Data Networks Project 1

Austin Warnock

**SOURCE CODE:**

Sender.C

1. #include <stdio.h>
2. #include <sys/types.h>
3. #include <sys/socket.h>
4. #include <netinet/in.h>
5. #include <time.h>
6. #include <string.h>
7. #include <arpa/inet.h>
8. #include <stdlib.h>
9. #include <unistd.h>
10. #define MAX 50
11. // Function designed for chat between client and server.
12. void readPortName(int sockfd)
13. {
14. char buff[MAX];
15. int n;
16. bzero(buff, sizeof(buff));
17. printf("\nWhat is your Port Name? ");
18. n = 0;
19. while ((buff[n++] = getchar()) != '\n')
20. ;
21. write(sockfd, buff, sizeof(buff));
22. bzero(buff, sizeof(buff));
23. read(sockfd, buff, sizeof(buff));
24. printf("From Server : %s", buff);
25. }
26. int readIpAddress(int sockfd)
27. {
28. char buff[MAX];
29. int n;
30. bzero(buff, sizeof(buff));
31. printf("\nWhat is your Port Number? ");
32. n = 0;
33. while ((buff[n++] = getchar()) != '\n')
34. ;
35. write(sockfd, buff, sizeof(buff));
36. bzero(buff, sizeof(buff));
37. read(sockfd, buff, sizeof(buff));
38. printf("From Server : %s", buff);
39. return atoi(buff);
40. }
41. void readUsername(int sockfd) //function to handle username read/write
42. {
43. char buff[MAX];
44. int n;
45. bzero(buff, sizeof(buff));
46. printf("\nWhat is your username? ");
47. n = 0;
48. while ((buff[n++] = getchar()) != '\n')
49. ;
50. write(sockfd, buff, sizeof(buff));
51. bzero(buff, sizeof(buff));
52. read(sockfd, buff, sizeof(buff));
53. printf("From Server : %s", buff);
54. }
55. int readPassword(int sockfd) // function to handle password read/write
56. {
57. char buff[MAX];
58. int n;
59. bzero(buff, sizeof(buff));
60. printf("What is your password? ");
61. n = 0;
62. while ((buff[n++] = getchar()) != '\n')
63. ;
64. write(sockfd, buff, sizeof(buff));
65. bzero(buff, sizeof(buff));
66. read(sockfd, buff, sizeof(buff));
67. printf("From Server : %s", buff);
68. return atoi(buff);
69. }
70. int readHammingStrings(int sockfd)
71. {
72. char buff[MAX];
73. int n;
74. bzero(buff, sizeof(buff));
75. printf("\nWhat is your first String? ");
76. n = 0;
77. while ((buff[n++] = getchar()) != '\n')
78. ;
79. write(sockfd, buff, sizeof(buff));
80. if((strncmp(&buff[0], "q", 1)) == 0){
81. printf("Got The close Signal");
82. exit(0);
83. }
84. bzero(buff, sizeof(buff));
85. printf("\nWhat is your second String? ");
86. n = 0;
87. while ((buff[n++] = getchar()) != '\n')
88. ;
89. write(sockfd, buff, sizeof(buff));
90. if((strncmp(&buff[0], "q", 1)) == 0){
91. printf("Got The close Signal");
92. exit(0);
93. }
94. bzero(buff, sizeof(buff));
95. read(sockfd, buff, sizeof(buff));
96. printf("From Server : %s", buff);
97. return 1;
98. }
99. int main(int argc, char \*\*argv)
100. {
101. int clisoc;
102. struct sockaddr\_in cliaddr;
103. bzero(&cliaddr, sizeof(cliaddr));
104. cliaddr.sin\_family = AF\_INET;
105. cliaddr.sin\_port = htons(atoi(argv[2]));
106. cliaddr.sin\_addr.s\_addr = inet\_addr(argv[1]);
107. if ((clisoc = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)
108. {
109. perror("\nSocket Errror");
110. exit(0);
111. }
112. else
113. {
114. printf("\nSocket opened");
115. }
116. if ((connect(clisoc, (struct sockaddr \*)&cliaddr, sizeof(cliaddr))) < 0)
117. {
118. perror("\nConnect Error");
119. exit(0);
120. }
121. else
122. {
123. printf("\nConnected Successfully");
124. }
125. do
126. {
127. readUsername(clisoc);
128. } while (readPassword(clisoc) < 0);
129. //broke out of do while - must have successful pair
130. do
131. {
132. readPortName(clisoc);
133. } while (readIpAddress(clisoc) < 0);
134. //broke out of do while - must have successful pair
135. while(readHammingStrings(clisoc) > 0){
136. readHammingStrings(clisoc);
137. }
138. return 0;
139. }
140. Server.c
141. #include <stdio.h>
142. #include <sys/types.h>
143. #include <sys/socket.h>
144. #include <netinet/in.h>
145. #include <time.h>
146. #include <string.h>
147. #include <arpa/inet.h>
148. #include <stdlib.h>
149. #include <unistd.h>
150. #define MAX 50
152. char \*concat(const char \*s1, const char \*s2)
153. {
154. char \*result = malloc(strlen(s1) + strlen(s2) + 2);
155. strcpy(result, s1);
156. strcat(result, " ");
157. strcat(result, s2);
158. return result;
159. }
160. void handleReciever(int senderfd, int recieverfd)
161. {
162. char buff[MAX];
163. bzero(buff, MAX); //clear buffer
164. int n;
165. // read the message from client
166. read(senderfd, buff, sizeof(buff));
167. while ((n=strncmp(&buff[0], "q", 1)) != 0)
168. {
169. write(recieverfd, buff, sizeof(buff));
170. read(senderfd, buff, sizeof(buff));
171. write(recieverfd, buff, sizeof(buff));
172. read(recieverfd, buff, sizeof(buff));
173. write(senderfd, buff, sizeof(buff));
174. }
175. //we got the quit code now close
176. write(recieverfd, buff, sizeof(buff));
177. //write(senderfd, buff, sizeof(buff));
178. close(senderfd);
179. close(recieverfd);
180. printf("CLOSING CONNECTIONS");
181. }
182. void openReciever(char \*portName, char \*ipAddress, int portNumber, int senderfd)
183. {
184. int clisoc;
185. struct sockaddr\_in cliaddr;
186. bzero(&cliaddr, sizeof(cliaddr));
187. cliaddr.sin\_family = AF\_INET;
188. printf("%i\n", portNumber);
189. cliaddr.sin\_port = htons(portNumber);
190. cliaddr.sin\_addr.s\_addr = inet\_addr(ipAddress);
192. if ((clisoc = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)
193. {
194. perror("\nSocket Errror");
195. exit(0);
196. }
197. else
198. {
199. printf("\nSocket opened");
200. }
201. if ((connect(clisoc, (struct sockaddr \*)&cliaddr, sizeof(cliaddr))) < 0)
202. {
203. perror("\nConnect Error");
204. exit(0);
205. }
206. else
207. {
208. printf("\nConnected Successfully");
209. }
211. for (;;)
212. {
213. handleReciever(senderfd, clisoc);
214. }
215. }
216. int passwordVerification(char \*username, char \*password)
217. {
218. FILE \*fp;
219. char buff[80];
220. char \*userAndPass = concat(username, password);
221. int fr;
222. if ((fp = fopen("userList.txt", "r")) < 0)
223. {
224. printf("Error Opening userList.txt\n");
225. exit(0);
226. }
227. fseek(fp, 0, SEEK\_SET); //make sure we read from the beginning
229. if ((fr = fread(&buff, sizeof(char), 80, fp)) < 0)
230. {
231. printf("Error reading userlist.txt\n");
232. exit(0);
233. }
235. char \*array[9];
236. int i = 0;
238. array[i] = strtok(buff, "\n"); // put header into one array slot
240. while (array[i] != NULL)
241. {
242. array[++i] = strtok(NULL, "\n");
243. }
245. fflush(stdout);
247. for (i = 1; i < 5; i++)
248. {
249. if (strncmp(array[i], userAndPass, strlen(userAndPass)) == 0) //matching pair found
250. return 1;
251. }
252. return -1;
253. }
254. int portVerification(char \*portName, char \*ipAddress)
255. {
256. FILE \*fp;
257. char buff[250];
258. char \*combinedPort = concat(portName, ipAddress);
259. int fr;
260. if ((fp = fopen("recieverList.txt", "r")) < 0)
261. {
262. printf("Error Opening recieverList.txt\n");
263. exit(0);
264. }
265. fseek(fp, 0, SEEK\_SET); //make sure we read from the beginning
267. if ((fr = fread(&buff, sizeof(char), 250, fp)) < 0)
268. {
269. printf("Error reading recieverlist.txt\n");
270. exit(0);
271. }
273. char \*array[9];
274. int i = 0;
276. array[i] = strtok(buff, "\n"); // put header into one array slot
278. while (array[i] != NULL)
279. {
280. array[++i] = strtok(NULL, "\n");
281. }
283. fflush(stdout);
285. for (i = 1; i < 7; i++)
286. {
287. if (strncmp(array[i], combinedPort, strlen(combinedPort)) == 0) //matching pair found
288. return 1;
289. }
290. return -1;
291. }
292. int handleUsername(int sockfd, char \*username)
293. {
294. char buff[MAX];
295. int n = 0;
296. bzero(buff, MAX); //clear buffer
298. // read the message from client
299. read(sockfd, buff, sizeof(buff));
301. strcpy(username, buff); // copy over the results
302. username[strlen(username) - 1] = '\0'; // removes newline for comparison later
303. bzero(buff, MAX); //clear buffer
305. // copy server message in the buffer
306. char successMessage[MAX] = "Successfully Recieved Username\n";
307. memcpy(&buff[0], successMessage, strlen(successMessage));
308. while ((buff[n++]) != '\n')
309. ;
310. // and send that buffer to client and zero it out
311. write(sockfd, buff, sizeof(buff));
312. return 1;
313. }
314. int handlePassword(int sockfd, char \*username, char \*password)
315. {
316. char buff[MAX];
317. int n = 0;
318. double isValid = -6;
319. bzero(buff, MAX); //clear buffer
320. // read the message from client
321. read(sockfd, buff, sizeof(buff));
323. strcpy(password, buff);
324. password[strlen(password) - 1] = '\0'; // removes newline for comparison later
325. bzero(buff, MAX); //clear buffer
327. isValid = passwordVerification(username, password);
328. // copy server message in the buffer
329. char successMessage[MAX] = "-1\n";
330. if (isValid > 0)
331. {
332. strcpy(successMessage, "Successful Username Password Combo\n");
333. }
334. memcpy(&buff[0], successMessage, strlen(successMessage));
335. while ((buff[n++]) != '\n')
336. ;
337. // and send that buffer to client and zero it out
338. write(sockfd, buff, sizeof(buff));
339. return isValid;
340. }
341. int handlePortName(int sockfd, char \*portName)
342. {
343. char buff[MAX];
344. int n = 0;
345. bzero(buff, MAX); //clear buffer
347. // read the message from client
348. read(sockfd, buff, sizeof(buff));
350. strcpy(portName, buff); // copy over the results
351. portName[strlen(portName) - 1] = '\0'; // removes newline for comparison later
352. bzero(buff, MAX); //clear buffer
354. // copy server message in the buffer
355. char successMessage[MAX] = "Successfully Recieved Port Name\n";
356. memcpy(&buff[0], successMessage, strlen(successMessage));
357. while ((buff[n++]) != '\n')
358. ;
359. // and send that buffer to client and zero it out
360. write(sockfd, buff, sizeof(buff));
361. return 1;
362. }
363. int handleipAddress(int sockfd, char \*portName, char \*ipAddress, int portNumber)
364. {
365. char buff[MAX];
366. int n = 0;
367. double isValid = -6;
368. bzero(buff, MAX); //clear buffer
369. // read the message from client
370. read(sockfd, buff, sizeof(buff));
372. strcpy(ipAddress, buff);
373. ipAddress[strlen(ipAddress) - 1] = '\0'; // removes newline for comparison later
374. bzero(buff, MAX); //clear buffer
376. isValid = portVerification(portName, ipAddress);
377. // copy server message in the buffer
378. char successMessage[MAX] = "-1\n";
379. if (isValid > 0)
380. {
381. strcpy(successMessage, "Successful Username Port Combo\n");
382. }
383. memcpy(&buff[0], successMessage, strlen(successMessage));
384. while ((buff[n++]) != '\n')
385. ;
386. // and send that buffer to client and zero it out
387. write(sockfd, buff, sizeof(buff));
388. if (isValid > 0)
389. {
390. openReciever(portName, ipAddress, portNumber, sockfd);
391. }
392. return isValid;
393. }
394. int main(int argc, char \*\*argv)
395. {
396. struct sockaddr\_in sa;
397. struct sockaddr\_in cli;
398. int sockfd, conntfd;
399. socklen\_t len;
400. int ch;
401. sockfd = socket(AF\_INET, SOCK\_STREAM, 0);
402. if (sockfd < 0)
403. {
404. printf("error in socket\n");
405. exit(0);
406. }
407. else
408. printf("\nSocket opened");
409. bzero(&sa, sizeof(sa));
410. sa.sin\_port = htons(atoi(argv[1]));
411. sa.sin\_addr.s\_addr = htonl(0);
412. if (bind(sockfd, (struct sockaddr \*)&sa, sizeof(sa)) < 0)
413. {
414. printf("Error in binding\n");
415. }
416. else
417. printf("\nBinded Successfully\n");
418. listen(sockfd, 50);
419. len = sizeof(ch);
421. char username[MAX];
422. char password[MAX];
423. char ipAddress[MAX];
424. char portName[MAX];
425. for (;;)
426. {
427. conntfd = accept(sockfd, (struct sockaddr \*)&cli, &len);
428. do
429. {
430. handleUsername(conntfd, username);
431. } while (handlePassword(conntfd, username, password) < 0);
433. do
434. {
435. handlePortName(conntfd, portName);
436. } while (handleipAddress(conntfd, portName, ipAddress, atoi(argv[1]) + 1) < 0);
437. }
438. }

 Reciever.C

1. #include "netinet/in.h"
2. #include "sys/socket.h"
3. #include "stdio.h"
4. #include "string.h"
5. #include "time.h"
6. #include <stdlib.h>
7. #include <unistd.h>
8. #include <signal.h>
9. #define MAX 100
10. int hammingDist(char \*str1, char \*str2)
11. {
12. printf("Str1 :%s and Str 2:%s \n", str1, str2);
13. int i = 0, count = 0;
14. while (str1[i] != '\0')
15. {
16. if (str1[i] != str2[i])
17. count++;
18. i++;
19. }
20. return count;
21. }
22. int main(int argc, char \*\*argv){
23. struct sockaddr\_in sa;
24. struct sockaddr\_in cli;
25. int sockfd, conntfd;
26. socklen\_t len;
27. int ch;
28. sockfd = socket(AF\_INET, SOCK\_STREAM, 0);
29. if (sockfd < 0)
30. {
31. printf("error in socket\n");
32. exit(0);
33. }
34. else
35. printf("\nSocket opened");
36. bzero(&sa, sizeof(sa));
37. sa.sin\_port = htons(atoi(argv[1]));
38. sa.sin\_addr.s\_addr = htonl(0);
39. if (bind(sockfd, (struct sockaddr \*)&sa, sizeof(sa)) < 0)
40. {
41. printf("Error in binding\n");
42. }
43. else
44. printf("\nBinded Successfully\n");
45. listen(sockfd, 50);
46. len = sizeof(ch);
48. char storageArray[100][MAX];
49. int numStored = 0;
50. for (;;)
51. {
52. conntfd = accept(sockfd, (struct sockaddr \*)&cli, &len);
54. char buff[MAX];
55. char response[MAX];
56. bzero(buff, MAX); //clear buffer
58. // read the message from client
59. read(conntfd, buff, sizeof(buff));
60. strncpy(storageArray[numStored],buff, MAX);
61. numStored++;
63. read(conntfd, buff, sizeof(buff));
64. strncpy(storageArray[numStored],buff, MAX);
65. numStored++;
66. if(numStored % 2 == 0){
67. int ham;
68. ham = hammingDist(storageArray[numStored-2],storageArray[numStored-1]);
69. sprintf(response, "The Hamming Distance is %i\n", ham);
70. printf("%s\n", response);
71. write(conntfd, response, sizeof(response));
72. }
74. }
75. }

 Makefile:

1. all: server sender reciever
2. server: Server.c
3. gcc -o server Server.c
4. sender: Sender.c
5. gcc -o sender Sender.c
6. reciever: Reciever.c
7. gcc -o reciever Reciever.c
8. clean:
9. rm -f \*.o reciever server sender

**Approach:**

I first started by taking the source code given to us in Project 1 and modified it to accept any port as a parameter. 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 added some shorter username password pairs and port and IP pairs in order to make testing the later functions easier. 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.

Functions in Sender.c:

* readPortName() – takes in the socket file descriptor and prompts the user for port name. It then takes that value and writes it to the server and then prints the server’s response
* readIpAddress() – takes in the socket file descriptor and prompts the user for IP address. It then takes that value and writes it to the server and then prints the server’s response
* readUsername() – takes in the socket file descriptor and prompts the user for username. It then takes that value and writes it to the server and then prints the server’s response
* readPassword() – takes in the socket file descriptor and prompts the user for password type. It then takes that value and writes it to the server and then prints the server’s response
* readHammingStrings() – takes in the socket file descriptor and prompts the user for their hamming strings. It then takes that value and writes it to the server and then prints the server’s response

Functions in Server.c

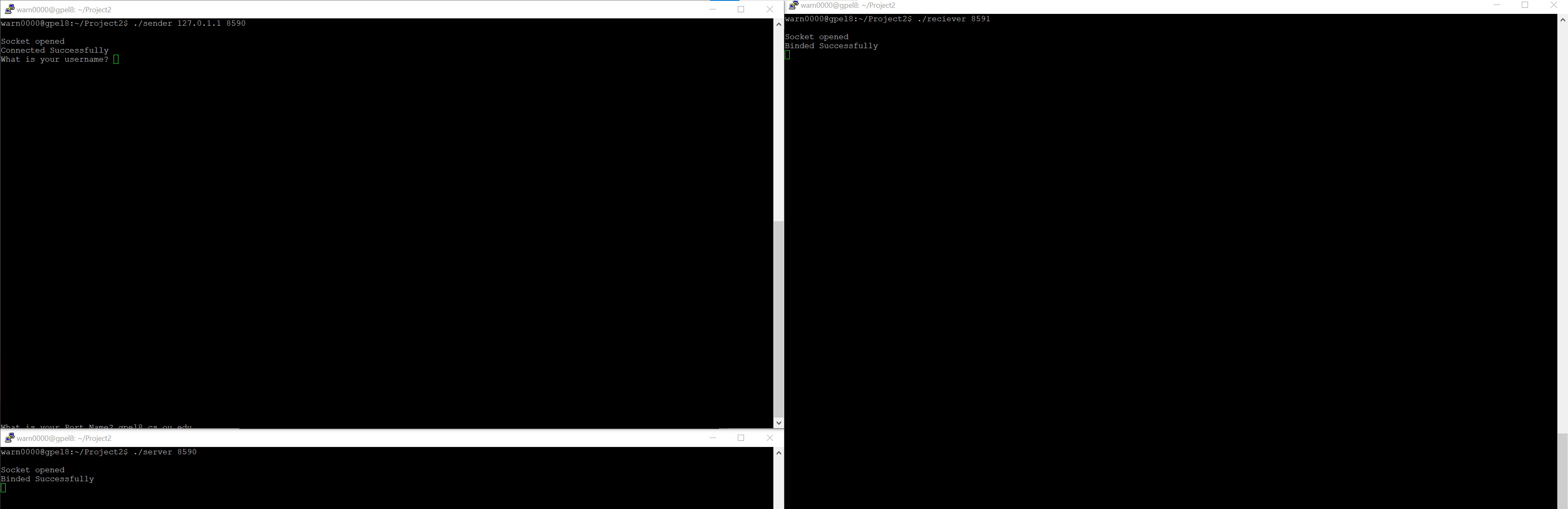
* concat() – takes two strings and concatenates them for comparison later.
* handleReciever() – takes the two socket file descriptors and manages the communication between sender and receiver.
* openReciever() – takes portName, ipaddress, portNumber, and senderfd in order to open communication between receiver and server. This method then calls the handleReceiver() function with the sender file descriptor.
* passwordVerification() – takes username and password and checks if valid pair.
* portVerification() – takes port name and IP address and checks if valid pair.
* handleUsername() – takes socket file descriptor and handles getting the username from sender
* handlePassword() – takes socket file descriptor and handles getting the password from sender
* handlePortName() – takes socket file descriptor and handles getting the port name from sender
* handleIpAddress() – takes socket file descriptor and handles getting the IpAddress from sender

**Assumptions:**

This program assumes that username password pairs 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:**

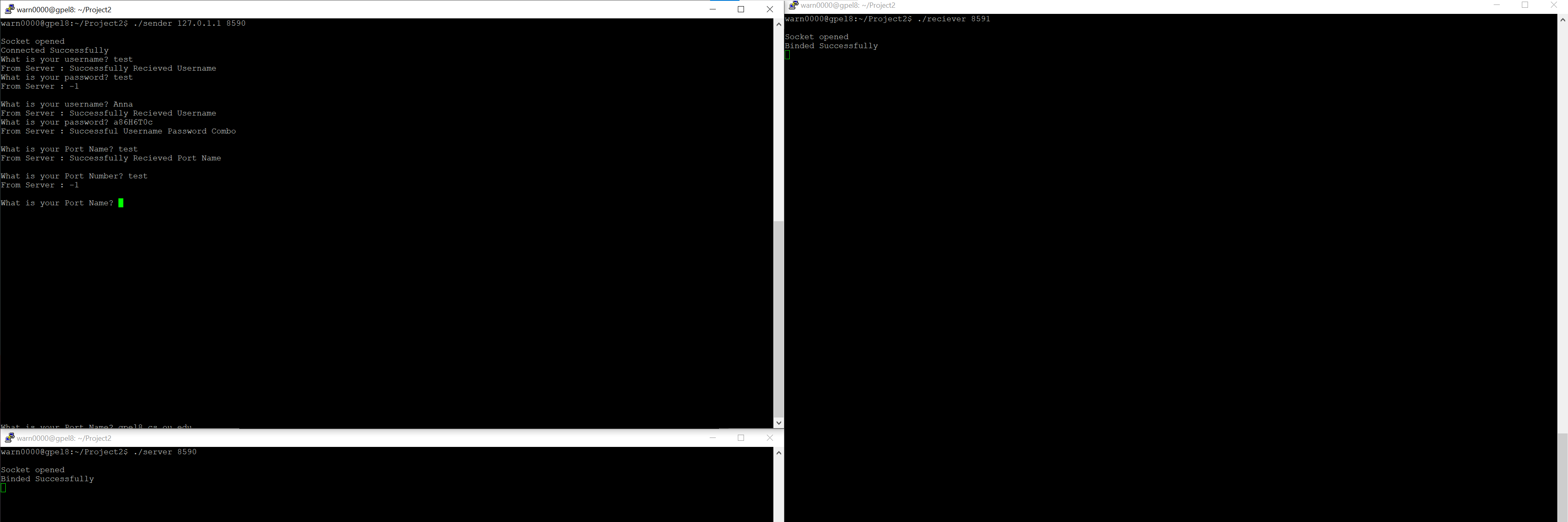
Startup:



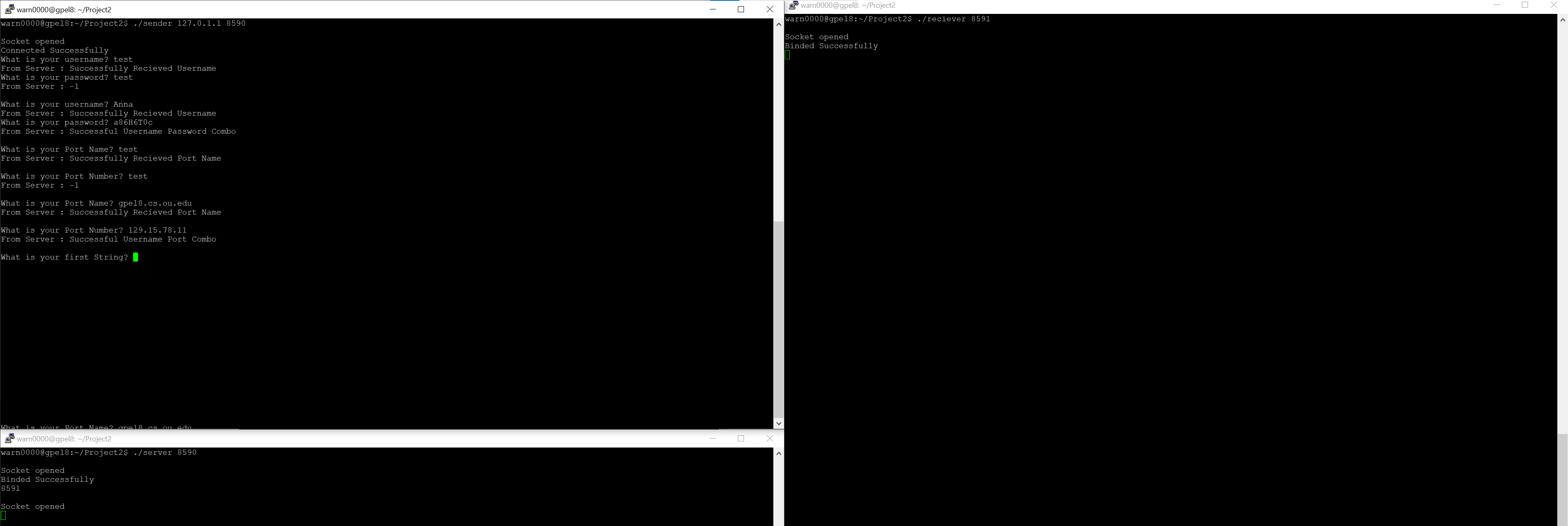
Error in username password combo:



Error in port ip combo:



Ask for strings:



Test Hamming:

