**CSCI 367 - Computer Networks I**

TCP/IP Sockets – Echo Client and Server – TCP Transport Layer Protocol

**Source Code Listing**

See 02\_EchoClientServer\_TCP.

**State of Socket**

netstat -na | grep "1234"

**Compilation**

gcc -g -o Program Program.c

The client and server programs shown below use sockets to communicate with each other.

The client program uses a socket to connect to a server program on a remote networked computer. The client then sends a message to the server, and the server echo’s the message back to the client. Following are two diagrams that illustrate the client-server communication cycle. First, the client sends a message to the server:

127.0.0.1



127.0.01



2025

5040

Client Computer

Server Computer

Server Port

Client Port

Then the server returns echo’s back the same message:

127.0.0.1



127.0.01



2025

5040

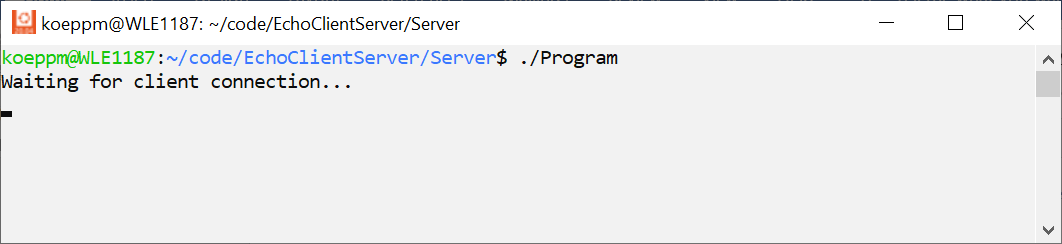
Client Computer

Server Computer

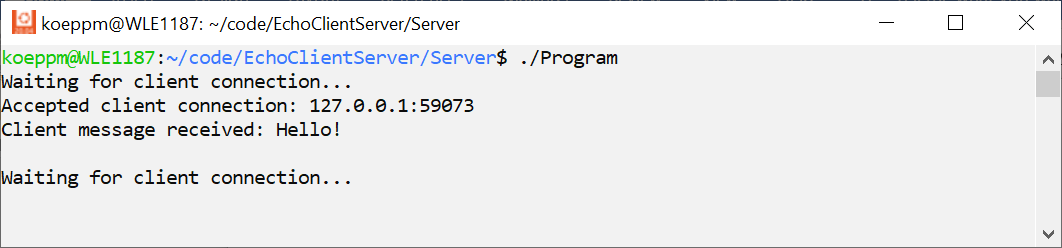
Server Port

Client Port

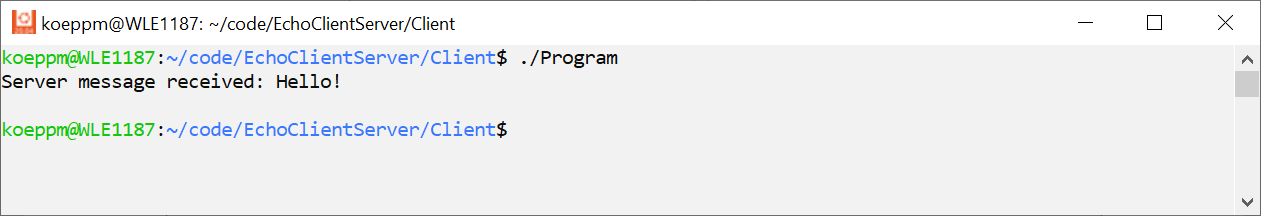
When I run the server program, the server waits for a connection from the client:



When a client connects, a connection is established. The client sends a message, and the server produces the following output, which displays the clients IP address and port number and the client message.



The client produces the following output, which displays the message sent back from the server.



**TCP send/recv vs write/read**

<https://man7.org/linux/man-pages/man2/send.2.html>

The send() call may be used only when the socket is in a *connected* state (so that the intended recipient is known). The only difference between send() and [write(2)](https://man7.org/linux/man-pages/man2/write.2.html) is the presence of *flags*. With a zero *flags* argument, send() is equivalent to [write(2)](https://man7.org/linux/man-pages/man2/write.2.html).

<https://man7.org/linux/man-pages/man2/recv.2.html>

The only difference between recv() and read(2) is the presence of flags. With a zero flags argument, recv() is generally equivalent to read(2) (but see NOTES).

**Type and Function Specifications**

<https://man7.org/linux/man-pages/man2/socket.2.html>

<https://www.sciencedirect.com/topics/computer-science/socket-descriptor>

**#include <sys/socket.h>**

**int socket(int** *domain***, int** *type***, int** *protocol***);**

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

<https://man7.org/linux/man-pages/man7/ip.7.html>

struct sockaddr\_in server;

<https://man7.org/linux/man-pages/man2/bind.2.html>

**#include <sys/socket.h>**

**int bind(int** *sockfd***, const struct sockaddr \****addr***,** **socklen\_t** *addrlen***);**

int result = bind(serverFd, (struct sockaddr \*)&server, socket\_length);

<https://man7.org/linux/man-pages/man2/listen.2.html>

**#include <sys/socket.h>**

**int listen(int** *sockfd***, int** *backlog***);**

int result = listen(server\_sock\_desc, 10);

<https://man7.org/linux/man-pages/man2/accept.2.html>

**#include <sys/socket.h>**

**int accept(int** *sockfd***, struct sockaddr \*restrict** *addr***,** **socklen\_t \*restrict** *addrlen***);**

int clientFd = accept(serverFd, (struct sockaddr \*)&client, &socket\_length);

<https://man7.org/linux/man-pages/man2/connect.2.html>

**#include <sys/socket.h>**

**int connect(int** *sockfd***, const struct sockaddr \****addr***,** **socklen\_t** *addrlen***);**

int result = connect(serverFd, (struct sockaddr \*)&server, socket\_length);

<https://man7.org/linux/man-pages/man2/read.2.html>

**#include <unistd.h>**

**ssize\_t read(int** *fd***, void \****buf***, size\_t** *count***);**

int byte\_count = read(serverFd, recieve\_buffer, sizeof(recieve\_buffer));

<https://man7.org/linux/man-pages/man2/write.2.html>

**#include <unistd.h>**

**ssize\_t write(int** *fd***, const void \****buf***, size\_t** *count***);**

int byte\_count = write(sock\_desc, message, strlen(message));

<https://man7.org/linux/man-pages/man3/htonl.3.html>

**#include <arpa/inet.h>**

**uint32\_t htonl(uint32\_t** *hostlong***);**

**uint16\_t htons(uint16\_t** *hostshort***);**

**uint32\_t ntohl(uint32\_t** *netlong***);**

**uint16\_t ntohs(uint16\_t** *netshort***);**

server.sin\_port = htons(SERVER\_PORT);

**References**

<https://vcansimplify.wordpress.com/2013/03/14/c-socket-tutorial-echo-server/>

<https://mohsensy.github.io/programming/2019/09/25/echo-server-and-client-using-sockets-in-c.html>

<https://learn.microsoft.com/en-us/windows/win32/winsock/getting-started-with-winsock>

**Restricted Pointers**

<https://en.wikipedia.org/wiki/Restrict>

In the C programming language, restrict is a keyword, introduced by the C99 standard,[1] that can be used in pointer declarations. By adding this type qualifier, a programmer hints to the compiler that for the lifetime of the pointer, no other pointer will be used to access the object to which it points. This allows the compiler to make optimizations (for example, vectorization) that would not otherwise have been possible.

**Socket API: Client and Server Method-Call Sequence**

