Data Networks Project 1

Austin Warnock

**SOURCE CODE:**

Client.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 readCurency(int sockfd) //function to handle currency read/write
13. {
14. char buff[MAX];
15. int n;
16. bzero(buff, sizeof(buff));
17. printf("\nWhat type of currency do you have? ");
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. void readPassword(int sockfd) // function to handle password read/write
27. {
28. char buff[MAX];
29. int n;
30. bzero(buff, sizeof(buff));
31. printf("What is your password? ");
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. }
40. int main(int argc, char \*\*argv)
41. {
42. int clisoc;
43. struct sockaddr\_in cliaddr;
44. bzero(&cliaddr, sizeof(cliaddr));
45. cliaddr.sin\_family = AF\_INET;
46. cliaddr.sin\_port = htons(atoi(argv[2]));
47. cliaddr.sin\_addr.s\_addr = inet\_addr(argv[1]);
48. if ((clisoc = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)
49. {
50. perror("\nSocket Errror");
51. exit(0);
52. }
53. else
54. {
55. printf("\nSocket opened");
56. }
57. if ((connect(clisoc, (struct sockaddr \*)&cliaddr, sizeof(cliaddr))) < 0)
58. {
59. perror("\nConnect Error");
60. exit(0);
61. }
62. else
63. {
64. printf("\nConnected Successfully");
65. }
66. readCurency(clisoc);
67. readPassword(clisoc);
68. int cl;
69. cl = close(clisoc);
70. printf("Socket closed with code: %i\n", cl);
71. return 0;
72. }

 Server.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 50
10. void exitfunc(int sig)
11. {
12. exit(0);
13. }
14. double passwordVerification(char \*userCurrency, char \*userPassword)
15. {
16. const char \*CURRENCY\_ARRAY[6] = {"US Dollar", "Canadian Dollar", "Euro", "British Pound", "Japanese Yen", "Swiss Franc"};
17. const char \*PASSWORD\_ARRAY[6] = {"uCh781fY", "Cfw61RqV", "Pd82bG57", "Crc51RqV", "wD82bV67", "G6M7p8az"};
18. const double BITCOIN\_ARRAY[6] = {11081.00, 14632.87, 9359.20, 8578.96, 1158748.55, 10100.44};
19. for (int i = 0; i < 6; ++i)
20. {
21. if (strcmp(userCurrency, CURRENCY\_ARRAY[i]) == 0)
22. {
23. if (strcmp(userPassword, PASSWORD\_ARRAY[i]) == 0)
24. {
25. return BITCOIN\_ARRAY[i];
26. }
27. }
28. }
29. return -1;
30. }
31. int handleCurrency(int sockfd, char \*currency)
32. {
33. signal(SIGALRM, exitfunc);
34. alarm(30); //alarm to end program after 30 seconds of inactivity
35. char buff[MAX];
36. int n = 0;
37. bzero(buff, MAX); //clear buffer
38. // read the message from client
39. read(sockfd, buff, sizeof(buff));
40. alarm(0); // cancel alarm because we got some data
41. strcpy(currency, buff); // copy over the results
42. currency[strlen(currency) - 1] = '\0'; // removes newline for comparison later
43. bzero(buff, MAX); //clear buffer
44. // copy server message in the buffer
45. char successMessage[MAX] = "Successfully Recieved Currency Type\n";
46. memcpy(&buff[0], successMessage, strlen(successMessage));
47. while ((buff[n++]) != '\n')
48. ;
49. // and send that buffer to client and zero it out
50. write(sockfd, buff, sizeof(buff));
51. return 1;
52. }
53. int handlePassword(int sockfd, char \*currency, char \*password)
54. {
55. signal(SIGALRM, exitfunc);
56. alarm(30); //alarm to end program after 30 seconds of inactivity
57. char buff[MAX];
58. int n = 0;
59. double bitcoin\_value = -6;
60. bzero(buff, MAX); //clear buffer
61. // read the message from client
62. read(sockfd, buff, sizeof(buff));
63. alarm(0); // cancel alarm because we got some data
64. strcpy(password, buff);
65. password[strlen(password) - 1] = '\0'; // removes newline for comparison later
66. bzero(buff, MAX); //clear buffer
67. bitcoin\_value = passwordVerification(currency, password);
68. // copy server message in the buffer
69. char successMessage[MAX] = "Invalid Currency Password Combo\n";
70. if (bitcoin\_value > 0)
71. {
72. sprintf(successMessage, "%s%.2f", "Your Bitcoin value is: \n", bitcoin\_value);
73. }
74. memcpy(&buff[0], successMessage, strlen(successMessage));
75. while ((buff[n++]) != '\n')
76. ;
77. // and send that buffer to client and zero it out
78. write(sockfd, buff, sizeof(buff));
79. close(sockfd);
80. return 1;
81. }
82. int main(int argc, char \*\*argv)
83. {
84. struct sockaddr\_in sa;
85. struct sockaddr\_in cli;
86. int sockfd, conntfd;
87. socklen\_t len;
88. int ch;
89. sockfd = socket(AF\_INET, SOCK\_STREAM, 0);
90. if (sockfd < 0)
91. {
92. printf("error in socket\n");
93. exit(0);
94. }
95. else
96. printf("\nSocket opened");
97. bzero(&sa, sizeof(sa));
98. sa.sin\_port = htons(atoi(argv[1]));
99. sa.sin\_addr.s\_addr = htonl(0);
100. if (bind(sockfd, (struct sockaddr \*)&sa, sizeof(sa)) < 0)
101. {
102. printf("Error in binding\n");
103. }
104. else
105. printf("\nBinded Successfully");
106. listen(sockfd, 50);
107. len = sizeof(ch);
108. char currency[MAX];
109. char password[MAX];
110. for (;;)
111. {
112. conntfd = accept(sockfd, (struct sockaddr \*)&cli, &len);
113. handleCurrency(conntfd, currency);
114. handlePassword(conntfd, currency, password);
115. }
116. }

Makefile:

1. all: server client
2. server: Server.c
3. gcc -o server Server.c
4. client: Client.c
5. gcc -o client Client.c
6. clean:
7. rm -f \*.o client server

**Approach:**

I first started by taking the source code given to us in Project 0 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. 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 Client.c:

* readCurrency() – takes in the socket file descriptor and prompts the user for currency type. 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

Functions in Server.c

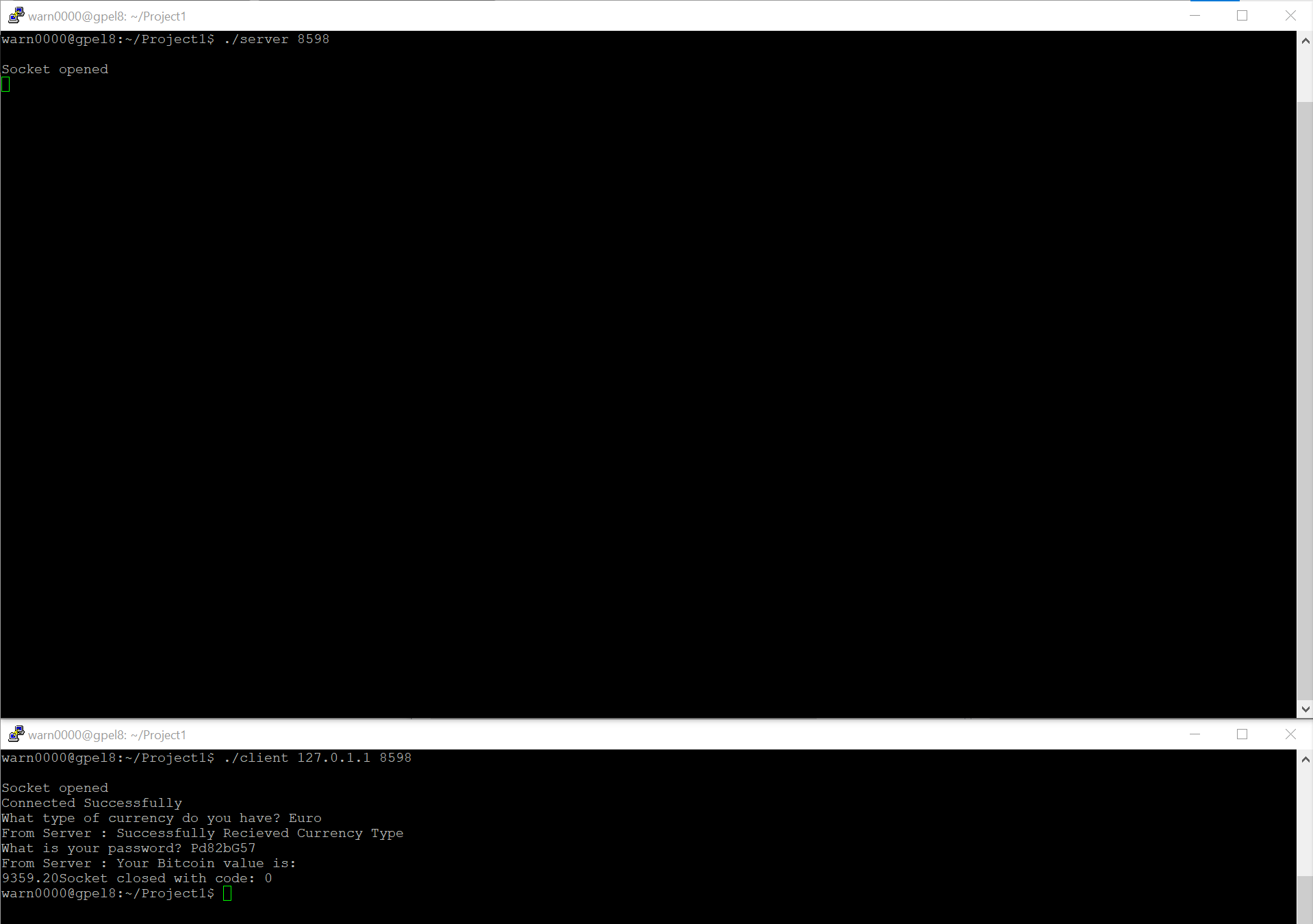
* exitfunc() – alarm handler to shut the server down after 30 seconds of inactivity
* passwordVerification() – takes in user currency and password and uses a for loop to compare the index of the currency value in a currency array with the corresponding index in a password array to see if they the form a valid pair. If so, the index is again used to return the corresponding bitcoin value. If not, the function returns -1.
* handleCurrency() – server equivalent of readCurrency(). Has alarm function to shut the socket down after 30 seconds of inactivity.
* handlePassword() – server equivalent of readPassword(). Has alarm function to shut the socket down after 30 seconds of inactivity.
* handleCurrency() – server equivalent of readCurrency(). Has alarm function to shut the socket down after 30 seconds of inactivity.

**Assumptions:**

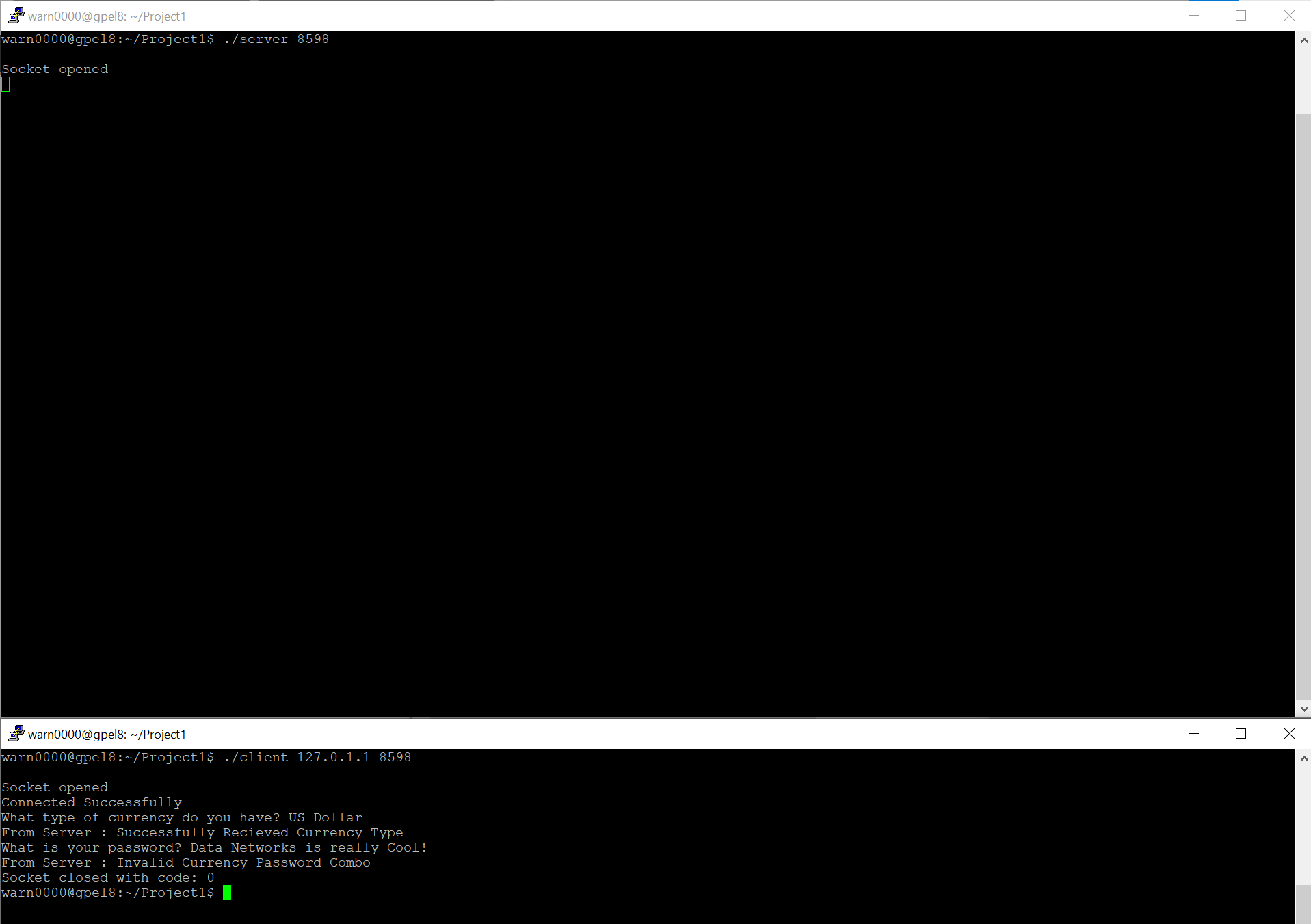
This program assumes that currency password pairs are case and whitespace sensitive and treats them as such. i.e US Dollar =/= usdollar. This is assumed because case and whitespace sensitivity would make sense in a secure banking application. This program also assumes that after 30 seconds of inactivity the user is no longer at the computer and therefore the code just shuts down the socket without regard towards saving any previously input data.

**Screenshots:**

Example of successful password currency combination:

****

Example of unsuccessful password currency combination:



Example of server timeout:

