Client Socket

```
11 /* Setup */
12
13 #include <stdio.h>                      // printf() & fprintf()
14 #include <sys/socket.h>                 // socket(), connect(), send(), & recv()
15 #include <arpa/inet.h>                  // sockaddr_in & inet_addr()
16 #include <stdlib.h>                     // atoi() & exit()
17 #include <string.h>                     // memset()
18 #include <unistd.h>                     // close()
19 #include <ctype.h>                      // Check for whitespace characters
20 #include "Practical.h"                  // Contains prototypes for error checking
21
22 #define BUFFSIZE 96                     // Size of receive buffer
23 #define TCP_PORT 48031                  // Default port number
24
25 int main(int argc, char *argv[])
26 {
27   int sock;                             // Socket descriptor
28   struct sockaddr_in servAddr, cliAddr; // Echo server address
29   char *servIP;                         // Server IP address (dotted quad)
30   char sendStr[96];                     // IP and port
31   char buffer[BUFFSIZE];                // Buffer for echo string
32   unsigned int sendStrLen;              // Length of string
33   int bytesRcvd, totalBytesRcvd;        // Bytes read in single recv() & total bytes read
34   socklen_t addrSize = sizeof(cliAddr);
35
36   if ((argc < 3) || (argc > 4))         // Check correct number of arguments
37     DieWithUserMessage("Parameter(2)", "<Server IP> <Request String> [<Server Port>]");
38
39   servIP             = argv[1];         // First arg: server IP address (dotted quad)
40   char* reqStr       = argv[2];         // request string = the 2nd parameter entered
41   in_port_t servPort = (argc == 4) ? atoi(argv[3]) : TCP_PORT; // Port entered or default
42
43   if ((sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)  // Create stream socket, TCP
44     DieWithSystemMessage("socket() failed");
45
46   // Address structure for Server
47   memset(&servAddr, 0, sizeof(servAddr));           // Zero out structure
48   servAddr.sin_family      = AF_INET;               // Internet address family
49   servAddr.sin_addr.s_addr = inet_addr(servIP);     // Server IP address
50   servAddr.sin_port        = htons(servPort);       // Server port
51
52   if (connect(sock, (struct sockaddr *) &servAddr, sizeof(servAddr)) < 0) // Connect to Server
53     DieWithSystemMessage("connect() failed");
```

Figure 1 Client Setup

Client Setup (Figure 1)

L13–24 Necessary header files and define the buffers size and PORT number. Variables used in the Client program
L36 If the number of parameters entered is not correct display an error message, with correct parameters
L39-41 Store the arguments as variables
L43 Create the socket
L47-50 Structure to hold address for Server
L52 Connect to Server

```
55  /* GETSOCKNAME */
56
57  getsockname(sock, (struct sockaddr *) &cliAddr, &addrSize); // Get address and port of local socket
58  fprintf(stderr, "Client Address & port:\t%s:%d\n", inet_ntoa(cliAddr.sin_addr), ntohs(cliAddr.sin_port));
59
60  /* SOCKET COMMUNICATION */
61
62  // SEND - Request String
63  sprintf(sendStr, "%s\r\n", reqStr);                // Add end-of-line marker to request string
64  printf("Sending:\t\t%s\n", reqStr);                // Output the request string
65
66  if (send(sock, sendStr, strlen(sendStr), 0) != strlen(sendStr) || strlen(sendStr) > BUFFSIZE)
67    DieWithSystemMessage("send() sent a different number of bytes than expected");
68  else if(isspace(sendStr[0]))                       // Check if the 1st character is a space
69    DieWithSystemMessage("Syntax error");            // Return a syntax error
70
71  // RECEIVE - Greeting / Error Message From Server
72  if ((bytesRcvd = recv(sock, buffer, BUFFSIZE - 1, 0)) <= 0)
73    DieWithSystemMessage("recv() failed or connection closed prematurely");
74
75  buffer[bytesRcvd] = '\0';                          // Terminate the string!
76  printf("Greeting:\t\t%s", buffer);                 // Greeting + random text
77
78  // Decide if further sending / receiving necessary
79  char msgReceived[96];
80  sprintf(msgReceived, "%s", buffer);
81  char* proceed = strtok (msgReceived, " ,-");
82
83  if(strcmp(msgReceived, "Error") != 0) {
84    // SEND - Num of bytes
85    sprintf(sendStr, "%d/r/n", bytesRcvd);             // End-of-line terminator
86    if (send(sock, sendStr, strlen(sendStr), 0) != strlen(sendStr) || strlen(sendStr) > BUFFSIZE)
87       DieWithSystemMessage("send() sent a different number of bytes than expected");
88    else if(isspace(sendStr[0]))                       // Syntax check
89       DieWithSystemMessage("Syntax error");
90
91    // RECEIVE - Random num of bytes
92    if ((bytesRcvd = recv(sock, buffer, BUFFSIZE - 1, 0)) <= 0)
93      DieWithSystemMessage("recv() failed or connection closed prematurely");
94
95    buffer[bytesRcvd] = '\0';                          // Terminate the string!
96    printf("Random Bytes:\t\t%s", buffer);             // Print the buffer
97
98    int sizeOfBuff = strstr(buffer,"\r\n")-buffer;     // Point in string where searched string "\r\n" begins
99
100   // SEND - Num of bytes
101   printf("Random Bytes Amount:\t%d\n", sizeOfBuff); // size
102   sprintf(sendStr, "%d", sizeOfBuff);                // Create a string to send
103
104   if (send(sock, sendStr, strlen(sendStr), 0) != strlen(sendStr) || strlen(sendStr) > BUFFSIZE)
105     DieWithSystemMessage("send() sent a different number of bytes than expected");
106   else if(isspace(sendStr[0]))                       // Syntax check
107     DieWithSystemMessage("Syntax error");
```

Figure 2 getsockname(), and communication with Server

Getsockname() and Socket Communication (Figure 2)

L57 use getsockname() function to get the local IP Address and Port Number

L63 Append end-of-line terminator to request string for sending

L66-69 Send the request string, if the first character of the string is a space, give syntax error

L72-76 Receive greeting or error from Server, depending on how the request string was formatted, terminate the string, and output it to screen

L79-83 Use the first word of the greeting to decide if the rest of the code is necessary, or skip straight to closing the socket

L85-89 Send the number of bytes received back to the Server

L92-96 Receive random number of bytes from the Request String back from the Server. Terminate the string and output it to screen

98-107 Send the number of bytes received back to the Server

```
109     /* SHUTDOWN SENDING SIDE */
110
111     shutdown(sock, SHUT_WR);                        // Shuts down only the sending side, still receives
112
113     // RECEIVE - Outcome string
114     if ((bytesRcvd = recv(sock, buffer, BUFFSIZE - 1, 0)) <= 0)
115       DieWithSystemMessage("recv() failed or connection closed prematurely");
116     buffer[bytesRcvd] = '\0';                       // Terminate the string!
117     printf("Outcome & Cookie:\t%s\n", buffer);      // Print the  buffer
118   }
119   close(sock);
120   exit(0);
121 } // end main
```

Figure 3 Client shutdown send, receive outcome, and close socket

Shutdown Client Send, Receive Outcome Message, Close Client Socket (Figure 3)

L111 Shutdown the sending side of the Client, as no more data needs to be sent

L114 Receive the outcome string from the server, OK and cookie, or error message and cookie, depending on if the random number of bytes sent by the Server matches up the random number of bytes received by the Client

Server Socket

```
11 /* Setup */
12
13 #include <stdio.h>                // printf() & fprintf()
14 #include <sys/socket.h>           // socket(), connect(), send(), & recv()
15 #include <arpa/inet.h>            // sockaddr_in & inet_addr()
16 #include <stdlib.h>               // random numbers, atoi() & exit()
17 #include <string.h>               // memset()
18 #include <unistd.h>               // Close()
19 #include <ctype.h>                // Check for whitespace characters
20 #include <sys/types.h>            // ssize_t
21 #include <netinet/in.h>           // sockaddr_in
22 #include <time.h>                 // rand numbers seed
23 #include "Practical.h"            // Contains prototypes for error checking
24
25 #define TCP_PORT 48031            // Default Port Number
26
27 int main(int argc, char *argv[]) {//run on command line = "echoSvr <port>";
28                                   // argc = 2 command and parameter-  argv[0] = echoSvr and argv[1] = <port>
29   static const int MAXPENDING = 5;// Maximum outstanding connection requests
30   char msgReceived[96];           // Request string received from client - split
31   char greeting[96];              // Greeting to send to client
32   char msgSplit[96];              // Use to parse request string
33   char* reqStr1, *reqStr2, *reqStr3, *reqStr4, *reqStr5, *gibberish;  // split the request string in separate strings
34   char buffer[BUFSIZE];           // Buffer
35   char outcome[40];               // Send the outcome, OK / Error
36   int servSock;                   // Socket descriptor for server, handles incoming connections
37   ssize_t numBytesSent;           // Number of bytes sent to client
38   ssize_t numBytesRcvd;           // Number of bytes received from client
39   time_t t;                       // For random number generation
40
41   if (argc < 1 || argc > 2)       // Check Arguments
42     DieWithUserMessage("Parameter(s)", "[<Server Port>]");
43
44   in_port_t servPort = (argc == 2) ? atoi(argv[1]) : TCP_PORT;  // Use default port if none entered
45
46   if ((servSock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) // Create socket
47     DieWithSystemMessage("socket() failed");
48
49   // Construct local address structure
50   struct sockaddr_in servAddr;                  // Local address; internet socket address structure
51   memset(&servAddr, 0, sizeof(servAddr));       // Zero out structure
52   servAddr.sin_family = AF_INET;                // IPv4 address family
53   servAddr.sin_addr.s_addr = htonl(INADDR_ANY); // Any incoming interface; host to network long[integer]
54   servAddr.sin_port = htons(servPort);          // Local port; host to network short[integer]
55   socklen_t addrSize = sizeof(servAddr);        // Size of structure to hold address
56
57   // Bind to the local address
58   if (bind(servSock, (struct sockaddr*) &servAddr, sizeof(servAddr)) < 0) // Cast servaddr as generic socket address structure
59     DieWithSystemMessage("bind() failed");
```

Figure 4 Server Setup

Setup The Server (Figure 4)

13-25 Include header files necessary to run the program, and define the default Port Number
29-39 Instantiate the variables required for the Server program to work
40-44 Check the number of arguments, use the default port number if none is entered
46 Create the socket
50-55 Create a structure to hold the Servers address
58-59 Bind the server to the socket

```
61  /* GETSOCKNAME */
62
63  getsockname(servSock, (struct sockaddr *) &servAddr, &addrSize);  // Get address and port of local socket
64  printf("listening on %s:%d\n", inet_ntoa(servAddr.sin_addr), servPort);
65
66  if (listen(servSock, MAXPENDING) < 0)       // Socket listens for incoming connections
67    DieWithSystemMessage("listen() failed");
68
69  /* LOOP */
70
71  for (;;) {                                  // Infinite loop
72    int proceed = 0;                          // Decides if rest of sends / recvs are necessary
73    struct sockaddr_in clntAddr;              // Client address structure
74    socklen_t clntAddrLen = sizeof(clntAddr); // Set length of client address structure (in-out parameter)
75
76    // Wait for a client to connect
77    int clntSock = accept(servSock, (struct sockaddr *) &clntAddr, &clntAddrLen);
78    if (clntSock < 0)
79      DieWithSystemMessage("accept() failed");
80
81    // clntSock is connected to a client!
82    char clntName[INET_ADDRSTRLEN];           // String to contain client address
83    if (inet_ntop(AF_INET, &clntAddr.sin_addr.s_addr, clntName, sizeof(clntName)) != NULL){
84      printf("NEW CONNECTION\n\n");
85      printf("Connected to: %s-%d, \n", clntName, ntohs(clntAddr.sin_port)); // client address / port
86    } else
87      puts("Unable to get client address");
```

Figure 5 getsockname() Server IP Address and Port, Loop for communication with Client sockets

Getsockname() and Server Loop to listen and process client connections (Figure 5)

63 Get the local or Server IP Address and Port Number

66 Check the maximum number of connections is not reached

71 Infinite loop to handle connections

72 Decision variable, decides if code after greeting message is processed is used

77 accept the connection from the client, and store the client address, print the IP Address and Port Number the Client is using

```
/* RECEIVE - Request String & Parse It */

numBytesRcvd = recv(clntSock, buffer, BUFSIZE - 1, 0);
if (numBytesRcvd < 0)
  DieWithSystemMessage("recv() failed");
buffer[numBytesRcvd] = '\0';            // Terminate string
printf("Request String: %s\n", buffer);
printf("Number of bytes received: %lu\n", numBytesRcvd);

/* PARSE - Request String */

sprintf(msgReceived, "%s", buffer);     // copy buffer to another string to split
char* server      = "netsvr";           // Stored server name to compare
char* requestType = "type0";            // Stored request type to compare
char* username    = "jaoregan";         // Stored username to compare
char port[20];
sprintf(port, "%d", ntohs(servAddr.sin_port)); // Server Port from input or stored default

sprintf(msgSplit, "%s", msgReceived);
reqStr1 = strtok (msgReceived, " ,-");  // Server
reqStr2 = strtok (NULL, " ,-");         // Request type
reqStr3 = strtok (NULL, " ,-");         // Username
reqStr4 = strtok (NULL, " ,-");         // IP Address
reqStr5 = strtok (NULL, "\r\n,-");      // Port,
gibberish = strtok (NULL, " ,-");       // Take any gibberish at end of string

// Check format and set greeting to send
printf("PARSED REQUEST STRING:\n");
if(strcmp(reqStr1, server) == 0 && strcmp(reqStr2, requestType) == 0 && strcmp(reqStr3, username) == 0
&& strcmp(reqStr4, clntName) == 0 && strcmp(reqStr5, port) == 0) {
  printf("Server:\t\t%s\n", server);
  printf("Request Type:\t%s\n", requestType);
  printf("Username:\t%s\n", username);
  printf("IP Address:\t%s\n", clntName);
  printf("Port:\t\t%s\n\n", port);

  sprintf(greeting, "Hello %s-%d, welcome to the netsvr server\r\n", clntName, ntohs(clntAddr.sin_port)); // client address / port
}
else {
  printf("Request string incorrectly formatted\n");

  sprintf(greeting, "Error not formatted right!\r\n"); // Request String not correct

  // Check input against store values
  printf("reqStr1: %s Vs %s\n", reqStr1, server);
  printf("reqStr2: %s Vs %s\n", reqStr2, requestType);
  printf("reqStr3: %s Vs %s\n", reqStr3, username);
  printf("reqStr4: %s Vs %s\n", reqStr4, clntName);
  printf("reqStr5: %s Vs %s\n", reqStr5, port);

  proceed = 1;                              // No need for communciation after greeting
}

printf("Greeting: %s\n", greeting);         // Display the greeting
```

Figure 6 Parse the Request String

```
/* SOCKET COMMUNICATION */

// SEND - Greeting
numBytesSent = send(clntSock, greeting, strlen(greeting), 0); // Send the greeting
if(isspace(greeting[0]))                        // Check for space at beginning of a line
  DieWithSystemMessage("Syntax error");
//if (strncmp(greeting, " ", 1) == 0)           // Old version, check for space
//  DieWithSystemMessage("Syntax error");
else if (numBytesSent < 0)
  DieWithSystemMessage("send() failed");
else if (numBytesSent != strlen(greeting))
  DieWithUserMessage("send()", "sent unexpected number of bytes");
```

Figure 7 Send greeting to Client

```c
// CONDITIONAL
    if(proceed == 0){   // Continue if the greeting is positive
    printf("Bytes sent: %lu\n", numBytesSent);    // Number of bytes in the greeting

      // RECEIVE - Num of bytes
      numBytesRcvd = recv(clntSock, buffer, strlen(buffer), 0);
      if (numBytesRcvd < 0)
        DieWithSystemMessage("recv() failed");

      // SEND - Random number of bytes
      srand((unsigned) time(&t));             // initialise random number
      int randomBytes = (rand() % 50) + 1;    // random number between 1 and 50
      printf("Random Bytes to send: %d\n", randomBytes);
      char message2[randomBytes];
      strncpy(message2, greeting,randomBytes);// Send back random bytes of request string
      strcat(message2,"\r\n");                // Ad end-of-line terminator

      numBytesSent = send(clntSock, message2, strlen(message2), 0);
      if(isspace(message2[0]))                // Check 1st character for whitespace
        DieWithSystemMessage("Syntax error");

      // RECEIVE - Num of bytes
      numBytesRcvd = recv(clntSock, buffer, BUFSIZE - 1, 0);
      if (numBytesRcvd < 0)
        DieWithSystemMessage("recv() failed");
      buffer[numBytesRcvd] = '\0';            // Terminate the string
      printf("Size of data received by client: %s \n", buffer);
      int clientReturnValue = atoi(buffer);   // Convert bytes client received to integer
      printf("Client bytes returned: %d\n", clientReturnValue); // Num bytes received by client

      /* COOKIE */

      // SEND - Outcome / Cookie
      srand((unsigned) time(&t));             // initialise random number
      int cookie = rand() % 1000000000;       // random number between 0 and 999999999

      /* Decide Outcome */

      if (clientReturnValue == randomBytes)   // If the bytes are equal
        sprintf(outcome, "OK %d", cookie);    // amounts match
      else                                    // amounts don't match
        sprintf(outcome, "Returned value different %d", cookie);

      numBytesSent = send(clntSock, outcome, strlen(outcome), 0); // Send outcome to client
      if(isspace(outcome[0]))                 // Check 1st character for whitespace
        DieWithSystemMessage("Syntax error");
      else if (numBytesSent < 0)
        DieWithSystemMessage("send() failed");
      printf("Outcome and Cookie: %s\n\n", outcome);
    }
    close(clntSock);                          // Close client socket
  }                                           // end of infinite for loop
}
```

Figure 8 Conditional socket communication, Random Bytes, Cookie, and Client socket close

```
joe@joemsi: ~
joe@joemsi:~$ ./cli 127.0.0.1 "netsvr type0 jaoregan 127.0.0.1-48031"
Client Address & port:  127.0.0.1:34656
Sending:                netsvr type0 jaoregan 127.0.0.1-48031
Greeting:               Hello 127.0.0.1-34656, welcome to the netsvr server
Random Bytes:           Hello 127.0.0.1-34656, welcome to the netsvr
Random Bytes Amount:    44
Outcome & Cookie:       OK 200172043
joe@joemsi:~$ ./cli 127.0.0.1 "netsvr TYPE8 jaoregan 127.0.0.1-48031"
Client Address & port:  127.0.0.1:34658
Sending:                netsvr TYPE8 jaoregan 127.0.0.1-48031
Greeting:               Error not formatted right!
joe@joemsi:~$
```

Figure 9 Client sending correct and incorrect request strings, and Server response

```
joe@joemsi: ~
joe@joemsi:~$ ./srv
listening on 0.0.0.0:48031
NEW CONNECTION

Connected to: 127.0.0.1-34656,
Request String: netsvr type0 jaoregan 127.0.0.1-48031

Number of bytes received: 39
PARSED REQUEST STRING:
Server:         netsvr
Request Type:   type0
Username:       jaoregan
IP Address:     127.0.0.1
Port:           48031

Greeting: Hello 127.0.0.1-34656, welcome to the netsvr server

Bytes sent: 53
Random Bytes to send: 44
Size of data received by client: 44
Client bytes returned: 44
Outcome and Cookie: OK 200172043

NEW CONNECTION

Connected to: 127.0.0.1-34658,
Request String: netsvr TYPE8 jaoregan 127.0.0.1-48031

Number of bytes received: 39
PARSED REQUEST STRING:
Request string incorrectly formatted
Greeting: Error not formatted right!
```

Figure 10 Server receives request strings from Client and decides if it is to send a greeting or an error

```
joe@joemsi: ~
joe@joemsi:~$ ./cli 127.0.0.1 "netsvr type0 jaoregan 127.0.0.1-48031"
Client Address & port:  127.0.0.1:35122
Sending:                netsvr type0 jaoregan 127.0.0.1-48031
Greeting:               Hello 127.0.0.1-35122, welcome to the netsvr server
Random Bytes:           Hello 127.0.0.
Random Bytes Amount:    14
Outcome & Cookie:       OK 837035513
joe@joemsi:~$ ./cli 127.0.0.1 "netsvr type0 jaoregan 127.0.0.1-48031"
Client Address & port:  127.0.0.1:35124
Sending:                netsvr type0 jaoregan 127.0.0.1-48031
Greeting:               Hello 127.0.0.1-35124, welcome to the netsvr server
Random Bytes:           Hello 127.0.0.1-3◆◆
Random Bytes Amount:    22
Outcome & Cookie:       Returned value different 494216966
joe@joemsi:~$
```

Figure 11 Client outcome message received for correct bytes, and incorrect bytes

```
joe@joemsi: ~
joe@joemsi:~$ ./srv
listening on 0.0.0.0:48031
NEW CONNECTION

Connected to: 127.0.0.1-35122,
Request String: netsvr type0 jaoregan 127.0.0.1-48031

Number of bytes received: 39
PARSED REQUEST STRING:
Server:         netsvr
Request Type:   type0
Username:       jaoregan
IP Address:     127.0.0.1
Port:           48031

Greeting: Hello 127.0.0.1-35122, welcome to the netsvr server

Bytes sent: 53
Random Bytes to send: 14
Size of data received by client: 14
Client bytes returned: 14
Outcome and Cookie: OK 837035513

NEW CONNECTION

Connected to: 127.0.0.1-35124,
Request String: netsvr type0 jaoregan 127.0.0.1-48031

Number of bytes received: 39
PARSED REQUEST STRING:
Server:         netsvr
Request Type:   type0
Username:       jaoregan
IP Address:     127.0.0.1
Port:           48031

Greeting: Hello 127.0.0.1-35124, welcome to the netsvr server

Bytes sent: 53
Random Bytes to send: 17
Size of data received by client: 22
Client bytes returned: 22
Outcome and Cookie: Returned value different 494216966
```

Figure 12 Server outcome message for correct and incorrect random bytes received by Client