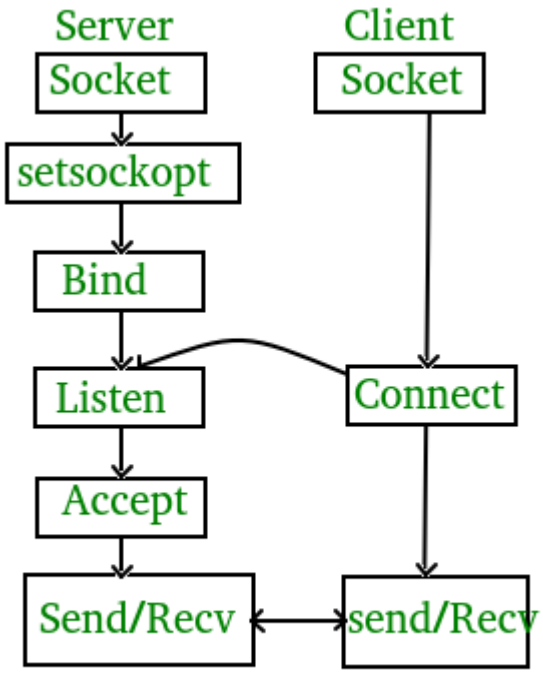
Creating a Socket Program in C++

The general process of creating a client-server socket progrem is to first create a server program to handle the client requests, and then a client program to provide the requests to the server.



Windows specific stuff -

For the Windows OS, we first need to perform some additional steps before we begin writing the actual socket code. We need to link the `wsock32.lib` file (in-built in windows) to the linker. Moreover, the header file to include is `winsock.h`.

```
#include <winsock.h>
```

Also, we need to call the `WSAStartup` function for the socket functions to work in windows. This function must be the first function called by an application or DLL. It allows an application to specify the version of Windows Sockets required and retrieve details of the specific Windows Sockets implementation.

The application can only issue further socket functions after successfully calling `WSAStartup`.

```
//Initialise the WSA variables (for windows)
    WSADATA ws;
    if (WSAStartup(MAKEWORD(2, 2), &ws) < 0)
    {
        cout << "WSA Failed to initialise... " << endl;
        WSACleanup();
```

```
        exit(EXIT_FAILURE);
    }
    else
    {
        cout << "WSA initialised... " << endl;
    }
```

Here, the `WSADATA` structure contains information about the Windows Sockets implementation. The `MAKEWORD(2,2)` parameter of `WSAStartup` makes a request for version 2.2 of Winsock on the system, and sets the passed version as the highest version of Windows Sockets support that the caller can use.

With that, we're ready to write the server code.

The Server Program -

Refer this program for the actual code.

Creating a socket -

The first step is to create a socket at the server side. This is done using the `socket()` function.

```
//Initialising the socket
    int nSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

Here the parameters correspond to-

- Address_Family = AF_INET This corresponds to the internetwork address families like UDP, TCP, etc.
- Socket_Type = SOCK_STREAM This corresponds to the TCP packet type. We can use SOCK_DGRAM for UDP sockets.
- Protocol = IPPROTO_TCP This corresponds to the protocol that we propose to use. Here we use TCP. For UDP, we have IPPROTO_UDP.

Initialising the sockaddr structure

The `sockaddr_in` is a C++ structure which stores essential information regarding our server socket. This structure needs to be initialised and bound with our server socket. The port number used is 9900

```
#define PORT 9900
struct sockaddr_in srv;
//Initialise the environment for sockaddr structure
    srv.sin_family = AF_INET;
    srv.sin_port = htons(PORT);
    srv.sin_addr.s_addr = INADDR_ANY;
    memset(&srv.sin_zero, 0, sizeof(srv.sin_zero));
```

Here -

- `sin_family` - The address family for the transport address. This member should always be set to `AF_INET`.
- `sin_port` - A transport protocol port number.
- `sin_addr` - An IN_ADDR structure that contains an IPv4 transport address. (`INADDR_ANY` gives the localhost address)
- `sin_zero` - Reserved for system use. Must be initialised to zero.

Selecting socket options

With the `setsockotpt()` function, we can set many options which alter the behaviour of the socket. There are a lot of options available for sockets. More information can be found here.

```
int setsockopt(
  SOCKET     s,
  int        level,
  int        optname,
  const char *optval,
  int        optlen
);
```

Here -

- `s` - A descriptor that identifies a socket.
- `level` - The level at which the option is defined (for example, SOL_SOCKET).
- `optname` - The socket option for which the value is to be set (for example, SO_BROADCAST). The optname parameter must be a socket option defined within the specified level, or behavior is undefined.
- `optval` - A pointer to the buffer in which the value for the requested option is specified.
- `optlen` - The size, in bytes, of the buffer pointed to by the optval parameter.

Let's say that we want our socket to bind to addresses that are already in use. In order to ensure this, we subsitute the following values -

```
//Setting Socket Options
int nOptVal = 0;
int nOptLen = sizeof(nOptVal);
nRet = setsockopt(nSocket, SOL_SOCKET, SO_REUSEADDR, (const char*)&nOptVal,
nOptLen);
```

Binding the socket

The next step is to bind the socket which we created to a local port using the `sockaddr_in` structure. This is done with the `bind()` function.

```
int bind(
  SOCKET         s,
```

```
  const sockaddr *addr,
  int            namelen
);
```

Here -

- `s` - A descriptor identifying an unbound socket.
- `addr` - A pointer to a sockaddr structure of the local address to assign to the bound socket .
- `namelen` - The length, in bytes, of the value pointed to by the name parameter.

We substitute the following -

```
//Bind the socket to the local port
nRet = bind(nSocket, (sockaddr*)&srv, sizeof(sockaddr));
```

Listening to client requests -

Now that we have bound our socket to a port on our system, it is time for the server to listen to requests from the client. For this we use the `listen()` function. When in this state, the socket keeps listening to an incoming connection.

```
int listen(
  SOCKET s,
  int    backlog
);
```

Here -

- `s` - A descriptor identifying a bound, unconnected socket.
- `backlog` - The maximum length of the queue of pending connections. For example, if the number of clients exceed the `backlog` value, they need to wait in the wait queue (maintained internally).

Let's say that we have a backlog of 5. In order to listen to new connections, we write the following -

```
//Listen to requests from the client
nRet = listen(nSocket, 5);
```

References -

1. https://docs.microsoft.com/en-us/windows/win32/winsock/initializing-winsock
2. https://youtu.be/W9b9SaGXIjA
3. https://docs.microsoft.com/en-us/windows/win32/api/ws2def/ns-ws2def-sockaddr_in
4. https://docs.microsoft.com/en-us/windows/win32/api/winsock/nf-winsock-setsockopt

5. https://docs.microsoft.com/en-us/windows/win32/api/winsock/nf-winsock-bind
6. https://docs.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-listen