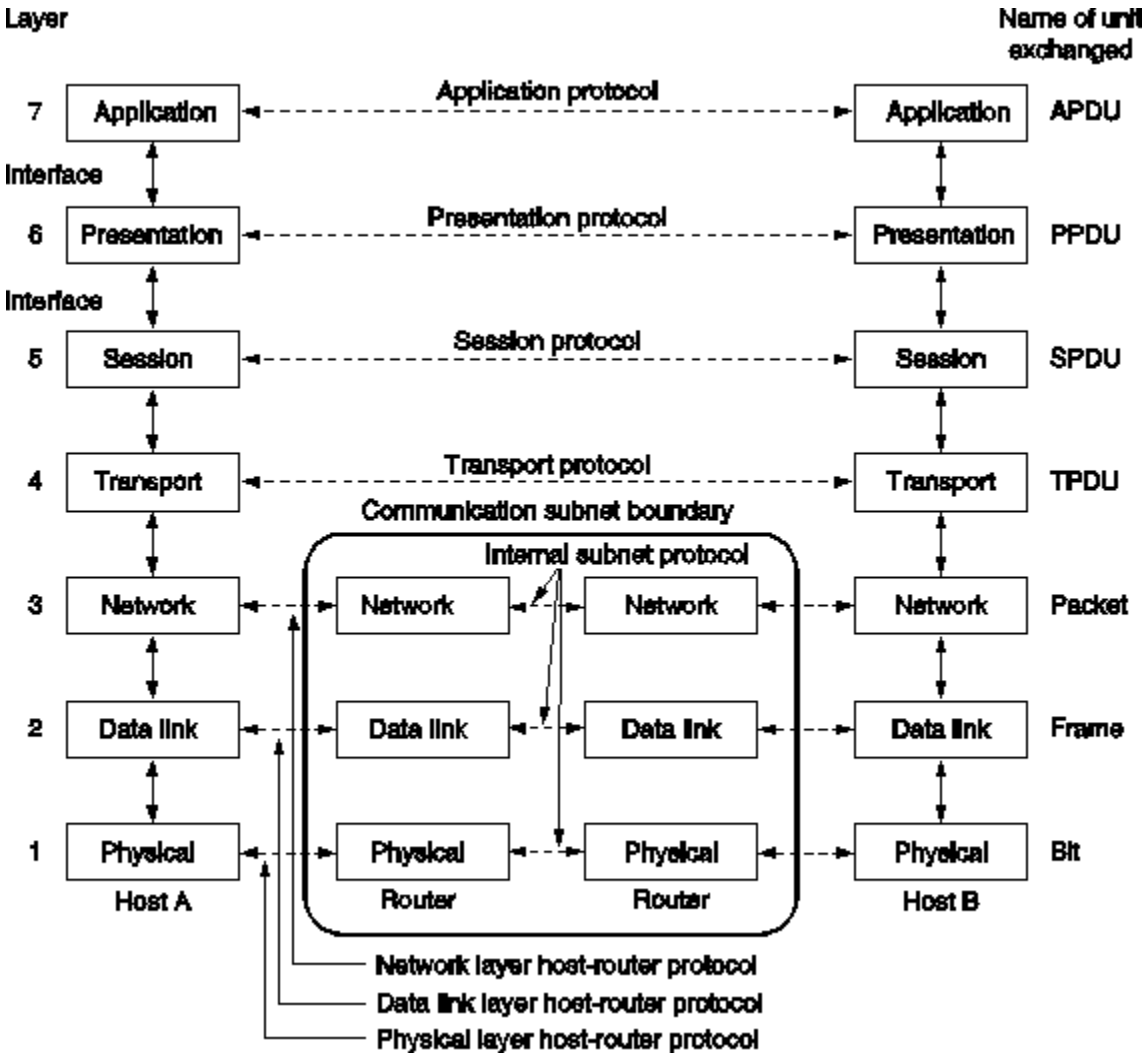
CO650 Advanced Programming - C++ Sockets

Sockets were initially developed for Unix. Windows developed their own version of the sockets called Winsock. On Windows systems, the core components of the socket come in the form of a dll file. (ws2_32.dll)

7 Layers of OSI model (Open Systems Interconnection)



The OSI model describes seven layers that computer systems use to communicate over a network. This is what happens underneath the surface when we send data from one machine to another over a network.

Architecture

- In order for connection to take place between two devices via the network, the data has to be sent from the application layer and descend down to the physical layer on a device sending data, and then ascend from the physical layer to the application layer on the device receiving the data.
- The connection is always between two devices, and each side uses its own IP and port number. Usually one side is called the **client**, the other side the **server**
- The server is continually waiting for incoming connections. This is called **listening** which is always done on a certain IP and port number.

IP addresses

- BOth the server and client use an IP and port number.
- THe IP address of both server and client is configured during Network setup unless it is allocated dynamically.
- A machine may have more than one network interface card(NIC), in which case it may have more than one IP address.
- When developing Network Programs the port number of the server is usually specified within the code, whereas the client port number is allocated by the O/S.
- The **Loopback address** 127.0.0.1 refers to the current machine. This can be used during development to test both client and server on a single machine.

Ports

- Port numbers (16 bit address) can be any integer between 1 and 65.535.
- Ports 1…1023 are described as well known ports and are reserved for specific applications (port 21 FTP).
- It is recommended to choose a number over 1024. To be sure that your desired port isn't already in use

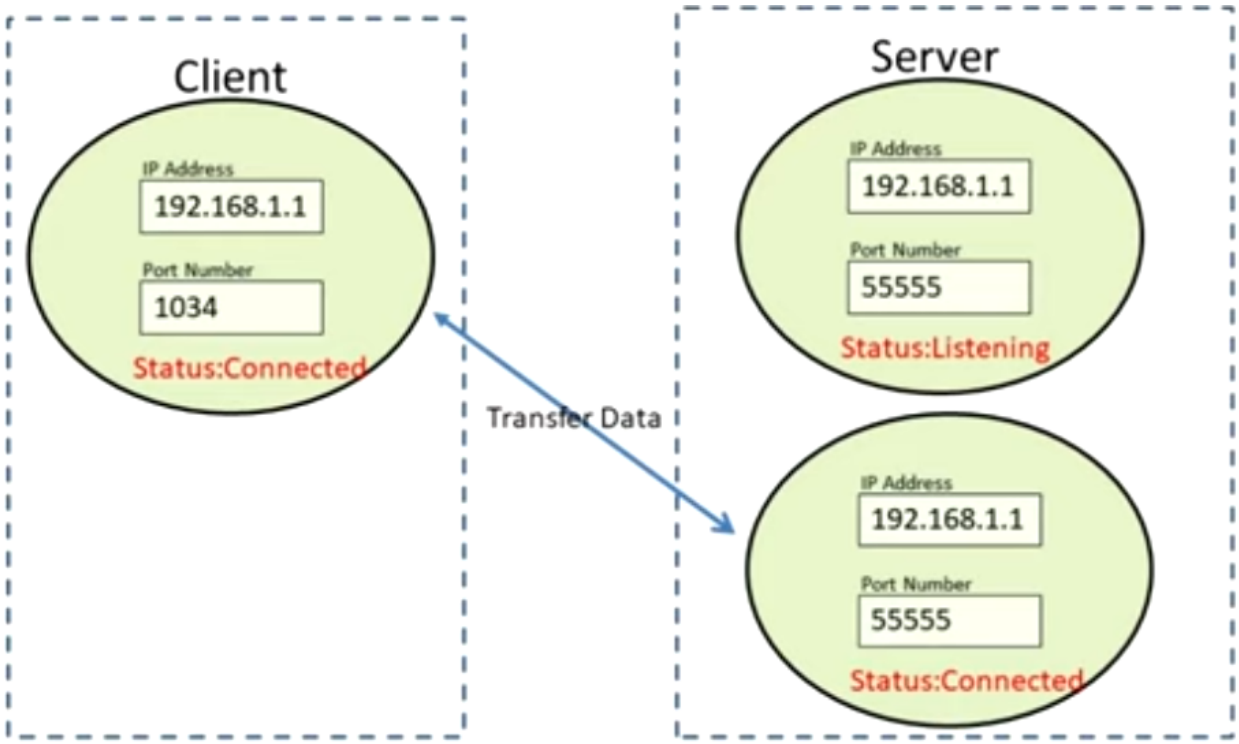
Sockets

-
- Definition "A pipe between two computers on a network through which data flows" (Mulholland 2004).
- Almost all Winsock functions operate on a socket, as it's your handle to the connection. Both sides of the connection use a socket.
- Sockets are also two-way, data can be both sent and received on a socket.
- There are two common types of socket
    - Streaming socket (SOCK_STREAM) TCP
    - Datagram socket (SOCK_DGRAM) UDP

Request a Connection

The client socket tries to connect to the known IP address of the server.

Once it's been bound, the client puts out a request for a connection to a server. Once the request has been accepted by the server, the server duplicates the socket to enable the connection between them. It is creating a new socket bound to its own IP address and valid port. From this point, the client socket and the server connection socket just created will be able to communicate, while the listening socket will keep listening for other connections.



Server Functions

1. Initialize WSA - WSAStartup().
2. Create a socket - socket().
3. Bind the socket - bind().
4. Listen on the socket - listen().
5. Accept a connection - accept(), connect().
6. Send and receive data - recv(), send(), recvfrom(), sendto().
7. Disconnect - closesocket().

Client Functions

1. Initialize WSA - WSAStartup().
2. Create a socket - socket().

3. Connect to the server - connect().
4. Send and receive data - recv(), send(), recvfrom(), sendto().
5. Disconnect - closesocket().

The Server Code

- The server must load the DLL by invoking WSAStartup
- It then creates a socket specifying the protocol to be used
- It binds the server's IP address to the socket
- Then listens for clients trying to establish connections
- On a client connecting the server creates a new socket to handle the client server communication

Initializing the DLL

Winsock DLL can be initiated by following:

```
int WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData);
```

wVersionRequested	The highest version of Windows Sockets specification that the caller can use. The high-order byte specifies the minor version number; the low-order byte specifies the major version number
lpWSAData	A pointer to the LPWSADATA data structure that is to receive details of the Windows Sockets implementation

If successful, the WSAStartup function returns zero.

The Socket Function

- The socket function creates a socket that is bound to a specific transport service provider

```
SOCKET WSAAPI socket (int af, int type, int protocol);
```

af	The address family specification (AF_INET for UDP or TCP).
type	The type specification for the new socket (SOCK_STREAM for TCP and SOCK_DGRAM for UDP).
protocol	THe protocol to be used (IPPROTO_TCP for TCP)

Deregister Winsock2 DLL

- The WSA Cleanup function terminates use of Winsock 2 DLL.
- The return value of the socket function is zero if the operation was successful. Otherwise, the value SOCKET_ERROR is returned. If the socket function **does not return 0** it would mean there was an **issue** with binding provided information to the socket or creating the socket. Because we have initialized the DLL, we need to **deregister** the previous application from using it so that other applications can make use of it. We can use the following function to free up the resources:

```
int WSACleanup(void);
```

- When it has completed the use of WIndows Sockets, the application or DLL must call WSACleanup to deregister itself from a Windows Sockets
- Multiple applications may share a DLL. Windows tracks the number of applications using each DLL and will only remove the DLL from system memory when it is no longer required.

```
SOCKET serverSocket = INVALID_SOCKET;
serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (serverSocket == INVALID_SOCKET) {

cout << “Error at socket(): “ << WSAGetLastError() << endl;
WSACleanup();
return 0;

} else {

cout << “socket() is OK!” << endl;

}

//….

WSACleanup();
```

Close Socket

- Closes the socket passed as an argument
- The socket must have previously been opened through a call to socket

```
int closesocket(Socket s);
```

Bind the IP address and Port to the Socket

- Associates a local address with a socket

```
int bind(SOCKET s, const struct sockaddr* name, int socklen);
```

s	Descriptor identifying an unbound socket
name	Address to assign to the socket from the *sockaddr* structure. This is a structure that contains attributes for the ip address and port number.
socklen	Length in bytes of the address structure. sizeof() command can calculate this for us.

- If no error occurs, bind() returns zero. Otherwise it returns SOCKET_ERROR.
- The **SOCKADDR_IN** structure is used by Windows SOckets (IP4) to specify a local or remote endpoint address to which to connect a socket.

```
struct sockaddr_in {

short sin_family;
unsigned short sin_port;
struct in_addr sin_addr;
char sin_zero[8];

}
```

sin_family	Address family (must be AF_INET)
sin_port	IP port

sin_addr	IP address
sin_zero	Padding to make the structure the same size as **SOCKADDR**. The **htons** function returns the value in TCP/IP network byte order.

Bind Example

```
sockaddr_in service;
service.sin_family = AF_INET;
InetPton(AF_INET, _T("127.0.0.1"), &service.sin_addr.s_addr);
service.sin_port = htons(port);
if (bind(serverSocket, (SOCKADDR*)&service, sizeof(service)) == SOCKET_ERROR){
        cout << "bind() failed: " <<  WSAGetLastError() << endl;
        closesocket(serverSocket);
        WSACleanup();
        return 0;
}
else{
        cout << "bind() is OK!" << endl;
}
```

Listening with socket using Listen Function

- Listen function places a socket in a state in which it is listening for an incoming connnection.

int listen(SOCKET s, int backlog);

s	Descriptor identifying a bound, unconnected socket.
backlog	The maximum number of connections allowed (also OS dependant)

If no error occurs, listen returns zero. Otherwise, SOCKET_ERROR is returned.

Listen Example

```
if ( listen(serverSocket, 1) == SOCKET_ERROR)
        cout << "listen(): Error listening on socket " <<  WSAGetLastError()<< endl;
else
        cout << "listen() is OK, I'm waiting for connections..." << endl;
```

Accept connection upon request

- The accept function permits an incoming connection on a socket.
- The Accept function will pause server execution until the client has established connection with the socket that was listening.

SOCKET accept(SOCKET s, struct sockaddr* addr, int* addrlen);

s	Descriptor that identifies a socket that has been placed in a listening state with the listen() function
addr	Optional structure containing the client address information
addrlen	Optional size of the address structure (if included)

If no error occurs, accept() returns the value of type SOCKET that is a descriptor for the new socket that is connected to the client. The original listening socket can then be used to listen for other incoming calls.

Accept Example

```
SOCKET acceptSocket;
acceptSocket = accept(serverSocket, NULL,NULL);
if (acceptSocket == INVALID_SOCKET){
        cout <<  "accept failed: " << WSAGetLastError() << endl;
                        WSACleanup();
                        return -1;
}
```

Connect Function executed by the Client

- Connects a client to a server (invoked from within the client)
- When the connect function is executed, the socket will be automatically bound to client IP address and port number.

int connect(SOCKET s, const struct sockaddr* addr, socklen_t addrlen);

s	Descriptor that identifies a socket
addr	Structure containing server IP address and port
addrlen	Size in bytes of addr structure

- Connect will wait 75 seconds for server to respond
- Returns 0 if successful or SOCKET ERROR if not.

Connect Example

```
sockaddr_in clientService;
clientService.sin_family = AF_INET;
InetPton(AF_INET, _T("127.0.0.1"), &clientService.sin_addr.s_addr);
clientService.sin_port = htons(port);
if ( connect(clientSocket, (SOCKADDR*)&clientService, sizeof(clientService)) == SOCKET_ERROR){
        cout << "Client: connect() - Failed to connect." << endl;
        WSACleanup();
        return 0;
}
else  {
      cout << "Client: connect() is OK." << endl;
      cout << "Client: Can start sending and receiving data..." << endl;
}
```