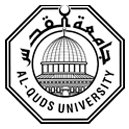
بسم الله الرحمن الرحيم



Faculty of Engineering

Computer Engineering Department

**Computer Network Project**

Time Server-Client over Socket in C

By:

**Haitham Mohammad Ajaj 21410520**

**Muhannad Oqbah Taha 21620447**

Supervisor**:**

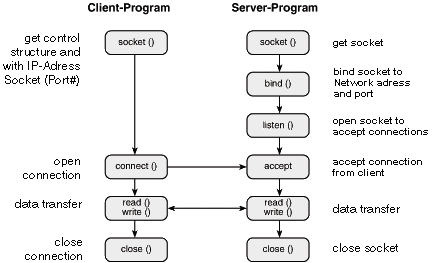
**Dr. Rushdi Hamamreh**

**Table of content:**

|  |  |
| --- | --- |
| Subject | Page Number |
| Introduction | 3 |
| Discussion | 4 |
| Objectives | 4 |
| Procedure | 5 |
| Conclusion | 10 |
| Appendix(Code) | 11 |

**Introduction:**

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket (node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

s

In this project, we have used C language to build our program which will do the task of sending and receiving the data and time between the server and the client, sending and receiving time is a very important process when we connecting multiple computer with each other to keep synchronization between all nodes of the network.

**Discussion:**

In this project, we have used C language to build the program because C is a primitive language and it include low-level access to memory, simple set of keywords, and clean style, and it is very fast in terms of execution time.

We have use Linux operating system and the program were compiled using GCC compiler which were installed on the operating system, the code will run on the virtual machine which installed on the virtual box.

This basic project demonstrate process of creating communication line between the server and the client using some required libraries that contain the functions, structures that we need for socket programming and for getting the time, and convert it to string before sending it.

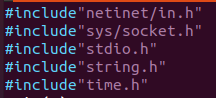
**Objectives:**

1. To create the code for the server that will send the time.
2. To create code for the client that will receive the time.
3. To learn the fundamentals of creating TCP-based applications.
4. To learn how to use time and string libraries.

**Procedure:**

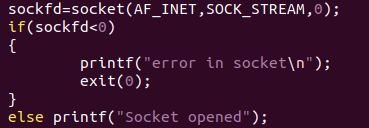
**Server side**

First, we have included the required libraries for socket programming such as:



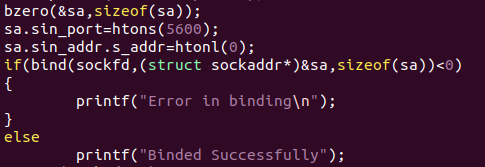
* Netinet/in.h: define the data types and structures for the addresses used in communication.
* Sys/socket.h: define the functions to set the socket and connection and sending and receiving data and the constants needed by these functions.
* Time.h: define the functions used to retrieve date and time.

In the next section of code we have used the *socket()* function to create a new communication endpoint.



* sockfd: socket descriptor, an integer (like a file-handle).
* AF\_INET: Address family, IPv4 protocols, Internet addresses (typically used).
* SOCK\_STREAM: Communication type, TCP, reliable, 2-way, connection-based service.
* 0: protocol, usually set to zero to use the default protocol.

The next step to define the addresses that we will use to communicate over socket but before that we have initializes the address structure to zero by using *bzero()* function then give the port address(5600) and the ip address (0) to use the local address.



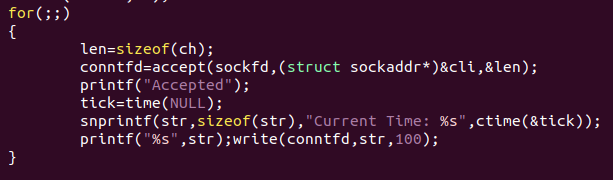
Then we have to bind the socket to the specified addresses using the *bind()* function which takes the socket describer as the first parameter and the address structure after typecasting it to *sockaddr* type as the second parameter, and the size of the address as the third parameter, this function will return a value of -1 if there is an error in the binding process.

After that we have used the *listen()* function to listen to the incoming requests to connection.



This function takes the socket descriptor as the first parameter and the queue length, which is number of active participants that can “wait” for a connection and in out code we have set it to 50 participants, this function return a value of -1 if there is an error.

After listening for the connection requests we to accept one of the them by using the *accept()* function, which creates new socket for transfer data.



* conntfd: socket descriptor, the new socket (used for data-transfer).
* sockfd: socket descriptor, the original socket (being listened on).
* clir: struct sockaddr, address of the active participant (filled in upon return).
* Len: size of client address (active participant).

After creating a new socket we will get the time from *time()* function which return the time as the number of seconds from 1 January 1971 till the current moment and store this value in *tick* variable, then we will use the function *ctime()* to convert the time to the format **Www Mmm dd hh:mm:ss yyyy** Which means as follows:

Www: Day of week. Mmm: Month name. dd : Day of month. hh : Hour digit. mm : Minute digit. ss : Second digit. yyyy : Year digit.

Then we have copied the time string to the buffer that we have defined before using *snprintf()* function, then we will send the time to the client by using *write()* function that three parameters:

* Conntfd: the socket for data transfer.
* Str: the string that we will send to client.
* 100: the size of the array of characters (message).

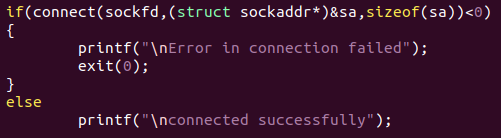
We put this code in loop to allow the clients request the time and date for unlimited times.

**Client side:**

From the client side, the code will look similar the server side except that the client has to use the *connect()* function to initialize the connection because the client is the active participant and the server is the passive participant, this function takes three parameters:

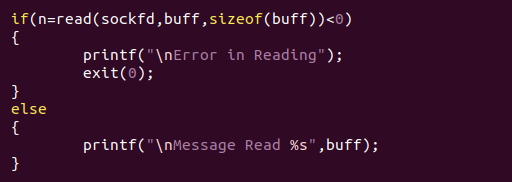
* sockfd: integer, socket to be used in connection that defined in the client code.
* sa: struct sockaddr: address of the passive participant which is the server.
* Size of the server address.

This function will return a value of -1 if there is an error.



After the connection established and the server has sent the message, the client need to read this message by calling the *read()* function this function takes three parameters:

* sockfd: integer, socket to be used in connection that defined in the client code.
* Buff: the array of characters that stores received bytes.
* The size of the received message.



**The output:**

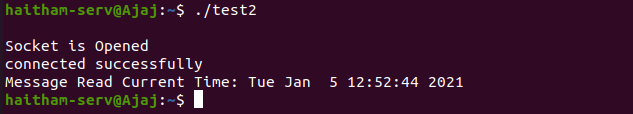
Server:





Client:





**Conclusion:**

After doing this project we have learned how made network-based programs using socket programming in C especially TCP sockets using Linux platform and how this process is done from server and client perspectives, and we have become familiar with time function and converting it to the appropriate format before sending it.

The full code will be attached in the end of this report and as a separate file in the Email.

**Server**:

#include"netinet/in.h"

#include"sys/socket.h"

#include"stdio.h"

#include"string.h"

#include"time.h"

main( )

{

struct sockaddr\_in sa;

struct sockaddr\_in cli;

int sockfd,conntfd;

int len,ch;

char str[100];

time\_t tick;

sockfd=socket(AF\_INET,SOCK\_STREAM,0);

if(sockfd<0)

{

printf("error in socket\n");

exit(0);

}

else printf("Socket opened");

bzero(&sa,sizeof(sa));

sa.sin\_port=htons(5600);

sa.sin\_addr.s\_addr=htonl(0);

if(bind(sockfd,(struct sockaddr\*)&sa,sizeof(sa))<0)

{

printf("Error in binding\n");

}

else

printf("Binded Successfully");

listen(sockfd,50);

for(;;)

{

len=sizeof(ch);

conntfd=accept(sockfd,(struct sockaddr\*)&cli,&len);

printf("Accepted");

tick=time(NULL);

snprintf(str,sizeof(str),"Current Time: %s",ctime(&tick));

printf("%s",str);write(conntfd,str,100);

}

}

**Client:**

#include"netinet/in.h"

#include"sys/socket.h"

#include"stdio.h"

main()

{

struct sockaddr\_in sa,cli;

int n,sockfd;

int len;char buff[100];

sockfd=socket(AF\_INET,SOCK\_STREAM,0);

if(sockfd<0){ printf("\nError in Socket");

exit(0);

}

else

printf("\nSocket is Opened");

bzero(&sa,sizeof(sa));

sa.sin\_family=AF\_INET;

sa.sin\_port=htons(5600);

if(connect(sockfd,(struct sockaddr\*)&sa,sizeof(sa))<0)

{

printf("\nError in connection failed");

exit(0);

}

else

printf("\nconnected successfully");

if(n=read(sockfd,buff,sizeof(buff))<0)

{

printf("\nError in Reading");

exit(0);

}

else

{

printf("\nMessage Read %s",buff);

}

}