BRO-SERVER

We will start by writing the full code in one cpp file and at a later stage will divide it into multiple header files and other libraries.

We will develop the server under two roles.

1. The server designer / creator
2. The web application developer / user of the web server

V1: The basic structure and dual roles

```
#include<iostream>
using namespace std;

// Amit [The bro programmer]

class Error{
public:
bool hasError(){
return false;
}
string getError()
{
return "";
}
};

class Request{};
class Response{
public:
void setContentType(string contentType){

}
Response & operator<<(string content)
{

return *this;
}

};

class Bro{
public:
void setStaticResourcesFolder(string staticResourcesFolder){

}
//for get method we will need 2 params
// first is url pattern and second is a pointer to a function which can store address of the
//lambda function
// and how do we declare pointer to a function
```

```
//step-1: put a star * and then name of pointer
//step-2: mention the return type of that function // void here
//step-3: and then mention the signature i.e. params type of that function

void get(string urlPattern, void (*callback)(Request &, Response &) ){

}

void listen(int portNumber, void (*calback)(Error &))
{
}


};

// Bobby [The app developer]

int main()
{
Bro bro;
// link to all static resources files eg. js , css files
bro.setStaticResourcesFolder("whatever");

// serve welcome html when a request arrives with "/"
bro.get("/",[] (Request &request, Response &response)   {
const char *html = R""""(   // multiline string - pointer named html will be pointing to the base address of
this html
<!DOCTYPE HTML>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Whatever</title>
</head>
<body>
<h1>Welcome</h1>
<h3>Some text</h3>
<a href="getCustomers">Customers List</a>
</body>
</html>
)"""";
response.setContentType("text/html"); //setting mime type
response<<html;
});


bro.get("/getCustomers",[](Request &request, Response &response) {

const char *html = R""""(
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Whatever</title>
</head>
```

```
<body>
<h1>List of customers</h1>
<ul>
<li>Ramesh</li>
<li>Suresh</li>
<li>Mohan</li>
</ul>
<a href="/">Home</a>

</body>
</html>

)""";
response.setContentType("text/html"); //setting mime type
response<<html;

});

bro.listen(6060,[](Error &error) {
//if server is able to bind itself at this port number this lambda expression will get called with no errors
if(error.hasError()){
cout<<error.getError();
return;
}
cout<<"Bro HTTP Server is ready to accept request on port 6060"<<endl;



});


return 0;
}
```

V2- Creating URL Mappings

We'll create a map to store pairs of url and a pointer to a function (whose params are request & and response&) which will get executed when the given url appears or is accessed.

⇒ So when a request arrives, the server will lookup the url in map, and if found it will execute the function kept against that url. However in case of url not found in map, it will be treated as a static resource and will be looked up in resources folder.

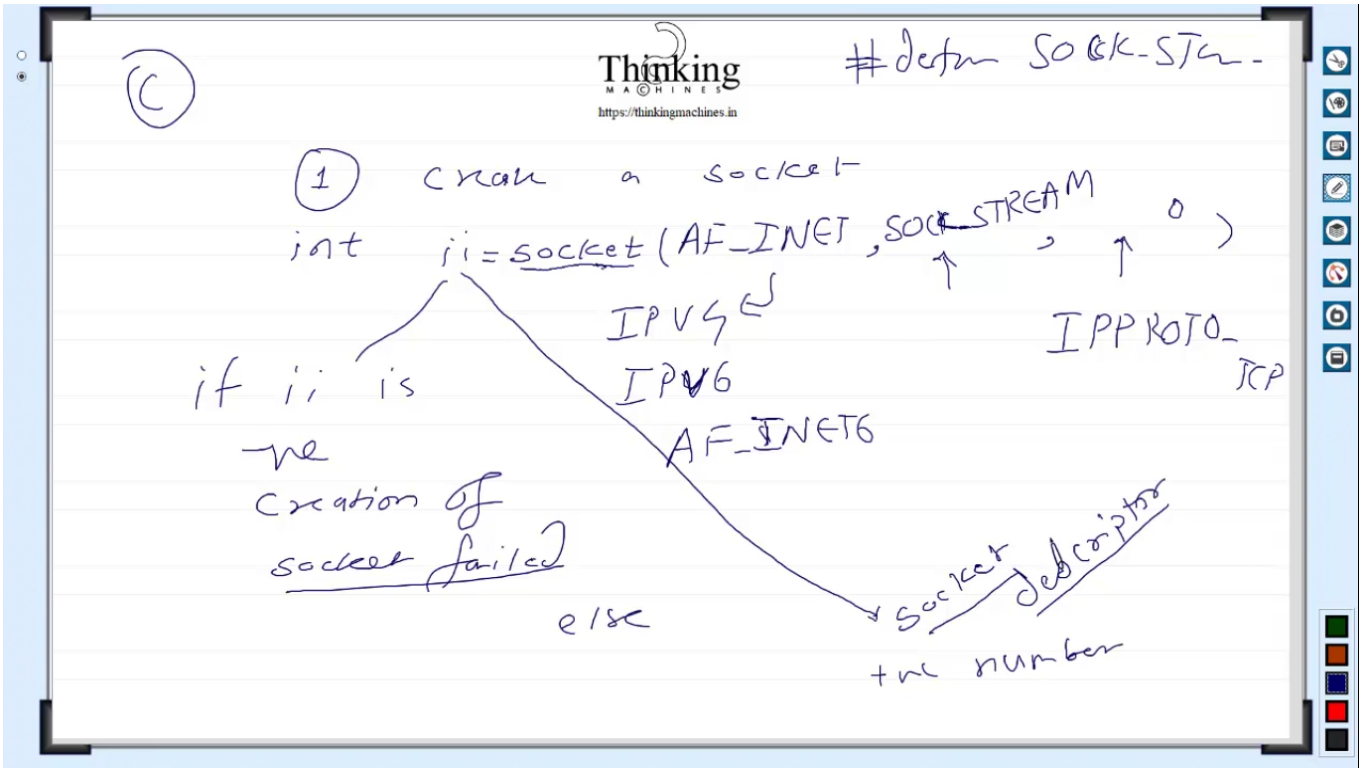
STEPS TO CREATE A CLIENT-SERVER ARCHITECTURE IN C++

Step-1: Create a socket by calling the socket function.
=socket (**family of addresses** that this socket will be addressing (ipv4/v6), **type of stream**, **protocol**)

⇒ Family of addresses = **AF_INET** for IPV4 or **AF_INET6** - for IPV6 support,
⇒ Type of stream = we want the connection to be created in such a way that connection stays live till either network error occurs or the client or server terminates the connection = **sock_stream**
⇒ Protocol = **IPPROTO_TCP** for tcp/ip protocol

⇒ If the functionality behind socket function is successful, a positive number will be returned which will act as a socket descriptor, however, if it returns a negative number, it means socket creation failed.

```
int socketDescriptor= socket (AF_INET,sock_stream,IPROTO_TCP)
```



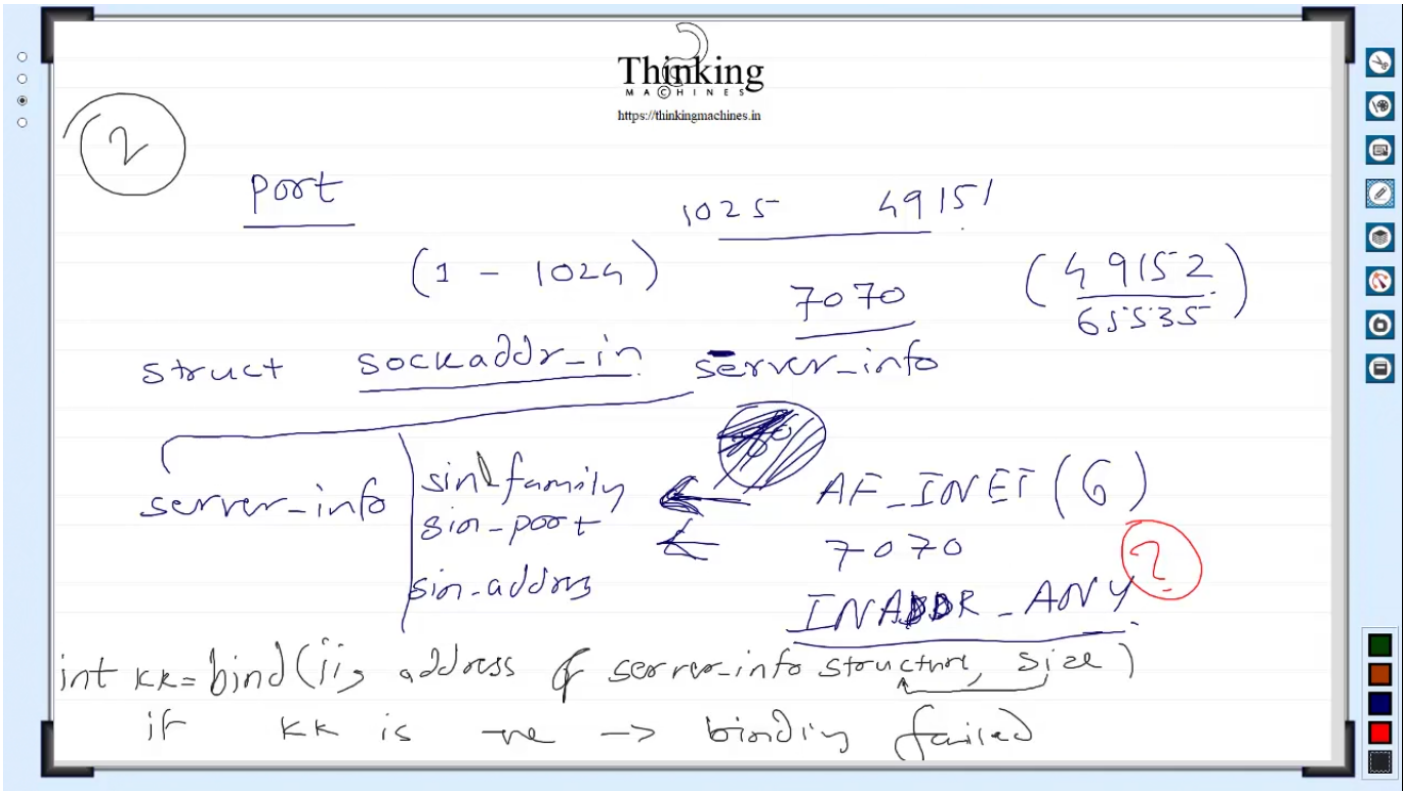
Step-2: We need to bind this socket to a port number
Port (1-1024): reserved for os
Port (49152-65535): dynamic ports
Ports we can use: 1024 - 49151

We'll need to create a struct of type **sockaddr_in**

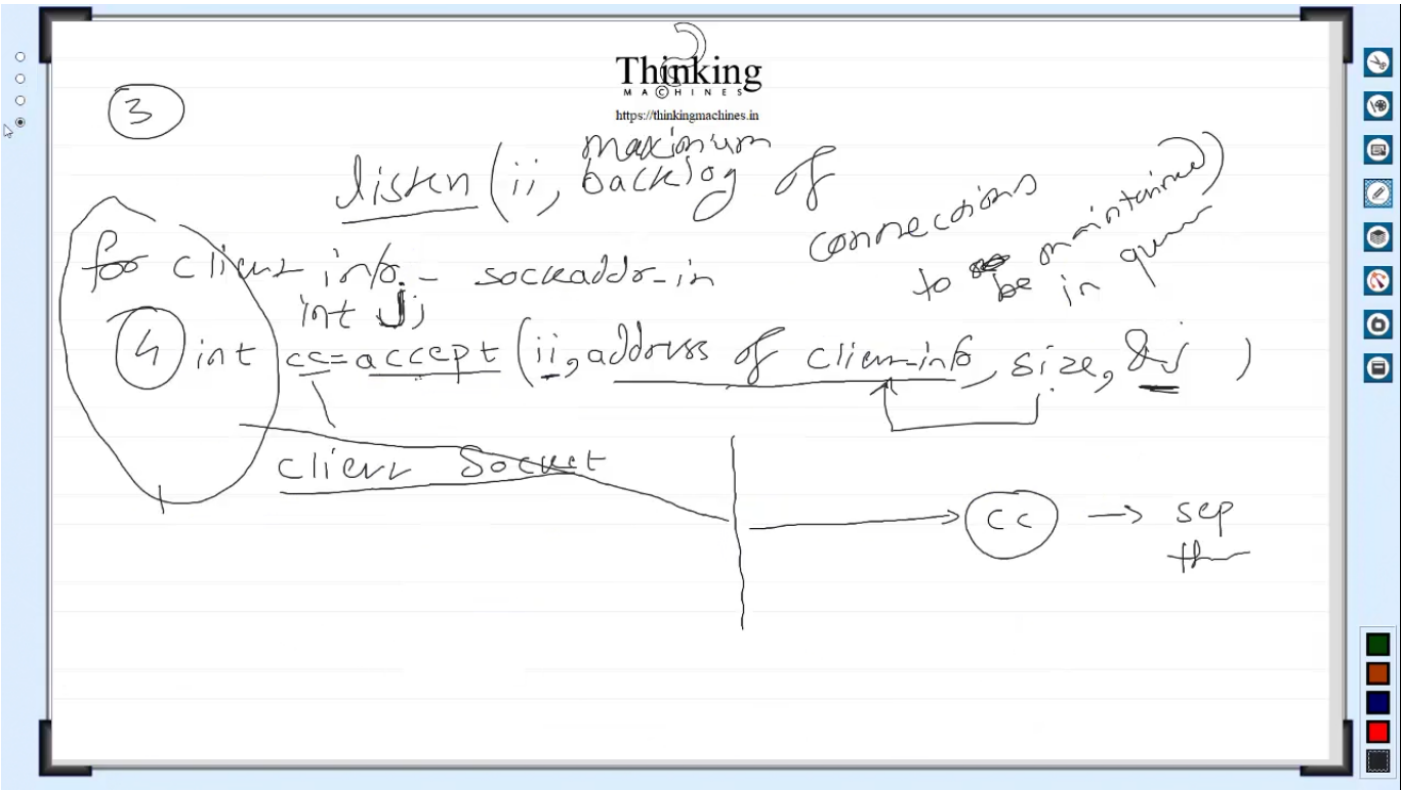
Member of structure
1- sin_family
2-sin_port
3-sin_addr

Then we'll place a call to bind function which again will return an int

int kk = bind(socketDescriptor, address of structure sockaddr_in, size of struct)



Step-3: Marking the socket as one which can accept requests
We'll place call to listen function
= listen (socket_descriptor, maximum backlog of requests in queue,)



Step-4: Accepting requests
int client_socket=accept(socket_descriptor, struct of type sockaddr_in for client info, size of client_info struct, address of int which is going to specify the actual size of client_info struct since it will be populated by accept function)

Video-3:
The concept of Host byte, htonl, Starting the Server

We know that in c++, the int occupies 4-byte size in memory equating to 32 bits of which the most significant bit is a signed bit.

The range for int is from (-2^31 to 2^31 - 1)

Question: How do we calculate binary for 7?

2^4	**2^3**	**2^2**	**2^1**	**2^0**
16	**8**	**4**	**2**	**1**

Ans: The binary of seven will be represented by turning on all the bits which are after 8 i.e. 2^3

Similarly if we know that 2^31 = 2147483648, and the range of int is one less than this number.

So to store 2147483648-1 = 2147483647 in memory we'll have to turn on all bits after 2^31

Thus it will be represented in memory by:
01111111111111111111111111111111 i.e. 0 followed by 31-1's

However these bits are not going to be stored in the memory in the order in which they appear. They will be stored in a completely different order.

So we'll divide the bits in 4 bytes

01111111 11111111 11111111 11111111
Group-1 Group-2 Group-3 Group-4

int	byte-1	byte-2	byte-3	byte-4
	Group-4	Group-3	Group-2	Group-1

Thus group 1 will go to last byte and group-4 will come to first byte

But how do we verify that this is what exactly is happening ? We'll write a program to verify

File Edit Setup Control Window Help

```c
#include<stdio.h>
int main()
{
int x;
char *p;
x=2147483647;
p=(char *)&x;
printf("Address %u\n",&x);
printf("Address %u\n",p);
printf("%d\n",*p);
p++;
printf("Address %u\n",p);
printf("%d\n",*p);
p++;
printf("Address %u\n",p);
printf("%d\n",*p);
p++;
printf("Address %u\n",p);
printf("%d\n",*p);



return 0;
}
```

Output

```
Address 1110797516
Address 1110797516
-1
Address 1110797517
-1
Address 1110797518
-1
Address 1110797519
127
```

How is -1 & 127 printed?

The base address is 1110797516 where group 4 is lying which is 11111111
However since we are using %d to print the number, the first bit will be treated as signed bit and thus 2's complement rule will apply and all bits will be inversed and 1 will be added to the rightmost bit

i.e. 1 1 1 1 1 1 1 1

---

0 0 0 0 0 0 0 0 (bits inversed)
+1 (1 added- 2's complement)

—---

0 0 0 0 0 0 0 1

Thus it is a binary equivalent of 1 and hence it was printed as -1

Question: But how 127?
Answer: The first group is stored at the last byte which is 01111111 and since this amounts to all bit turned on after the 8th bit i.e. 2^7th bit, thus it will amount to (2^7-1) = 127

HTONL Function

```
#include<stdio.h>
#include<arpa/inet.h>
int main()
{
int x;
int y;
char *p;
x=2147483647;
p=(char *)&x;
printf("Address %u\n",&x);
printf("Address %u\n",p);
printf("%d\n",*p);
p++;
printf("Address %u\n",p);
printf("%d\n",*p);
p++;
printf("Address %u\n",p);
printf("%d\n",*p);
p++;
printf("Address %u\n",p);
printf("%d\n",*p);
y=htonl(x);
p=(char *)&y;
printf("Address %u\n",&y);
printf("Address %u\n",p);
printf("%d\n",*p);
p++;
printf("Address %u\n",p);
printf("%d\n",*p);
p++;
printf("Address %u\n",p);
printf("%d\n",*p);
p++;
printf("Address %u\n",p);
printf("%d\n",*p);

return 0;
}
```

What did the HTONL do?
The HTONL grouped the bits in the opposite order. This is also called the network byte order.

The code won't compile in windows as windows does not have arpa/inet.h header file. Instead we'll include windows.h and compile using
g++ bro.cpp -o bro.exe -lws2_32

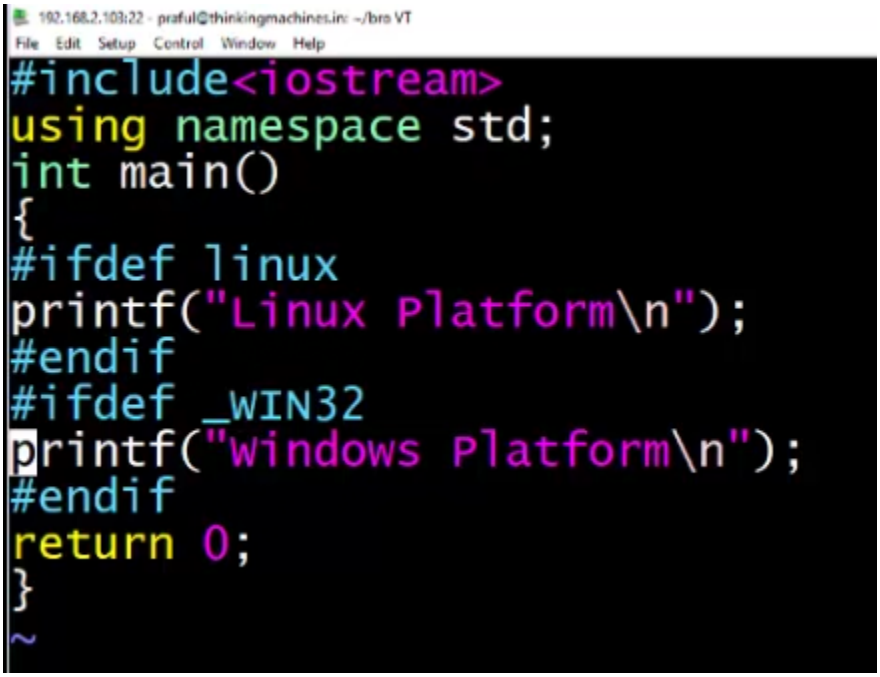
After adding this the code will compile, but the server wont bind itself to the socket.
So we need to add the following lines in the listen function:

```
// for windows - to initialize socket api
WSADATA wsaData;
WORD ver;
ver = MAKEWORD(1,1);
WSAStartup(ver,&wsaData);
```

Also, after closing the socket, we'll add the following line: **WSACleanup();**

Video-4

Writing portable code



Video-5

Bug in the recv read part and implementing mechanism to parse request

When we run the code at this moment, as soon as we send request from the browser, we will see no response but the browser will keep on showing the loading circle. This is because the recv function inside accept is waiting to receive bytes but since browser is not sending any, recv is stuck in the infinite loop.

When we click cancel request on browser, the recv receives 0 bytes and it gets out of that loop.

Thus our strategy would be to parse the request header and learn in advance how many bytes are we receiving and after reading all those bytes, we'll break out of loop manually.