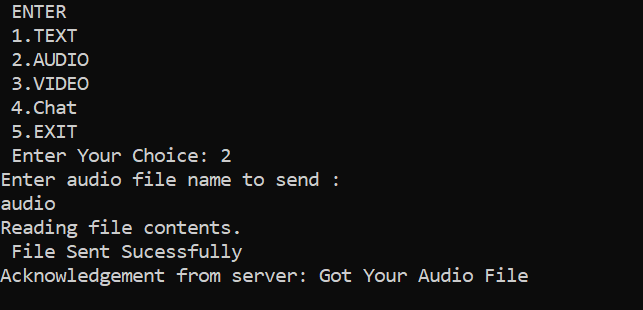
**Client end’s output**



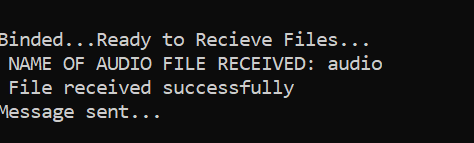
**Server end’s output**

# Sending Audio Files

**Client Side’s Output**

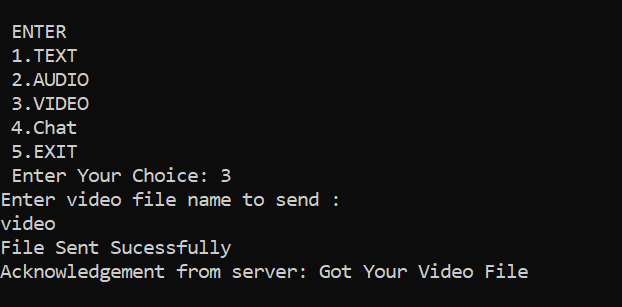


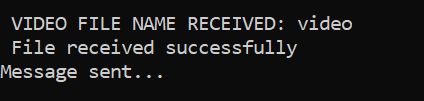
**Server Side’s Output**



# Sending Video Files

**Client side’s Output**



**Server side’s Output**

# REFERENCES

1. Simple File Transfer System using GUI and Socket Programming for Window Operating System [International Journal of Engineering Research & Technology (IJERT)]
2. Danny Briere, and Pat Hurley, "Wireless Home network For Dummies", 4th Edition, pp50.
3. Babaoglu, Ozalp (2012). "Introduction to Peer-to-Peer Systems". Complex Systems. Universitá di Bologna.
4. Winkelman, Dr. Roy. "Software". Florida Center for Instructional Technology College of Education, University of South Florida.
5. Tyson, Jeff." How the Old Napster Workded".
6. Vu, Quang H.; et al. (2010). Peer-to-Peer Computing: Principles and Applications. Springer. p. 8
7. Liu, Xiangning; Cheng, Lebin; Bhargava, Bharat; and Zhao, Zhiyuan, "Experimental Study of TCP and UDP Protocols for Future Distributed Databases" (1995). Department of Computer Science Technical Reports. Paper 1221.
8. Milanovic, A. & Srbljic, Sinisa & Sruk, Vlado. (2000). Performance of UDP and TCP communication on personal computers. 1. 286 - 289 vol.1. 10.1109/MELCON.2000.880422.
9. Kamil, Wisam & Awang Nor, Shahrudin & Alubady, Raaid. (2015). Research Article Performance Evaluation of TCP, UDP and DCCP Traffic Over 4G Network. Research Journal of Applied Sciences, Engineering and Technology. 11. 1048-1057. 10.19026/rjaset.11.2118.