C++
Transmitting Data (TCP)

CO650 Advanced Programming

Sending Data

- The **send** function sends data on a connected socket.

```
int send( SOCKET s,
          const char *buf,
          int len,
          int flags );
```

- S : The descriptor that identifies a connected socket.
- *buf* : A pointer to the buffer to the data to be transmitted.
- *len* : The length, in bytes, of the buffer pointed to by the *buf* parameter.
- *flags* : Optional set of flags that influences the behavior of this function (No routing etc).
- If no error occurs, **send** returns the number of bytes sent. Otherwise SOCKET_ERROR is returned.

Send Example

In this example we are using the socket variable named clientSocket but any valid socket connected socket on the server or client could be used.

```
.............
char buffer[200];
printf("Enter your message ");
cin.getline(buffer,200);
int byteCount = send(clientSocket, buffer, 200, 0);
if(byteCount == SOCKET_ERROR){
        printf("Server send error %ld.\n", WSAGetLastError());
        return -1;
}
else{
        printf("Server: sent %ld bytes \n", byteCount);
}
```

Receiving Data

- The **recv** function receives data from a connected socket.

```
int recv( SOCKET s,
          char *buf,
          int len,
          int flags );
```

- S : The descriptor that identifies a connected socket.
- *buf* : A pointer to the buffer to receive the incoming data.
- *len* : The length, in bytes, of the buffer pointed to by the *buf* parameter.
- *flags* : Optional set of flags that influences the behavior of this function.
- If no error occurs, **recv** returns the number of bytes received. If the connection has been gracefully closed, the return value is zero. Otherwise SOCKET_ERROR is returned.

Receive Example

In this example we are using the socket variable named acceptSocket but any valid socket connected socket on the server or client could be used.

```
.............
char receiveBuffer[200] = "";
int byteCount = recv(acceptSocket, receiveBuffer, 200, 0);
if (byteCount < 0){
        printf("Client: error %ld.\n", WSAGetLastError());
        return 0;
}
else {
        printf("Received data :  %s  \n", receiveBuffer);
}
```

Transmitting Objects

Sending an object. Assuming the class Data has be defined elsewhere. Cast the object's address to a char *

```
Data data;
data.health = 100;
byteCount = send(socket,(char *)&data,sizeof(Data), 0);
```

Receiving an object

```
Data data;
byteCount = recv(clientSocket,(char *) &data, sizeof(Data), 0);
printf("Health :  \"%d\"\n", data.health);
```

User Input

- Include **<iostream>** and **using namespace std**
- Invoke cin.getline() to capture a series of characters (including spaces) entered into the console.
- Pass a valid char array and length of the array to getLine as arguments.

```
char buffer[200] = "";
cout << "Enter your message “ ;
cin.getline(buffer,200);
cout “You typed “ << buffer << endl;
```

Assigning Values to char array

When defining the character array we can initialise its value.

```
char sendBuffer[200] = "Message received by server";
```

We can’t however use the assignment operator after it has been defined.

Instead we must use the strcpy function or if depreciated strcpy_s.

```
char buffer[200] ;
strcpy_s(buffer,”hello World”);
```

Use the strlen function to return the number of characters in the array.

```
strlen(sendBuffer);
```

Comparing Character Arrays

- The strcmp() function compares two strings

```
int strcmp( const char * string1, const char * string2);
```

- Takes two string as arguments (pointers to character arrays)
- Returns 0 if they are equal
- Returns < 0 if string1 less than string2
- Returns > 0 if string 1 greater than string2

```
if (strcmp(buffer,“PASSWORD")== 0){
        // Strings are equal
}
```