Operating systems, security and networks (207SE)

Lab 17: TCP Server

# Your task

Modify the given TCP server (**use the tcp\_server.cc and tcp\_client.cc and Makefile code on moodle**) to create either a) or b). **Do not do both:**

1. Simple RPN (Reverse Polish Notation) calculator server. For example, if the client sends three inputs (3, 5 and +), then the server will return 8. You should allow addition, subtraction, multiplication and division. (Up to 3 out of 5)

Extend the program to allow more general calculations by implementing a stack for values and operators that apply to the top pair. Example, if the client sends the input (3 5 6 + \*) 5 and 6 should be added together and the result multiplied by 3. See the W ikipedia article on RPN for a good description. (for up to 5 out of 5)

**Lab Activity 17 TCP Server**

1. Brief description of the TCP Server Activity
2. Commented Code showing changing to tcp-server.cc tcp-client.cc

[Commented code here]

**tcp-client.cc**

1. #include <netdb.h>
2. #include <netinet/in.h>
3. #include <unistd.h>
4. #include <iostream>
5. #include <cstring>
6. #include <stdlib.h>
8. #define MAX\_LINE 100
9. #define LINE\_ARRAY\_SIZE (MAX\_LINE+1)
11. **using** **namespace** std;
13. **int** main()
14. {
15. **int** socketDescriptor;
16. unsigned **short** **int** serverPort;
17. **struct** sockaddr\_in serverAddress;
18. **struct** hostent \*hostInfo;
19. **char** buf[LINE\_ARRAY\_SIZE], c;
21. cout << "Enter server host name or IP address: ";
22. cin.get(buf, MAX\_LINE, '\n');
24. // gethostbyname() takes a host name or ip address in "numbers and
25. // dots" notation, and returns a pointer to a hostent structure,
26. // which we'll need later.  It's not important for us what this
27. // structure is actually composed of.
28. hostInfo = gethostbyname(buf);
29. **if** (hostInfo == NULL) {
30. cout << "problem interpreting host: " << buf << "\n";
31. exit(1);
32. }
34. cout << "Enter server port number: ";
35. cin >> serverPort;
36. cin.get(c); // dispose of the newline
38. // Create a socket.  "AF\_INET" means it will use the IPv4 protocol.
39. // "SOCK\_STREAM" means it will be a reliable connection (i.e., TCP;
40. // for UDP use SOCK\_DGRAM), and I'm not sure what the 0 for the last
41. // parameter means, but it seems to work.
42. socketDescriptor = socket(AF\_INET, SOCK\_STREAM, 0);
43. **if** (socketDescriptor < 0) {
44. cerr << "cannot create socket\n";
45. exit(1);
46. }
48. // Connect to server.  First we have to set some fields in the
49. // serverAddress structure.  The system will assign me an arbitrary
50. // local port that is not in use.
51. serverAddress.sin\_family = hostInfo->h\_addrtype;
52. memcpy((**char** \*) &serverAddress.sin\_addr.s\_addr,
53. hostInfo->h\_addr\_list[0], hostInfo->h\_length);
54. serverAddress.sin\_port = htons(serverPort);
56. **if** (connect(socketDescriptor,
57. (**struct** sockaddr \*) &serverAddress,
58. **sizeof**(serverAddress)) < 0) {
59. cerr << "cannot connect\n";
60. exit(1);
61. }
63. cout << "\nEnter an RPN Expression(use space in between):\n";

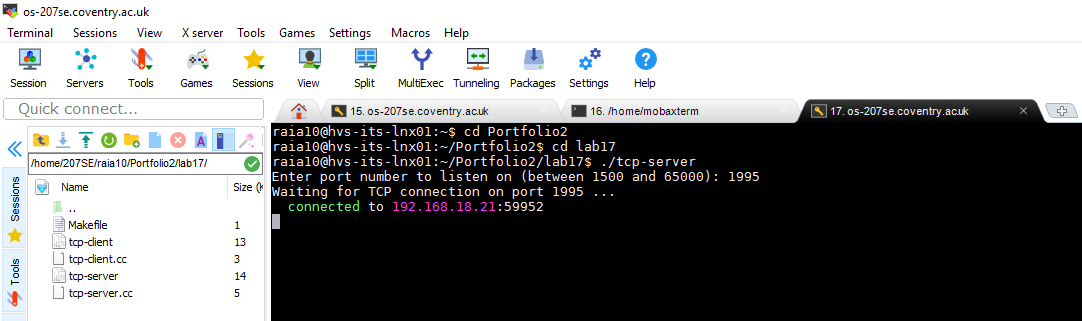

67. // Prompt the user for input, then read in the input, up to MAX\_LINE
68. // charactars, and then dispose of the rest of the line, including
69. // the newline character.
70. cout << "Input: ";
71. cin.get(buf, MAX\_LINE, '\n');
72. **while** (cin.get(c) && c != '\n')
73. ; //Loop does nothing except consume the spare bytes

76. // Stop when the user inputs a line with just a dot.
77. **while** (strcmp(buf, ".")) { //strcmp returns 0 when the two strings
78. //are the same, so this continues when
79. //they are different
80. // Send the line to the server.
81. **if** (send(socketDescriptor, buf, strlen(buf) + 1, 0) < 0) {
82. cerr << "cannot send data ";
83. close(socketDescriptor); //Note this is just like using files...
84. exit(1);
85. }
87. // Zero out the buffer.
88. memset(buf, 0x0, LINE\_ARRAY\_SIZE);
90. // Read the modified line back from the server.
91. **if** (recv(socketDescriptor, buf, MAX\_LINE, 0) < 0) {
92. cerr << "didn't get response from server?";
93. close(socketDescriptor);
94. exit(1);
95. }
97. cout << "Answer: " << buf << "\n";
99. // Prompt the user for input, then read in the input, up to MAX\_LINE
100. // charactars, and then dispose of the rest of the line, including
101. // the newline character.  As above.
102. cout << "Input: ";
103. cin.get(buf, MAX\_LINE, '\n');
104. **while** (cin.get(c) && c != '\n')
105. ; //Chomp chomp chomp
106. }
108. close(socketDescriptor);
109. **return** 0;
110. }

**tcp-server.cc**

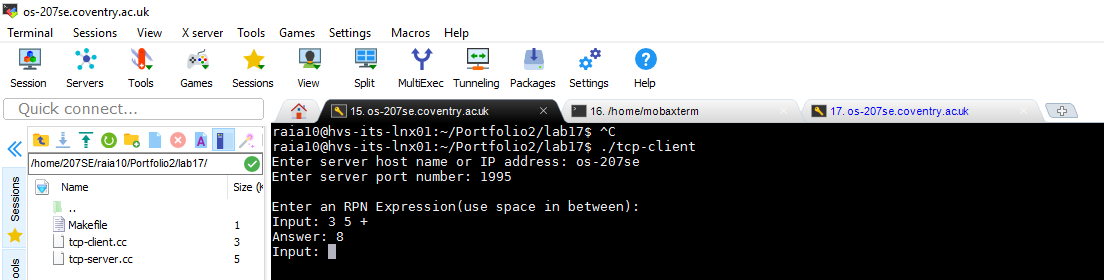
1. #include <arpa/inet.h>
3. #include <netdb.h>
4. #include <netinet/in.h>
5. #include <unistd.h>
6. #include <iostream>
7. #include <cstring>
8. #include <stdlib.h>
9. #include <stdio.h>
11. #define MAX\_MSG 100
12. #define LINE\_ARRAY\_SIZE (MAX\_MSG+1)
14. **using** **namespace** std;
16. //Create stack, variable and
17. //top fo stack variable.
18. **int** stack[MAX\_MSG];
19. string str\_temp;
20. **int** top;
22. //function to push a value.
23. **void** push(**int** x){ stack[top++]=x; }
25. //function to pop a value.
26. **int** pop(){
28. **int** t = stack[--top];
29. stack[top]=0;
30. **return** t;
32. }
34. **int** main()
35. {
36. **int** listenSocket, connectSocket, i;
37. unsigned **short** **int** listenPort;
38. socklen\_t clientAddressLength;
39. **struct** sockaddr\_in clientAddress, serverAddress;
40. **char** line[LINE\_ARRAY\_SIZE];
42. cout << "Enter port number to listen on (between 1500 and 65000): ";
43. cin >> listenPort;
45. // Create socket for listening for client connection
46. // requests.
47. listenSocket = socket(AF\_INET, SOCK\_STREAM, 0);
48. **if** (listenSocket < 0) {
49. cerr << "cannot create listen socket";
50. exit(1);
51. }
53. // Bind listen socket to listen port. First set various
54. // fields in the serverAddress structure, then call
55. // bind().
57. // htonl() and htons() convert long integers and short
58. // integers (respectively) from host byte order (on x86
59. // this is Least Significant Byte first) to network byte
60. // order (Most Significant Byte first).
61. serverAddress.sin\_family = AF\_INET;
62. serverAddress.sin\_addr.s\_addr = htonl(INADDR\_ANY);
63. serverAddress.sin\_port = htons(listenPort);
65. **if** (bind(listenSocket,
66. (**struct** sockaddr \*) &serverAddress,
67. **sizeof**(serverAddress)) < 0) {
68. cerr << "cannot bind socket";
69. exit(1);
70. }
72. // Wait for connections from clients. This is a
73. // non-blocking call; i.e., it registers this program with
74. // the system as expecting connections on this socket, and
75. // then this thread of execution continues on.
76. listen(listenSocket, 5);
78. **while** (1) {
79. cout << "Waiting for TCP connection on port " << listenPort << " ...\n";
81. // Accept a connection with a client that is requesting
82. // one. The accept() call is a blocking call; i.e., this
83. // thread of execution stops until a connection comes
84. // in. connectSocket is a new socket that the system
85. // provides, separate from listenSocket. We \*could\*
86. // accept more connections on listenSocket, before
87. // connectSocket is closed, but this program doesn't do
88. // that.
89. clientAddressLength = **sizeof**(clientAddress);
90. connectSocket = accept(listenSocket,
91. (**struct** sockaddr \*) &clientAddress,
92. &clientAddressLength);
93. **if** (connectSocket < 0) {
94. cerr << "cannot accept connection ";
95. exit(1);
96. }
98. // Show the IP address of the client.
99. // inet\_ntoa() converts an IP address from binary form to the
100. // standard "numbers and dots" notation.
101. cout << "  connected to " << inet\_ntoa(clientAddress.sin\_addr);
103. // Show the client's port number.
104. // ntohs() converts a short int from network byte order (which is
105. // Most Significant Byte first) to host byte order (which on x86,
106. // for example, is Least Significant Byte first).
107. cout << ":" << ntohs(clientAddress.sin\_port) << "\n";
109. // Read lines from socket, using recv(), storing them in the line
110. // array.  If no messages are currently available, recv() blocks
111. // until one arrives.
112. // First set line to all zeroes, so we'll know where the end of
113. // the string is.
114. memset(line, 0x0, LINE\_ARRAY\_SIZE);
115. **while** (recv(connectSocket, line, MAX\_MSG, 0) > 0) {
117. // RPN evaluation loop.
118. **for** (i =0; i<LINE\_ARRAY\_SIZE;i++)
119. {
120. //If number then add it to str\_temp
121. **if**(line[i] >='0' && line[i] <= '9'){
122. str\_temp+=line[i];
123. }
124. // if separator char (SPACE)
125. **else** **if**(line[i] == ' ')
126. {
127. // if str\_temp has a value in it,
128. //  then push and clear the string.
129. **if**(!str\_temp.empty()){
130. push(atoi(str\_temp.c\_str()));
131. str\_temp.clear();}
132. // else go to next char in input string.
133. **else**
134. **continue**;
136. }
137. // if operator
138. **else** **if**(line[i] == '+' || line[i] == '-'
139. ||line[i] == '\*' ||line[i] == '/' ){
141. // pop two value from stack.
142. **int** a = pop();
143. **int** b = pop();
145. // Do operation depending on operator
146. // push result to stack.
147. **switch** (line[i]){
149. **case** '+':
150. push(a+b);
151. **break**;
153. **case** '-' :
154. push(b-a);
155. **break**;
157. **case** '\*':
158. push(a\*b);
159. **break**;
161. **case** '/':
162. push(b/a);
163. **break**;
165. }
167. }
169. }
171. // Only value left in stack is the answer.
172. sprintf(line,"%d",pop());
174. // Send converted line back to client.
175. **if** (send(connectSocket, line, strlen(line) + 1, 0) < 0)
176. cerr << "Error: cannot send modified data";
178. memset(line, 0x0, LINE\_ARRAY\_SIZE);  // set line to all zeroes
179. }
180. }
181. }

Before opening client, I needed to open the server first so the client could connect to the server.



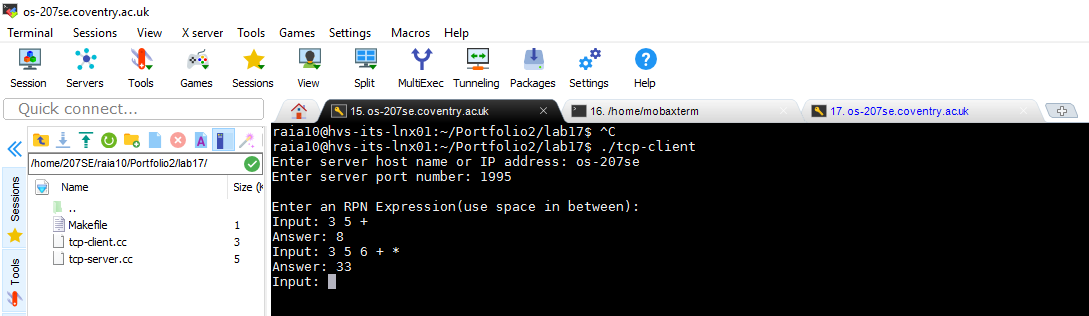
1. Examples of TCP server doing simple RPN calculations (2 numbers 1 arithmetic operation)

[Screenshots showing server doing simple RPN calculations]



1. Examples of TCP server doing more complex RPN calculations with stack or stack-like structure.

[Screenshots showing server doing more complex RPN calculations]



# Evidence

* Source code, with *sensible* comments saying how you client-server code creates a RPN calculator or parser.

* Output of the program showing it solving basic and more complex examples of the reverse polish notation or grammar parser.