Data Networks Project 4

Austin Warnock

**SOURCE CODE:**

**Server.c**

1. // Server side implementation of UDP client-server model
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. #include <sys/stat.h>
11. #include <time.h>
13. #define MAXLINE 1024
15. struct Packet
16. {
17. int seq\_num;
18. char data[256];
19. };
21. int sendFile(int sockfd, struct sockaddr\_in cliaddr, int len, char \*fileName)
22. {
23. int ack = 0, nack = 0;
24. char buff[256];
25. char response[256];
26. struct Packet packet;
27. packet.seq\_num = 0;
29. FILE \*fp;
30. fp = fopen(fileName, "r");
31. int n;
32. do
33. {
34. n = fread(buff, 256, 1, (FILE \*)fp);
35. printf("%d\n", n);
36. memcpy(&packet.data, buff, 256);
37. sendto(sockfd, (char \*)&packet, sizeof(packet),
38. MSG\_CONFIRM, (const struct sockaddr \*)&cliaddr,
39. len);
40. packet.seq\_num++;
41. recvfrom(sockfd, (char \*)response, 256,
42. MSG\_WAITALL, (struct sockaddr \*)&cliaddr,
43. &len);
44. ack++;
45. if (strncmp(response, "NACK", 4) == 0)
46. {
47. packet.seq\_num--;
48. sendto(sockfd, (char \*)&packet, sizeof(packet),
49. MSG\_CONFIRM, (const struct sockaddr \*)&cliaddr,
50. len);
51. packet.seq\_num++;
52. nack++;
53. }
54. } while (n > 0);
55. packet.seq\_num = -5;
56. sendto(sockfd, (char \*)&packet, sizeof(packet),
57. MSG\_CONFIRM, (const struct sockaddr \*)&cliaddr,
58. len);
60. fseek(fp, 0L, SEEK\_END);
61. int size = ftell(fp);
62. fclose(fp);
63. struct stat t\_stat;
64. stat(fileName, &t\_stat);
65. struct tm \*timeinfo = localtime(&t\_stat.st\_ctime);
67. printf("File Summary\n------------------\n");
68. printf("Reciever IP: %d\n Reciever Port: %d\n", cliaddr.sin\_addr, cliaddr.sin\_port);
69. printf("File Name: %s\n File size %d bytes: \n", fileName, size);
70. printf("Creation Date and Time: %s", asctime(timeinfo));
71. printf("Number of packets sent: %d\n", ack);
72. printf("Number of packets resent: %d\n", nack);
73. printf("Number of ACK's sent: %d\n", ack);
74. printf("Number of NACK's sent and SEQ #: %d and %d\n", nack, 2);
75. }
76. int recieveFileName(int sockfd, struct sockaddr\_in cliaddr, int len)
77. {
78. char fileName[MAXLINE];
79. recvfrom(sockfd, (char \*)fileName, 100,
80. MSG\_WAITALL, (struct sockaddr \*)&cliaddr,
81. &len);
82. FILE \*file;
83. if ((file = fopen(fileName, "r")) > 0)
84. {
85. fclose(file);
86. sendto(sockfd, (const char \*)"1", strlen("1"),
87. MSG\_CONFIRM, (const struct sockaddr \*)&cliaddr,
88. len);
89. sendFile(sockfd, cliaddr, len, fileName);
90. return 1;
91. }
92. else
93. {
94. sendto(sockfd, (const char \*)"-1", strlen("-1"),
95. MSG\_CONFIRM, (const struct sockaddr \*)&cliaddr,
96. len);
97. return -1;
98. }
99. }
100. char \*concat(const char \*s1, const char \*s2)
101. {
102. char \*result = malloc(strlen(s1) + strlen(s2) + 2);
103. strcpy(result, s1);
104. strcat(result, " ");
105. strcat(result, s2);
106. return result;
107. }
108. char \*verifyUser(char \*user, char \*pass)
109. {
110. FILE \*fp;
111. char buff[200];
112. char \*userAndPass = concat(user, pass);
113. int fr;
114. if ((fp = fopen("userList.txt", "r")) < 0)
115. {
116. printf("Error Opening userList.txt\n");
117. exit(0);
118. }
119. fseek(fp, 0, SEEK\_SET); //make sure we read from the beginning
121. if ((fr = fread(&buff, sizeof(char), 200, fp)) < 0)
122. {
123. printf("Error reading userlist.txt\n");
124. exit(0);
125. }
127. char \*array[9];
128. int i = 0;
130. array[i] = strtok(buff, "\n"); // put header into one array slot
132. while (array[i] != NULL)
133. {
134. array[++i] = strtok(NULL, "\n");
135. }
137. fflush(stdout);
139. for (i = 0; i < 7; i++)
140. {
141. if (strncmp(array[i], userAndPass, strlen(userAndPass)) == 0) //matching pair found
142. return "1";
143. }
144. return "-1";
145. }
147. // Driver code
148. int main(int argc, char \*\*argv)
149. {
150. const int PORT = atoi(argv[1]);
151. //const int WINDOW\_SIZE = 1;
152. int sockfd;
153. char user[MAXLINE], pass[MAXLINE];
154. struct sockaddr\_in servaddr, cliaddr;
156. // Creating socket file descriptor
157. if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0)
158. {
159. perror("socket creation failed");
160. exit(EXIT\_FAILURE);
161. }
163. memset(&servaddr, 0, sizeof(servaddr));
164. memset(&cliaddr, 0, sizeof(cliaddr));
166. // Filling server information
167. servaddr.sin\_family = AF\_INET; // IPv4
168. servaddr.sin\_addr.s\_addr = INADDR\_ANY;
169. servaddr.sin\_port = htons(PORT);
171. // Bind the socket with the server address
172. if (bind(sockfd, (const struct sockaddr \*)&servaddr,
173. sizeof(servaddr)) < 0)
174. {
175. perror("bind failed");
176. exit(EXIT\_FAILURE);
177. }
179. int len, n;
181. len = sizeof(cliaddr); //len is value/resuslt
183. n = recvfrom(sockfd, (char \*)user, MAXLINE,
184. MSG\_WAITALL, (struct sockaddr \*)&cliaddr,
185. &len);
186. user[n] = '\0';
188. n = recvfrom(sockfd, (char \*)pass, MAXLINE,
189. MSG\_WAITALL, (struct sockaddr \*)&cliaddr,
190. &len);
191. pass[n] = '\0';
193. sendto(sockfd, (const char \*)verifyUser(user, pass), strlen(verifyUser(user, pass)),
194. MSG\_CONFIRM, (const struct sockaddr \*)&cliaddr,
195. len);
197. recieveFileName(sockfd, cliaddr, len);
199. return 0;
200. }

Client.c

1. // Client side implementation of UDP client-server model
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. #include <sys/stat.h>
11. #include <time.h>
13. #define MAXLINE 1024
14. struct Packet
15. {
16. int seq\_num;
17. char data[256];
18. };
20. int readFile(int sockfd, struct sockaddr\_in servaddr, char \*fileName, char \*OUTPUT\_FILE)
21. {
22. int len, n, nack=1, ack=0, counter = 0;
23. struct Packet packet;
24. FILE \*fp = fopen(OUTPUT\_FILE, "ab+");
25. ;
27. do
28. {
29. n = recvfrom(sockfd, (char \*)&packet, sizeof(packet),
30. MSG\_WAITALL, (struct sockaddr \*)&servaddr,
31. &len);
32. counter++;
33. fwrite(packet.data, 1, strlen(packet.data), fp);
34. printf("Server : %s\n", packet.data);
35. if (rand() % 10 != 0)
36. {
37. sendto(sockfd, (char \*)"ACK", 256,
38. MSG\_CONFIRM, (const struct sockaddr \*)&servaddr,
39. sizeof(servaddr));
40. ack++;
41. }
42. if (packet.seq\_num != counter)
43. {
44. sendto(sockfd, (char \*)"NACK", 256,
45. MSG\_CONFIRM, (const struct sockaddr \*)&servaddr,
46. sizeof(servaddr));
47. nack++;
48. }
49. } while (packet.seq\_num > 0);
51. fseek(fp, 0L, SEEK\_END);
52. int size = ftell(fp);
53. fclose(fp);
54. struct stat t\_stat;
55. stat(fileName, &t\_stat);
56. struct tm \* timeinfo = localtime(&t\_stat.st\_ctime);
58. printf("File Summary\n------------------\n");
59. printf("File Name: %s\n File size %d bytes: \n", fileName, size);
60. printf("Creation Date and Time: %s", asctime(timeinfo));
61. printf("Number of ACK's sent: %d\n", ack);
62. printf("Number of NACK's sent: %d\n", nack);
63. }
64. int sendFile(int sockfd, struct sockaddr\_in servaddr, char \*OUTPUT\_FILE)
65. {
66. char fileName[100];
67. int len;
68. char buffer[MAXLINE];
69. do
70. {
71. printf("What is filename you want to send?\n");
72. gets(fileName);
73. sendto(sockfd, fileName, 100,
74. MSG\_CONFIRM, (const struct sockaddr \*)&servaddr,
75. sizeof(servaddr));
76. recvfrom(sockfd, (char \*)buffer, MAXLINE,
77. MSG\_WAITALL, (struct sockaddr \*)&servaddr,
78. &len);
79. } while (atoi(buffer) < 0);
80. readFile(sockfd, servaddr, fileName, OUTPUT\_FILE);
81. }
82. int auth(int sockfd, struct sockaddr\_in servaddr)
83. { //collect username and pass and send to server
84. char user[100];
85. char pass[100];
87. printf("What is your username?\n");
88. gets(user);
89. sendto(sockfd, user, strlen(user),
90. MSG\_CONFIRM, (const struct sockaddr \*)&servaddr,
91. sizeof(servaddr));
93. printf("What is your password?\n");
94. gets(pass);
95. sendto(sockfd, pass, strlen(pass),
96. MSG\_CONFIRM, (const struct sockaddr \*)&servaddr,
97. sizeof(servaddr));
98. return 1;
99. }
101. // Driver code
102. int main(int argc, char \*\*argv)
103. {
104. const int PORT = atoi(argv[2]);
105. const int WINDOW\_SIZE = 1;
106. const char \*INPUT\_FILE = argv[3];
107. const char \*OUTPUT\_FILE = argv[4];
108. int sockfd;
109. char buffer[MAXLINE];
110. struct sockaddr\_in servaddr;
112. // Creating socket file descriptor
113. if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0)
114. {
115. perror("socket creation failed");
116. exit(EXIT\_FAILURE);
117. }
119. memset(&servaddr, 0, sizeof(servaddr));
121. // Filling server information
122. servaddr.sin\_family = AF\_INET;
123. servaddr.sin\_port = htons(PORT);
124. servaddr.sin\_addr.s\_addr = inet\_addr(argv[1]);
126. int n, len;
127. auth(sockfd, servaddr);
129. n = recvfrom(sockfd, (char \*)buffer, MAXLINE,
130. MSG\_WAITALL, (struct sockaddr \*)&servaddr,
131. &len);
132. buffer[n] = '\0';
133. printf("Server : %s\n", buffer);
134. if (atoi(buffer) > 0)
135. {
136. sendFile(sockfd, servaddr, OUTPUT\_FILE);
137. }
139. close(sockfd);
140. return 0;
141. }

  Makefile:

1. all: server client
3. server: Server.c
4. gcc -pthread -o server Server.c -Wall -pedantic
6. client: Client.c
7. gcc -pthread -o client Client.c -Wall -pedantic
9. clean:
10. rm -f \*.o client server

**Approach:**

I first started by taking the source code given to us in Project 2 and modified it to accept a UDP socket. I then shifted my focus to creating a suitable test environment to develop and test my code. This mainly involved making a Makefile to generate all the binary executables in one quick command. I also read through and took some of the sample code from G4G to help get me started: <https://www.geeksforgeeks.org/udp-server-client-implementation-c/> and <https://www.geeksforgeeks.org/multithreading-c-2/> . Once I got the main pieces of my code working, I abstracted them out into several functions to make the code a lot more readable. I also managed to setup vscode remote which made it a lot easier to debug my code.

Functions in Client.c:

* readFile() – takes in a file descriptor, and server socket in order to form a loop that reads 90% of the messages sent out by the server. It also handles sending ACKs and NACKs as well as providing the summary of data at the end
* sendFile()– takes in a file descriptor, and server socket in order to form a loop to ask the user for a valid filename.
* auth()– takes in a file descriptor, and server socket in order to validate username and password combos

Functions in Server.c

* sendFile() – takes in a file descriptor, and server socket in order to form a loop that reads 90% of the messages sent out by the server. It also handles sending ACKs and NACKs as well as providing the summary of data at the end
* recieveFile()– takes in a file descriptor, and server socket in order to form a loop to ask the user for a valid filename.
* concat() – takes two strings and concatenates them for comparison later.
* veryifyUser() – takes username and password and checks if valid pair.

Data Structures:

1. struct Packet
2. {
3. int seq\_num;
4. char data[256];
5. };

The packet struct is a way for us to track the sequence number of a given packet as we send it over. This way, if there is an error in transmission, we can go back and ask for the correct packet again.

**Assumptions:**

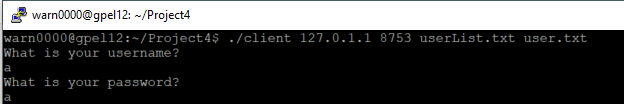
This program assumes that usernames are case and whitespace sensitive and treats them as such. i.e Anna =/= anna. This is assumed because case and whitespace sensitivity would make sense in a secure password-based application.

**Screenshots:**

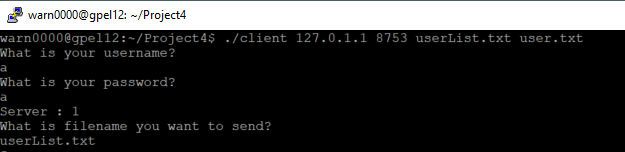
Server startup:



Client Startup:

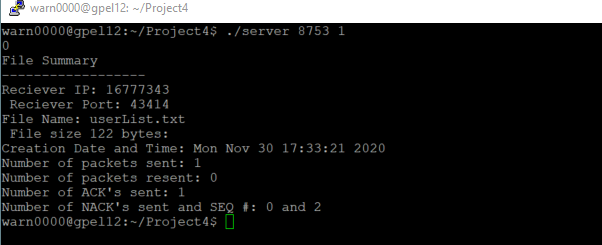


Client when there is invalid file name:

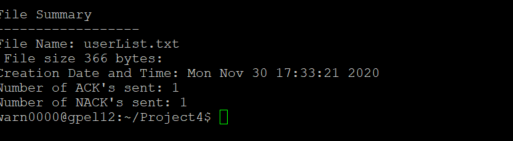


Sending Test:

Server:



Client:



Created File:

