Socket++

Tutorial: Usage and Examples

1. Introduction

Sockets provide a mechanism for processes to exchange data. This tutorial present a collection of classes in C++ that provide an easy and intuitive interface for TCP/IP socket programming. These classes allow a programmer to create and use sockets for various purposes without the jargon required to do so in C, while providing type-safety at the same time.
Unlike the many C++ libraries already available for socket programming (such as Boost.ASIO, Practical C++ Sockets etc.), our interface is IP version agnostic, i.e., IPv4 and IPv6 addresses can be used interchangeably. Thus, applications that use this interface will be able to disregard any problems caused by the transition from IPv4 to IPv6. Further, this interface allows sockets to be treated like streams, and data can be input/output using the standard "<<" and ">>" operators. This uses the latest C++11 features and thus provide more flexibility for the user as well as developer to implement them. You don't need to know about the features of C++11 used in this library implementation, but still if you are interested you can look our code and documentation at: https://github.com/ankitgupta29/socket.
To know more about C++11 features refer: http://stroustrup.com/C++11FAQ.html.

This tutorial helps you in using our C++ socket++ library. The tutorial is not about what is socket programming, but it's about how to use our library for socket programming. If you want to learn about sockets please refer the Beej guide (http://beej.us/guide/bgnet/) or any other socket programming tutorial.

We start with explaining basic classes of our library and how you can use them in your code for their usage. Each topic is explained with the help of example.
Let's get started!!

2. Creating Address Object:

The following example demonstrates how to create a default address with family as AF_INET family and socket type as SOCK_STREAM using the inet_stream_addr class.
User can create various address types with all the possible combinations of address type and port. Port can either be string or int and similarly Ipaddress can be "http name" or an 4 byte/6 byte Ipaddress.

Code	Output
Inet_stream_addr a("http", "google.com"); cout << a;	[173.194.33.104]
Inet_stream_ a("http", "localhost"); cout << a;	[::1] [127.0.0.1]
Inet_stream_ a("http", "ipv6.google.com"); cout << a;	[2001:4860:800f::63]

Of course, instead of "http", one can write the port number as 80 or "80".

```
#include "base_address.hpp"
int main()
{
int port = 80;
inet_stream_addr a1(port,"www.google.com");
string port2 = "80";
inet_stream_addr a2(port2 ,"10.12.112.2");
inet_stream_addr a3(port, "localhost");
}
```

3. Creating a Socket Object:

This example tells how to create a raw socket of Stream type.

```
#include "socket.hpp"
int main()
{
//creating a stream socket
sock_stream s;

//return the sock id of socket created
s.get_sockfd();

//print the socket Id
cout<<s2.get_sockfd();

// set the socket to non-blocking
s.set_non_blocking(1);

//set the socket to blocking again
```

```
s.set_non_blocking(0);

//set the keep alive option of a socket
s.set_keep_alive(1);

//set the socket to no-delay
s.set_no_delay(1);

//set the socket to re use address mode
s.set_reuse_address(1);

//close the socket
s.socket_close();


//write bytes to socket
string data = "Write to a socket";
s.write_bytes(data);

//return to buffer if there is something to read on socket
string buffer;
buffer = s.read_bytes();
cout << buffer;
}
```

4. Creating a Server Socket:

If we want a socket that would use for creation of a server, then we can directly use following notation. We can bind that socket with INET address for particular port and ipaddress and set the number of maximum client that can connect to that server. User do not need to call bind and listen to this server socket as they are internally implemented.
If user wants to create a synchronous server(which is in blocking mode) it can use a accept_conn() function, which is a wrapper for the C accept call. This function accepts the socket id of server on which it is listening and returns the sock fd which client uses further for communication with server.

```
#include<server_socket.hpp>
int main ()
{
int port = 80;
int max_listen = 10;
string ipaddress = "localhost";
server_sock_stream serversock(port,ipaddress,max_listen);
string port2 = "80";
server_sock_stream serversock2(port2,ipaddress,max_listen);
sock_stream sock;
int new_sock = serversock.accept_conn(sock.get_sockfd());
}
```

5. Creating a Client Socket:

we can create a asynchronous stream client just by calling like this: client_sock_stream
This objects takes argument of port and ipaddress of a server to which client needs to connect.
#include<client_socket.hpp>

```
int main ()
{
int port = 80;
string ipaddress = "localhost";

client_sock_stream clientsock(port,ipaddress);
string port2 = "80";
client_sock_stream clientsock2(port2,ipaddress);
sock_stream sock;
}
```

Up to this point all the basics to be used by the user for creating its application is explained. Now we will with the help of all the above classes create a asynchronous TCP echo_server and echo_client.

6. Creating Application Server:

```
//echo_server.cpp

#include "server.hpp"
#include "client_socket.hpp"
template <typename T>
class handle: public socket_handler<T>
{
public:
```

```
handle(unique_ptr<Socket<T>> sock): socket_handler<T>(std::move(sock)){};
int handle_read()
{
socket_handler<T>::handle_read();
write(this->read_data);
}
void write(string data)
{
this->write_data = data;
socket_handler<T>::write();
}
};

int main()
{
Reactor *rec = Reactor::get_instance();
async_server<inet_stream_addr, handle, server_socket_handler> server(8080);
rec->Run();
return 0;
}
```

Explaination:

Handle Class:
User has to define handle class in server code which is inherited from the socket_handler class and defines functions handle_read(), handle_write().
Whenever any read or write or new connection event comes on the server socket, handles passes to these function and these functions can take care of reading, writing and connection accept from socket.

This application server inherits from the server_socket_handler and used for accepting the connection from the clients. We can call this as a acceptor class to a server.

Reactor *rec = Reactor::get_instance();
rec->Run();
Since all the event handling for the asynchronous server is taken care by the reactor pattern, so user has to take the instance of the reactor pattern and call the run method of the class. This run method implements the select system call and thus ensures that whenever there are any new connections or read and writes event arrives on the existing connections, then proper handling of them will be taken care.
To read more about reactor pattern refer:

7. Creating a Application Client:

User can make application client as following:

```
#include "client.hpp"
template <typename T>
class handle: public client_socket_handler<T>
{
public:
handle(unique_ptr<client_sock_stream> sock): client_socket_handler<T>(std::move(sock)){};
int handle_read()
{
client_socket_handler<T>::handle_read();
std::cout<<this->read_data;
write("hello");
}

void write(string data)
{
this->write_data = data;
client_socket_handler<T>::write();
}
};

int main()
{
try
{
Reactor *rec = Reactor::get_instance();
async_client<inet_stream_addr, handle> client(80, "localhost");
client.write("hello");
rec->Run();
}
catch (sock_error& serr)
{
std::cout<<"Got error"<<serr.what();
}
return 0;
}
```

User has to create async_client with inet addr and handle object, through which handle_read() and write () be called whenever there is data on the client socket from the server side or if client needs to write to the server then the client simply call the write()

User can define all these within try and catch and handles if there is any error and throw exception.

Getting Data from Client:

WebClient w("google.com"); c << "GET / HTTP/1.0\n"; c << "Host: www.google.com\n\n"; c >> cout;	The contents of the google.com home page.
WebClient w("google.com"); cout << w.getWebPage("imghp");	The contents of the google images home page.

8. REFERENCES:

- www.stroustrup.com
- www.stackoverflow.com
- Beej guide : http://beej.us/guide/bgnet/
- Boost asio :http://www.boost.org/doc/libs/1_52_0/doc/html/boost_asio.html
- Ace Socket library: http://www.cs.wustl.edu/~schmidt/ACE-overview.html
- Poco C++ Library: http://pocoproject.org/