# IT304 Computer Networks
# Socket programming assignment I

## 1 Socket programming for connection oriented processes.

### 1.1 Socket

In computer networking, an Internet socket or network socket is an endpoint of a bidirectional inter process communication flow across an internet protocol-based computer network such as the internet. The term internet socket is also used as a name for an application programming interface (API) for the TCP/IP protocol stack, usually provided by the operating system. Internet sockets constitute a mechanism for delivering incoming data packets to the appropriate application process or thread, based on a combination of local and remote IP address and port numbers. Each socket is mapped by the operating system to a communicating application process or thread. A socket address is the combination of an IP address (the location of the computer) and a port (which is mapped to the application program process) into a single identity, much like one end of a telephone connection is the combination of a phone number and a particular extension.

#### 1.1.1 Two types of socket

1. Stream sockets: They are connection oriented reliable sockets also called TCP sockets.

2. Datagram sockets: They are connectionless unreliable sockets also called UDP sockets.

### 1.2 TCP

TCP is a transport layer protocol used by applications that require guaranteed delivery. It is a sliding window protocol that provides handling for both timeouts and re-transmissions. TCP establishes a full duplex virtual connection between two endpoints. Each endpoint is defined by an IP address and a TCP port number. The operation of TCP is implemented as a finite state machine. The byte stream is transferred in segments. The window size determines the number

of bytes of data that can be sent before an acknowledgement from the receiver is necessary.

## 1.3 The client server model

Most interprocess communication uses the client server model. These terms refer to the two processes which will be communicating with each other. One of the two processes, the client, connects to the other process, the server, typically to make a request for information. Notice that the client needs to know of the existence of and the address of the server, but the server does not need to know the address of (or even the existence of) the client prior to the connection being established. Notice also that once a connection is established, both sides can send and receive information. The system calls for establishing a connection are somewhat different for the client and the server, but both involve the basic construct of a socket. A socket is one end of an interprocess communication channel. The two processes each establish their own socket.

The steps involved in establishing a socket on the client side are as follows:

1. Create a socket with the *socket()* system call
2. Connect the socket to the address of the server using the *connect()* system call.
3. Send and receive data. There are a number of ways to do this, but the simplest is to use the *read()* and *write()* system calls

The steps involved in establishing a socket on the server side are as follows:

1. Create a socket with the *socket()* system call.
2. Bind the socket to an address using the *bind()* system call. For a server socket on the Internet, an address consists of a port number on the host machine.
3. Listen for connections with the *listen()* system call.
4. Accept a connection with the *accept()* system call. This call typically blocks until a client connects with the server. Send and receive data

The basic code structure is given below: 1

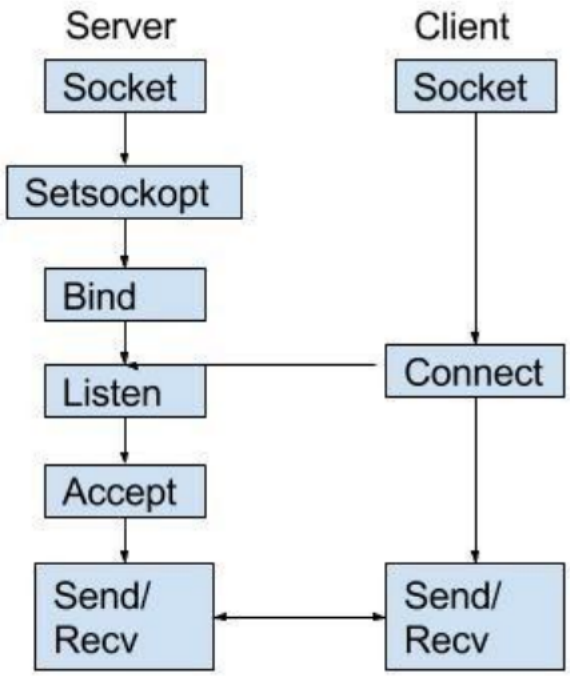


Figure 1: Client-server model

## 1.4 Time Functions in C

### 1.4.1 *ctime()*

```
char * ctime ( const time_t * ptr_time );
```

Parameters:

Pointer *ptr_time* to a *time_t* object that contains a calendar time.
Return Value:
The function returns a C string containing the date and time information. The string is followed by a newline character ('\n') Converts the *time_t* object pointed by timer to a C string containing a human-readable version of the corresponding local time and date. The functions *ctime* and as *ctime* share the array which holds this string. If either one of these functions is called, the content of the array is overwritten.

Explanation:

The string that is returned will have the following format:

*Www Mmm dd hh:mm:ss yyyy*
*Www* = which day of the week. *Mmm* = month in letters. *dd* = the day of the month. *hh:mm:ss* = the time in hour, minutes, seconds. *yyyy* = the year.

```
#include <stdio.h>
#include <time.h>
int main ()
{
time_t time_raw_format;
time ( &time_raw_format );
printf ( "The current local time: %s", ctime(&time_raw_format));
return 0;
}
```

### 1.4.2 *time()*

Syntax:

```
#include <time.h>
time_t time( time_t *time );
```

Description:
The function *time()* returns the current time, or -1 if there is an error. If the argument time is given, then the current time is stored in time.

# 2 Exercises

1. Study simple client-server hello message program. Compile and run the program and understand output.
2. FTP using socket programming: Design an application of FTP using TCP. Implement GET, PUT and LIST methods in FTP. Design your own message structure. Use appropriate data structure. Hint:[TCP client server code is available in folder]

# 3 Submission guidelines

Submit a zip file which contain all the codes and output screenshot. Submission only considered in c/c++ language.

The naming convention should be as following:

- zip file name must be ID_socket1.
- Codes must have name ExerciseNumber_Client and ExerciseNumber_server.

# 4 Suggested reading:

1. **Socket Programming:**
   `https://www.geeksforgeeks.org/socket-programming-cc/`
2. **FTP:**
   `https://gist.github.com/XBachirX/865b00ba7a7c86b4fc2d7443b2c4f238`