# Machine Problem Five:

# The Data Server Moved Out!

Daniel Frazee and Edgardo Angel

Due: May 4, 2015

## Introduction

In Machine Problem Five we expanded from Machine Problem Four, by putting both the client-side request channel and the server-side request channel on two different machines. They communicate with each other along a single TCP connection. For this a modification of the Request Channel is made in order to handle communication between a network. The data server was also modified in order to handle incoming requests over network request channels instead of request channels.

## Procedures

For this assignment to work, we needed to develop the *`NetworkRequestChannel`* class. This would request a request channel that would communicate over a `tcp` connection in order to have a link between the client and the server.

To start off here are the declaration of the constructor and some of its functions. This is from the `NetworkRequestChannel.H` file.

```
/*
Network Request Channel by Daniel Frazee & Edgardo Angel
H File
*/

#ifndef _NETWORKREQUESTCHANNEL_H_
#define _NETWORKREQUESTCHANNEL_H_

#include <iostream>
#include <fstream>
#include <string.h>
#include <string>
#include "semaphore.H"

using namespace std;

class NetworkRequestChannel {

private:

  int fd;

public:

  NetworkRequestChannel(const string _server_host_name, const unsigned short _port_no);
  NetworkRequestChannel(const unsigned short _port_no, void * (*connection_handler) (void *), int backlog);

  ~NetworkRequestChannel();

  string cread();

  int cwrite(string _msg);

  int get_fd();

};

#endif
```

Then we can take a closer look at each function to see how they work. Here are the two constructors used. The first one is used as the constructor for the client-side, which takes as input the host name and the port number..

```
104  // CONSTRUCTOR CLIENT
105 NetworkRequestChannel::NetworkRequestChannel(const string _server_host_name, const unsigned short _port) {
106      stringstream ss;
107      ss << _port;
108      string port = ss.str();
109
110      fd = createClientConnection(_server_host_name.c_str(), port.c_str());
111
112  }
```

This constructor uses another helper function, called `createClientSide`. This function creates the connection using the port number, host name and creates a connection with the socket.

```
27   // CLIENT SIDE
28  int createClientSide(const char * host, const char * port) {
29      struct sockaddr_in socketInput;
30      memset(&socketInput, 0, sizeof(socketInput));
31      socketInput.sin_family = AF_INET;
32
33      // Porting
34      if (struct servent * pse = getservbyname(port, "tcp")) {
35          socketInput.sin_port = pse->s_port;
36      } else if ((socketInput.sin_port = htons((unsigned short)atoi(port))) == 0) {
37          cout << "Can't connect to port " << atoi(port);
38          exit(-1);
39      }
40
41      // Host
42      if (struct hostent * hn = gethostbyname(host)) {
43          memcpy(&socketInput.sin_addr, hn->h_addr, hn->h_length);
44      } else if((socketInput.sin_addr.s_addr = inet_addr(host)) == INADDR_NONE) {
45          cout << "Can't resolve host <" << host << ">";
46          exit(-1);
47      }
48
49      // Socket
50      int s = socket(AF_INET, SOCK_STREAM, 0);
51      if (s < 0) {
52          cout << "Can't establish socket";
53          exit(-1);
54      }
55
56      // Connection
57      if (connect(s, (struct sockaddr *)&socketInput, sizeof(socketInput)) < 0) {
58          cout << "Can't connect to " << host << "::" << port;
59          exit(-1);
60      }
61
62      return s;
63  }
```

The second one is used for the server-side, and take in the port number, the connection handler and the backlog buffer.

```
111
112  // CONSTRUCTOR SERVER
113 NetworkRequestChannel::NetworkRequestChannel(const unsigned short _port, void * (*connection_handler) (void *), int backlog) {
114     stringstream ss;
115     ss << _port;
116     string port = ss.str();
117
118     int master_sock = createServerSide(port.c_str(), backlog);
119     int serverSize = sizeof(serverInput);
120
121     while (true) {
122         int * tempSocket = new int;
123
124         pthread_t thread;
125         pthread_attr_t attr;
126         pthread_attr_init(&attr);
127
128         *tempSocket = accept(master_sock,(struct sockaddr*)&serverInput, (socklen_t*)&serverSize);
129
130         if (tempSocket < 0) {
131             delete tempSocket;
132
133             if (errno == EINTR) continue;
134             else {
135                 cout << "Accept failure!";
136                 exit(-1);
137             }
138         }
139
140         pthread_create(&thread, &attr, connection_handler, (void*)tempSocket);
141
142     }
143     cout << "Connection established";
144 }
```

This constructor uses another helper function, called `createServerSide`. This function maps the port, by binding it to the socket and then being open to a listening mode for communication.

```
64
65  // SERVER SIDE
66 int createServerSide(const char * svc, int backlog) {
67
68     memset(&serverInput, 0, sizeof(serverInput));
69     serverInput.sin_family = AF_INET;
70     serverInput.sin_addr.s_addr = INADDR_ANY;
71
72     // Mapping
73     if (struct servent * pse = getservbyname(svc, "tcp")) {
74         serverInput.sin_port = pse->s_port;
75     } else if ((serverInput.sin_port = htons((unsigned short)atoi(svc))) == 0) {
76         cout << "Can't get service entry";
77         exit(-1);
78     }
79
80     // Binding
81     int snum  = socket(AF_INET, SOCK_STREAM, 0);
82
83     if (snum < 0) {
84         cout << "Can't create socket";
85         exit(-1);
86     }
87
88     if (bind(snum, (struct sockaddr *)&serverInput, sizeof(serverInput)) < 0) {
89         cout << "Can't bind";
90         exit(-1);
91     }
92
93     // Listening
94     if (listen(snum, backlog) < 0) {
95         cout << "Error trying to begin listening";
96         exit(-1);
97     }
98
99     return snum;
100 }
```

The read and write functions are shown below. These are used to read and write that the `NetworkRequestChannel` sends to the dataserver.

```
150
151   // READ
152   string NetworkRequestChannel::cread() {
153       char buf[255];
154
155       if (read(fd, buf, 255) < 0) {
156           perror("Can't read");
157           exit(-1);
158       }
159
160       string s = buf;
161       return s;
162   }
163
164   // WRITE
165   int NetworkRequestChannel::cwrite(string _msg) {
166       if (_msg.length() >= 255) {
167           cout << "Message exceeded 255";
168           return -1;
169       }
170
171       const char * s = _msg.c_str();
172
173       if (write(fd, s, strlen(s)+1) < 0) {
174           perror("Can't write.");
175           exit(-1);
176       }
177   }
178
179   int NetworkRequestChannel::get_fd() {
180       return fd;
181   }
182
```

**Results**

To get the results, the two screenshots below show how to run the program. The first screenshot is the dataserver; this one is run first.

```
[drf6745]@sun ~/GitHub/313/MachineProblems/MachineProblems/MP5> (22:32:41
15)
:: ./dataserver -p 15001 -b 200
```

This second screenshot is the client, and it is run second. Once this one is started, all the interactions begin. These interactions are shown below by the output.

```
[drf6745]@sun ~/GitHub/313/MachineProblems/MachineProblems/MP5> (22:32:37
15)
:: ./client -n 1000 -b 500 -w 40 -p 15001
```

Below is the output for the client. The statistics are displayed similarly to Machine Problem four. Our results were successful, and the statistics below show this. Communication between the server and the client were successful.

```
---Server: localhost

---Port: 15001

Creating Data Server...
Creating Request Buffer...

Creating Response Buffer...

Creating Request Threads...

Creating Request Channels...

Creating Worker Threads...

Creating Statistics Thread...

Creating Histogram...

Statistics for Joe Smith...
0 - 9           90
10 - 19         82
20 - 29         96
30 - 39         99
40 - 49         104
50 - 59         96
60 - 69         95
70 - 79         82
80 - 89         91
90 - 99         60
Total requests: 1000

Statistics for Jane Smith...
0 - 9           96
10 - 19         83
20 - 29         80
30 - 39         88
40 - 49         73
50 - 59         88
60 - 69         100
70 - 79         95
80 - 89         72
90 - 99         79
Total requests: 1000

Statistics for John Doe...
0 - 9           96
10 - 19         98
20 - 29         89
30 - 39         80
40 - 49         99
50 - 59         91
60 - 69         90
70 - 79         88
80 - 89         96
90 - 99         70
Total requests: 1000

Closing Everything...
```

Finally, this last screenshot shows the communication between the client and the dataserver. The requests are displayed as the client asks for them. This is a small portion of them.

```
New request is data Joe Smith
New request is data John Doe
New request is data Joe Smith
New request is data Jane Smith
New request is data Jane Smith
New request is data Joe Smith
New request is data Joe Smith
New request is data John Doe
New request is data John Doe
New request is data Jane Smith
New request is data Joe Smith
New request is data John Doe
New request is data Jane Smith
New request is data Jane Smith
New request is data Jane Smith
New request is data Jane Smith
New request is data John Doe
New request is data Jane Smith
```

**Conclusion**

After running test on our code, we came to see that the change in the number of network request channels and the amount of buffer of the backlog, there was almost no visible change. This could have been due to the fast internet connection used for testing, as the University servers tend to run a higher internet speed that those found locally.