

CO650 Advanced Programming - UDP

C++ Transmitting Data (UDP)

UDP stands for **User/Universal Datagram Protocol**. In UDP, client and server don't maintain a constant connection. Either can transmit a packet but does not establish a connection or maintain a constant one - so they do not use an accept or connection function like TCP.

Instead, it allows client and server to specify IP address and port number which they want to send data to, and also receive data from. Advantage of using this method of transmission is that it **reduces lag** associated with establishing connection.

It is, however, **less secure** and **less reliable** as there is no guarantee that different packets used to split up your data will actually reach the destination. So it is usually used in circumstances where it is not crucial for data to be received in its **entirety**. For example, if data is regularly sent, missing data could be retrieved in the next transmissions so few missing packets would not matter overall.

UDP Setup

- Initialize the library as with TCp
- Create a socket configured for UDP
- For a server only bind the socket to an address
- Invoke **sendto** to transmit data
- Invoke **recvfrom** to receive data
- Note: There is no notion of a fixed connection between client and server and hence the server doesn't utilize the accept function nor the client the connect function

UDP Socket

- Invoke the socket function as with a TCP socket

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

- The address family specification is **AF_INET** as with TCP
- The type specification for UDP is **SOCK_DGRAM**
- The protocol to be used is passed down as a constant- **IPPROTO_UDP**

```
socket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
```

Sending Data

- The **sendto** function sends data to a specific destination.

```
int sendto(SOCKET s, const char *buf, int len, int flags, const struct sockaddr *to, int tolen);
```

s	A descriptor identifying a (possibly connected) socket
buf	A pointer to the data to be transmitted
len	Size in bytes, of the data pointed to by the <i>buf</i> parameter.
flags	Flags that specify the way in which the call is made.
to	As an optional pointer to a <code>sockaddr_in</code> structure that contains the address of the target socket
tolen	The size, in bytes, of the address pointed to by the <i>to</i> parameter

If no error occurs, `sendto` returns the total number of bytes sent, else it returns -1.

UDP Send Example

```
*****
sockaddr_in address;
address.sin_family = AF_INET;
InetPton(AF_INET, T("127.0.0.1"), &address.sin_addr.s_addr);
address.sin_port = htons(5555);
*****
char buffer[200] = "send this to server";
int bytesSent = sendto(socket, (const char*)buffer, strlen(buffer), 0,
                      (struct sockaddr*)&address, sizeof(address));

if(bytesSent == -1){
    cout << "Error transmitting data." << endl;
    WSACleanup();
    return 0;
}
else {
    cout << "Data sent : " << buffer << endl;
}
```

Receiving Data

- The `recvfrom` function receives a datagram and stores the source address. This is a blocking function.

```
int recvfrom(SOCKET s, char *buf, int len, int flags, struct sockaddr *from, int *fromlen);
```

s	A descriptor identifying a bound socket
buf	A buffer for the incoming data
len	Size in bytes, of the buffer pointed to by the buf parameter.
flags	Flags that modify the behavior of the function
from	An optional pointer to a buffer in a sockaddr_in structure that will hold the source address upon return.
fromlen	The size, in bytes, of the address pointed to by the from parameter

If no error occurs, **recvfrom** returns the number of bytes received else -1.

UDP Receive Example

```
char buffer[200] = "";
sockaddr_in clientAddress;
int clientAddress_length = (int)sizeof(clientAddress);

int bytes_received = recvfrom(socket, buffer, 200, 0, (struct sockaddr*)&clientAddress,
&clientAddress_length);

if (bytes_received < 0){
    cout << "Could not receive datagram." << endl;
    WSACleanup();
    return 0;
}
```

Sending Objects via UDP

Class Vector3D Example

```
class Vector3D {  
    public:  
        float x, y, z;  
        Vector3D(float X, float Y, float Z) : x(X), y(Y), z(Z) { }  
};  
  
Vector3D point(0.0, 0.0, 0.0);  
  
//point.x = 10;  
//point.y = 4.0;  
//point.z = 9.0;
```

- A class representing a 3D vector with x,y & z member variables.
- x, y, & z are public and can therefore be manipulated directly.

In order to send this object via UDP, we would first convert it to a char array. Primitive data types should be converted to char arrays before being transmitted.

The sprintf() performs this task

```
sprintf(char* buffer, const char* format, list of arguments);
```

The % character in the format string is a placeholder.

6.1 indicates the size and decimal places and the f that a float will be passed to it.

```
Vector3D position(1.0,2.0,3.0);  
  
char positionBuffer[200] = "";  
  
sprintf(positionBuffer, "%6.1f %6.1f %6.1f", position.x, position.y, position.z);
```