Farshad Chowdhury

Assignment 4: Client Server Communication

December 11, 2017

Objective

The objective of this assignment was to introduce students to client server communication using sockets. Client server communications are not only extremely important in network communications but in inter-process communications as well. This allows different processes running on an OS to communicate to manage shared data. This assignment specifically focuses on sending a message over a TCP socket from one machine to another on the anaconda network. This assignment also utilizes other concepts that we have learned throughout this semester, mainly multithreading. Multithreading, in this case, allows the program to listen for data on a socket, while having a timeout function that retries sending the message in case the server does not respond.

Background

As previously mentioned this assignment uses the client server model for communication over the anaconda network at UML. In this implementation the server is constantly listening over a socket defined by the last 4 digits of the student's ID + 2000. This ensures that the socket being used is unique to each student to prevent stray pings. The server first begins by creating a socket using the *int socket(int domain, int type, int protocol)*; function. The function returns a -1 if an error is encountered while creating the socket. After defining the port using sa.sin_port(int port) for the socket address struct, the port and address are binded by using the *int bind(int socket, const struct sockaddr *address, socklen_t address_len);* function. The server then starts to listen for any packets being sent over the socket. Any messages sent over the socket are stored in a character array msg using the recvfrom function. Then using strcmp and a series of if and else statements, the server executes the appropriate function based on the message that is sent over the socket, ensuring to close the socket each time. The server, in every case, sends back a message to the client to acknowledge (ack)is request. This code is looped in a while(1) loop so that once the server executes a function it returns to listening for a message.

The client in this implementation is actually a multithreaded program. This is due to the timeout functionality added to the client. If the client pings the server and receives no response, the client will wait a short while and attempt to ping the server again, repeating for a maximum of three times. A separate thread had to be used in this implementation because when the client waits for the message to be received from the server it halts. BEcause of this a seperate thread is used to handle the timeout. The client first starts by initializing all of the pthread parameters needed to create a thread. The client then prompts the user to enter a message to be sent to the server. A new thread is created in a while loop that exits when a message is received or if the message is sent three times. The thread created runs the sendMSG function which handles sending the message to the server. The msg entered by the user is passed to this thread. Once the thread is created, the main program sleeps for 3 seconds and then checks the global variable received, which is the character array in which the ack from the server is stored. If the
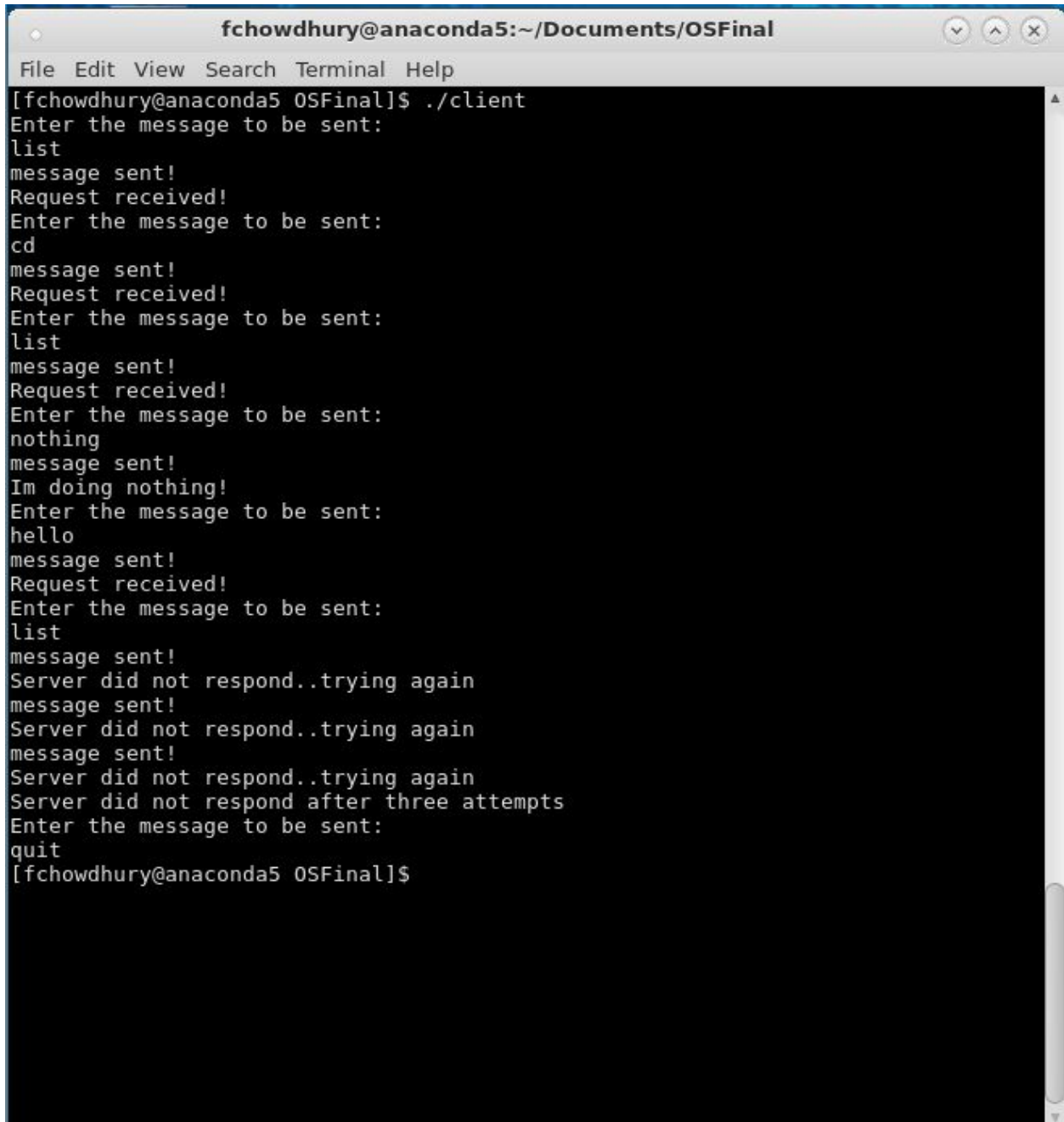
character array still contains Empty, the server has not sent back a message, meaning it is either busy or down. If the character array is empty the sendMSG thread is cancelled while the loopcheck variable is incremented. If received contains a message from the server loopcheck is set to 5 which causes the loop to exit. The sendMSG works very similarly to the server. The first the host address of the server is defined. In this case anaconda3.uml.edu. Next the port is defined as the same port as the server. The socket is then created using the same parameters as in the server. The client finally sends the message passed into the function over the socket created using the sendto function. The client then immediately waits for a response over the socket using the recvfrom function. The received message is is stored in the character array received. If no response is received the character array does not change prompting the program to retry sending the message.

Algorithms Used

This assignment did not really make use of any popular algorithms. This program however did draw from a couple concepts that we have learned throughout the semester. This assignment made use of multithreading to handle possible server errors on the client side.
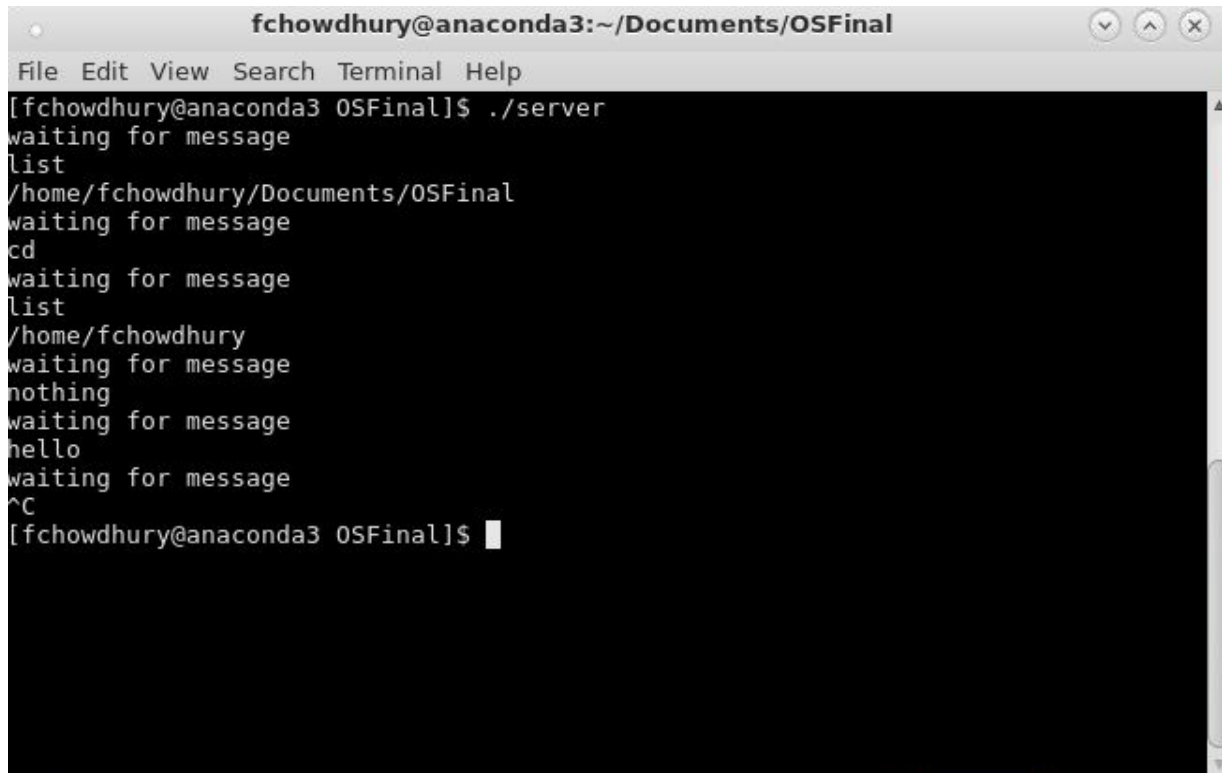
It can be noted that although this is a multi threaded program, no synchronization was used or needed for this program to function properly. This is because only one variable is being shared between the threads, and while the send message function sets the value of received, the main program is always sleeping ensuring the variable is not changed by accident. In addition if the transaction takes over 3 seconds, which is how long the main thread sleeps, the server has timed out anyways.

Results


```
fchowdhury@anaconda5:~/Documents/OSFinal

File  Edit  View  Search  Terminal  Help
[fchowdhury@anaconda5 OSFinal]$ ./client
Enter the message to be sent:
list
message sent!
Request received!
Enter the message to be sent:
cd
message sent!
Request received!
Enter the message to be sent:
list
message sent!
Request received!
Enter the message to be sent:
nothing
message sent!
Im doing nothing!
Enter the message to be sent:
hello
message sent!
Request received!
Enter the message to be sent:
list
message sent!
Server did not respond..trying again
message sent!
Server did not respond..trying again
message sent!
Server did not respond..trying again
Server did not respond after three attempts
Enter the message to be sent:
quit
[fchowdhury@anaconda5 OSFinal]$
```

Terminal Output for Client

Terminal Output for Server

Observations

The code ran as expected. The client was run on anaconda 5 while the server was run on anaconda 3. When a message was sent through the client, the server immediately receives the message and executes the appropriate action as seen above. To test the client timeout the server is killed using control C, while the client sends a message. As can be seen above the client waits for a response from the server, and when it does not receive a response it tries to ping the server 3 times before prompting the user to enter another message.

Conclusions

As mentioned above the code ran as expected. This assignment combined the use of some different concepts we have learned throughout the semester. In doing so the program gave students valuable practice in using multithreading to accomplish different tasks as well as using socket based client server communication to communicate with machines on a network or different processes for inter-process communication.

Source Code

Client.C

```c
1.  /*
2.  Operating Systems and Kernel Design
3.  Assignment 4
4.  Client and Server Communication
5.  Farshad Chowdhury
6.  December 11, 2017
7.  Client.c
8.  This program sends a message to a server and times out if the server
9.  does not respond in time.
10.
11. */
12.
13.
14.
15. #include<sys/types.h>
16. #include<sys/socket.h>
17. #include<netdb.h>
18. #include<netinet/in.h>
19. #include<stdio.h>
20. #include<stdlib.h>
21. #include<sys/socket.h>
22. #include<sys/time.h>
23. #include<errno.h>
24. #include<arpa/inet.h>
25. #include<string.h>
26. #include <pthread.h>
27. #include <semaphore.h>
28. #define RECEIVER_HOST "anaconda3.uml.edu" /* Server machine */
29. /* Declaring errno */
30. extern int errno;
31. char received[50]="Empty";
32. int BUFSIZE = 50;
33. void sendMSG(char *msg);
34.
35. /* Function for error */
36. void report_error(char *s)
37. {
38.  printf("sender: error in %s, errno = %d\n",s,errno);
39.  exit(1);
40. }
41. void main(int argc, char *argv[])
42. {
43.    int lpchk=1;
44.    while(lpchk==1){
45.       char msg[BUFSIZE];
46.       int loopchk=0;
47.       strcpy(received,"Empty");
48.       printf("Enter the message to be sent: \n"); //User prompted for message
49.       scanf("%s",msg);
50.       if(strcmp(msg,"quit")==0){
```

```
51.        lpchk= 0;
52.        exit(1);
53.      }
54.     pthread_t tid;
55.     pthread_attr_t attr;
56.     pthread_attr_init(&attr);
57.     while(loopchk<3){ //loop used to retry message sending
58.       pthread_create(&tid,&attr,sendMSG,msg); // thread created to send message
59.       sleep(3);// main thread sleeps for three seconds before checking array
60.       if(strcmp(received,"Empty")==0){
61.         printf("Server did not respond..trying again\n");
62.         pthread_cancel(tid);
63.         loopchk++;
64.       }else{
65.         loopchk=5;
66.       }
67.     }
68.   }
69. }
70. void sendMSG(char *msg){
71.
72.   int s,i;
73.
74.   struct sockaddr_in sa= {0};
75.   int length = sizeof(sa);
76.   struct hostent *hp;
77.
78.   /* FILL SOCKET ADDRESS*/
79.   if((hp = gethostbyname(RECEIVER_HOST))==NULL)
80.   report_error("gethostbyname");
81.   bcopy((char*)hp->h_addr, (char *)&sa.sin_addr, hp->h_length);
82.   sa.sin_family = hp->h_addrtype;
83.   sa.sin_port = htons(0213 + 20000); /* define port
84.   number based on student ID*/
85.   /* Creating the socket and returns error if unsuccessfull */
86.   if((s=socket(AF_INET, SOCK_DGRAM, PF_UNSPEC))== -1)
87.   report_error("socket");
88.   /* Sending the message to server and returns error if unsuccesfull */
89.   if(sendto(s, msg, BUFSIZE, 0, (struct sockaddr *) &sa, length)== -1)
90.   report_error("sendto");
91.   printf("message sent!\n");
92.   /* Receives message from server and returns error if unsuccesfull */
93.   recvfrom(s, received, BUFSIZE, 0, (struct sockaddr *) &sa, &length);
94.   printf("%s\n",received);
95.   close(s);
96.
97. }
```

Server.c

```
1. /*
```

```
2.   Operating Systems and Kernel Design
3.   Assignment 4
4.   Client and Server Communication
5.   Farshad Chowdhury
6.   December 11, 2017
7.   Server.c
8.   This program handles messages sent by a client. It can execute a number
9.   of different actions depenedent on the message sent by the client.
10. */
11.
12.
13. #include<sys/socket.h>
14. #include<sys/types.h>
15. #include<netdb.h>
16. #include<netinet/in.h>
17. #include<stdio.h>
18. #include<stdlib.h>
19. #include<errno.h>
20. #include<strings.h>
21. #include<string.h>
22. #define RECEIVER_HOST "anaconda3.uml.edu" /* Server machine */
23. /* Declaring errno */
24. extern int errno;
25. /* Function for printing error */
26. void report_error(char *s)
27. {
28.  printf("receiver: error in%s, errno = %d\n", s, errno);
29.  exit(1);
30. }
31. /* Dynamically giving the 'size' of message as argument */
32. void main(int argc, char *argv[])
33. {
34.   while(1){
35.     int size=50;
36.     int s;
37.     char m[200]="Request received!";
38.     char test[200]="list";
39.     char response[size];
40.     char msg[size];
41.     struct sockaddr_in sa = {0}, r_sa = {0};
42.     int r_sa_l = sizeof(r_sa);
43.     int len;
44.     int backlog = 5;
45.     struct hostent *hp;
46.     socklen_t length;
47.     printf("waiting for message\n");
48.     strcpy(response,m);
49.     /* Creating the socket and returns error if unsuccesfull */
50.
51.     if((s= socket(AF_INET, SOCK_DGRAM, PF_UNSPEC)) == -1)
```

```c
52.        report_error("socket");
53.
54.        sa.sin_family = AF_INET;
55.        sa.sin_addr.s_addr=INADDR_ANY;
56.        sa.sin_port = htons(0213+ 20000); /* define port
57.        number based on student ID*/
58.
59.        /* Binding the socket and returns error if unsuccesfull */
60.
61.        if(bind(s, (struct sockaddr *)&sa, sizeof(sa))== -1)
62.        report_error("bind");
63.
64.        listen(s, 10);
65.        length = sizeof(r_sa);
66.        /* Receiving message from client and returns error if unsuccessfull */
67.
68.        if((len = recvfrom(s, msg, size, 0, (struct sockaddr *)&r_sa, &r_sa_l))== -1)
69.        report_error("recvfrom");
70.        printf("%s\n",msg);
71.        //series of if and else statements to execute different function based on client
      message
72.        if (strcmp(msg,"list")==0){
73.          system("pwd");
74.          sendto(s,response,size,0,(struct sockaddr *)&r_sa,r_sa_l);
75.          close(s);
76.        }
77.        else if(strcmp(msg,"quit")==0){
78.          char quiting[50]="Exiting Now";
79.          sendto(s,quiting,size,0,(struct sockaddr *)&r_sa,r_sa_l);
80.          exit(1);
81.
82.        }
83.        else if(strcmp(msg,"cd")==0){
84.          chdir("/home/fchowdhury");
85.          sendto(s,response,size,0,(struct sockaddr *)&r_sa,r_sa_l);
86.          close(s);
87.        }else if(strcmp(msg,"nothing")==0){
88.          char nothing[50]="Im doing nothing!";
89.          sendto(s,nothing,size,0,(struct sockaddr *)&r_sa,r_sa_l);
90.          close(s);
91.        }
92.        else{
93.          sendto(s,response,size,0,(struct sockaddr *)&r_sa,r_sa_l);
94.          close(s);
95.        }
96.    }
97. }
```