Socket API

An API(Application Program Interface) in general is created in a network to provide proper communication between the elements of the network and provide some set of rules by which the different elements in the network are restricted access.
Example:
The use of socket APIs in a college campus.
We would have a network topology in the college network, this topology would have different access requests from say a faculty is asking for a list of students in a batch, the network takes the request from client to the server (following the network protocols), particularly a socket where the request is be seen by the server.
The server itself has a program that will have the database set and return the list of students, this is where the socket comes into play the request's authority for that particular information, etc. are seen and then the information is passed to the socket where the request from the client was received.
Server itself will have multiple ports and these ports itself can have multiple socket listeners to them. The sockets here play the role of receiving the requests from particular (some application that recognises these sockets).
By using the socket API in this case we are able to communicate between two different computers, even different softwares they could be working on and send them along the network.

Speaking more formally, the Socket API definition as I interpreted from **IBM website** is this:

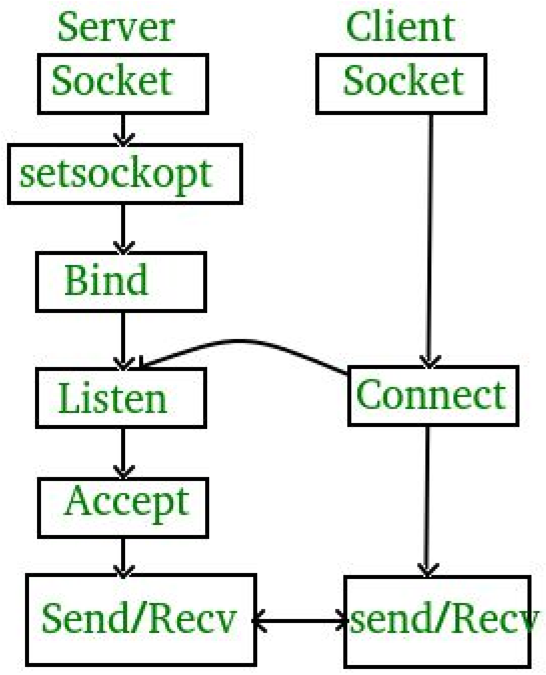
The socket API is a group of socket file descriptors(the socket call used to create a socket in C return an integer socket file descriptor) that enable users to perform primary communication functions between application programs. These communication may include:

- Set up and establish connections to other users on the network
- Send and receive data to and from other users
- Close down connections if required

The API also enables users to:

- Request the network system to provide names and status of relevant resources on the network. This can be an advantage if the user is not familiar with the network and its resources.
- System and control functions as required.

Sockets in C/C++



taken from geeksforgeeks.org

This image shows the socket working in a two way connection.

The socket creation: socket().

#include <sys/types.h>
#include <sys/socket.h>

int sockfd = socket(domain, type, protocol)

Domain is whether the socket follows IPv4 or IPv6, type is choosing between UDP or TCP, whereas the protocol we give 0 for IP.

bind()

#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd, struct sockaddr *myaddr, int addrlen)

This is a server particular usability function, where the client can use this but the function of the binding is not particularly useful for the client.

The structure sockaddr is custom data structure (available in netinet.h). It is used to store the port, address of the sockets, IP version data and even in choosing the port for the socket.

The arguments in **bind()** are the socket number in int, address for sockaddr type structure and length of the sock_addr block.

The function of bind is making the server listen:

1. Make the server listen to the socket its bound to, so that other servers can’t bind to this server.The connectionless or connection-oriented both types of communication need this.
2. It can also be used to check the availability of the port for the socket.

connect()

```
#include <sys/types.h>
#include <sys/socket.h>
```

```
int connect(int sockfd, struct sockaddr *myaddr, int addrlen)
```

The connect is to a client what bind is to the server.

The connect has the same arguments for with client sock_addr instead of server.

The function of the **connect()** is :

1. Check connection between socket and client.

listen()

```
#include <sys/types.h>
#include <sys/socket.h>
int listen(int sockfd, int backlog);
```

The server passively waits for the client to pass a message to the socket. The server socket (**sockfd**) and the length of the connection requests (**backlog**).

accept()

```
#include <sys/types.h>
#include <sys/socket.h>
int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

This where the server actually connects to the client, here from listening to the socket(sockfd) the connection and returns a new socket descriptor. The data can now be transferred between the server and client.

sendto / recvfrom(sockfd, msg, message size, protocol, (struct sockaddr*), length)

The length is sock_addr, msg the char* pointer passed and the message size or more accurately buffer size for reading the message, the protocol passes here is same as the one for which sockfd is created (IP = 0)

close(int sockfd)

This closes the use of completely for both server and client as this is independent whether the socket is used by server or client but the only sees that socket descriptor

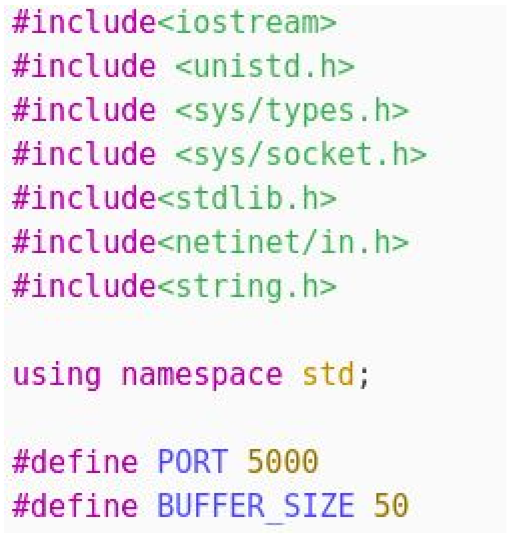
LINUX Commands

- ifconfig -a :
  This returns all available modes of network in the system used. Usually a system has one ethernet and one wireless connection; the return value contains information like the packets sent or received, the information of the IP Address of the ports to which they are connected and the nomenclature of the ports as identified by the system.
- traceroute google.com
  This requests the route taken by the packet of information from source(the place where it's executed) to the destination(in this case google.com). Although the complete route is requested there can be security features used to prevent passing of such sensitive information. This is used to trace out the data loss in the packet path and possible leaks in the security.
- dig google.com
  dig stands for Domain Information Groper is used for querying the DNS requests. It is a useful tool for the DNS diagnosis, it looks for the DNS and returns the answer from the server.
- telnet google.com 443
  There exists TELNET protocol and this commands requests a remote connection from the system to the domain name. This connection can include from data exchange to controlling the other computer in bounds of TELNET protocol.
- nslookup google.com
  This command is used when to find the IP address of the queried DNS, this is more useful for network administrators as even though the results of the query are displayed

but these are displayed taking in mind the system security and usually this can be identified by the “Non-authoritative answer” displayed.

- netstat
  Print network connections, routing tables, interface statistics, masquerade connections, and multicast memberships of the linux system.
- ping google.com
  This is used for testing and diagnostics in a network, the ping sends continuous requests to the destination IP and gives the statistics of the connection.
- Nload
  Nload is a command line tool that allows users to monitor the incoming and outgoing traffic separately.
- iftop
  iftop is a network analyzing tool used by system administrators to view the bandwidth related stats. It shows a quick overview of the networking activities on an interface.

My code: chatRoom.cpp



```
#include<iostream>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include<stdlib.h>
#include<netinet/in.h>
#include<string.h>

using namespace std;

#define PORT 5000
#define BUFFER_SIZE 50
```

```cpp
class client
{
public:
  int client_socket;
  struct sockaddr_in client_address;

  client()
  {
    client_socket = socket(AF_INET, SOCK_DGRAM, 0);
    if (client_socket == 0)
    {
      printf( "Client Socket Failed\n");
      return;
    }
    printf( "Client Socket created\n");
    client_address.sin_family = AF_INET;
    client_address.sin_addr.s_addr = INADDR_ANY;
    client_address.sin_port = htons(PORT);
    if (connect(client_socket, (struct sockaddr *)&client_address, sizeof(client_address)) < 0)
    {
      printf( "Connection Failed from Client\n");
      return;
    }
    printf( "Client connection Success\n");
  }
  int send_to_server(char *msg)
  {
    sendto(client_socket, msg, BUFFER_SIZE, 0, (struct sockaddr*) NULL, sizeof(client_address) );
    printf( "                                    Message Sent from Client:%s\n", msg);
    return 1;
  }
  int receive_from_server()
  {
    char buffer[BUFFER_SIZE];
    recvfrom(client_socket, buffer, sizeof(buffer), 0, (struct sockaddr*)NULL, NULL);
    printf("                                    Message Recieved from Server:%s\n", buffer);
    return 1;
  }
  void close_client()
  {
    close(client_socket);
    printf("Client Socket closed\n");
  }
};
```

```cpp
class server
{
public:
    int server_socket;
    struct sockaddr_in server_address;
    int new_socket;
    server()
    {
      server_socket = socket(AF_INET, SOCK_DGRAM, 0);
      if (server_socket == 0)
      {
        printf( "Server Socket Failed\n");
        return;
      }
      printf( "Server Socket created\n");
      server_address.sin_family = AF_INET;
      server_address.sin_addr.s_addr = INADDR_ANY;
      server_address.sin_port = htons(PORT);
      if (bind(server_socket, (struct sockaddr*)&server_address, sizeof(server_address)) < 0)
      {
        printf( "Connection Failed from Server\n");
        return;
      }
      printf( "Server bound to PORT\n");
    }
    int receive_from_client(struct sockaddr_in* client_address)
    {
      char buffer[BUFFER_SIZE];
      unsigned int len = sizeof(*client_address);
      recvfrom(server_socket, buffer, BUFFER_SIZE, 0, (struct sockaddr*)client_address, &len);
      printf("                                    Message Recieved from Client:%s\n", buffer);
      return 1;
    }
    int send_to_client(char *msg, struct sockaddr_in* client_address)
    {
      int len = sizeof(*client_address);
      sendto(server_socket, msg, BUFFER_SIZE, 0, (struct sockaddr*)client_address, len);
      printf("                                    Message Sent from Server:%s\n", msg);
      return 1;
    }
    void close_server()
    {
      close(server_socket);
      printf("Server Socket closed\n");
    }
};
```

```
int main()
{
  server *s = new server();
  client *c = new client();
  char msgSent1[100];
  char msgSent2[100];
  char bye[] = "bye";
  while(strcmp(bye, msgSent1) && strcmp(bye, msgSent2))
  {
    printf("Client: ");
    fflush(stdin);
    cin.getline(msgSent1, 100);
    c->send_to_server(msgSent1);
    s->receive_from_client(&(c->client_address));
    printf("Server: ");
    fflush(stdin);
    cin.getline(msgSent2, 100);
    s->send_to_client(msgSent2, &(c->client_address));
    c->receive_from_server();
  }
  s->close_server();
  c->close_client();
  return 0;

}
```

For the link to the file (containing the code for file not in):
https://docs.google.com/document/d/1IkyD-VAJmlMctRcfvxG1ILe1n-ansvT3HCeGdBRphwc/edit?usp=sharing