Socket API in C Programming Language

- What is a socket?
  - The **point** where a **local application process** attaches to the **network**
  - An **interface** between an **application** and the **network**
  - An application creates the socket
- The interface defines operations for
  - Creating a socket
  - Attaching a socket to the network
  - Sending and receiving messages through the socket
  - Closing the socket

Socket

- Socket Family
  - PF_INET denotes the Internet family
  - PF_UNIX denotes the Unix pipe facility
  - PF_PACKET denotes direct access to the network interface (i.e., it bypasses the TCP/IP protocol stack)
- Socket Type
  - SOCK_STREAM is used to denote a byte stream
  - SOCK_DGRAM is an alternative that denotes a message oriented service, such as that provided by UDP

Creating a Socket

```
int sockfd = socket(address_family, type,
  protocol);
```

- The socket number returned is the socket descriptor for the newly created socket

- `int sockfd = socket (PF_INET, SOCK_STREAM, 0);`
- `int sockfd = socket (PF_INET, SOCK_DGRAM, 0);`

  The combination of PF_INET and SOCK_STREAM implies TCP

Client-Serve Model with TCP

Server

- Passive open
- Prepares to accept connection, does not actually establish a connection

Server invokes

```
int bind (int socket, struct sockaddr *address,
                                        int addr_len)
int listen (int socket, int backlog)
int accept (int socket, struct sockaddr
*address,
                                    int *addr_len)
```

Client-Serve Model with TCP

Bind

- Binds the newly created socket to the specified address i.e. the network address of the local participant (the server)
- Address is a data structure which combines IP and port

Listen

- Defines how many connections can be pending on the specified socket

Client-Serve Model with TCP

Accept

- Carries out the passive open
- Blocking operation
  - **Does not return** until a remote participant has established a connection
  - When it does, it returns a new socket that corresponds to the new established connection and the address argument contains the remote participant's address

Client-Serve Model with TCP

Client

- Application performs active open
- It says who it wants to communicate with

Client invokes

```
int connect (int socket, struct sockaddr
*address,
                                    int addr_len)
```

Connect

- **Does not return** until TCP has successfully established a connection at which application is free to begin sending data
- Address contains remote machine's address

Client-Serve Model with TCP

In practice

- The client usually specifies only remote participant's address and let's the system fill in the local information
- Whereas a server usually listens for messages on a well-known port
- A client does not care which port it uses for itself, the OS simply selects an unused one

Client-Serve Model with TCP

Once **a connection is established**, the application process invokes two operation

```
int send (int socket, char *msg, int msg_len,
                                        int flags)

int recv (int socket, char *buff, int buff_len,
                                           int
flags)
```

Example Application: Client

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 5432
#define MAX_LINE 256

int main(int argc, char * argv[])
{
        FILE *fp;
        struct hostent *hp;
        struct sockaddr_in sin;
        char *host;
        char buf[MAX_LINE];
        int s;
        int len;
        host = argv[1];
```

Example Application: Client

```
    /* translate host name into peer's IP address */
    hp = gethostbyname(host);

    /* build address data structure */
    bzero((char *)&sin, sizeof(sin));
    sin.sin_family = AF_INET;  /* Internet Address*/
    bcopy(hp->h_addr, (char *)&sin.sin_addr, hp->h_length);
    sin.sin_port = htons(SERVER_PORT);

    /* active open  PF_INET is protocol family*/
    fill your code the create the socket;
    fill your code the connect the socket with server;

    /* main loop: get and send lines of text */
    while (fgets(buf, sizeof(buf), stdin)) {
            buf[MAX_LINE-1] = '\0';
            len = strlen(buf) + 1;
            send(s, buf, len, 0);
    }
}
```

Example Application: Server

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define SERVER_PORT 5432
#define MAX_PENDING 5
#define MAX_LINE 256

int main()
{
        struct sockaddr_in sin;
        char buf[MAX_LINE];
        int len;
        int s, new_s;

        /* build address data structure */
        bzero((char *)&sin, sizeof(sin));
        sin.sin_family = AF_INET;
        sin.sin_addr.s_addr = INADDR_ANY;
        sin.sin_port = htons(SERVER_PORT);

        /* setup passive open */
        fill your code here to create the socket
```

Example Application: Server

```
    fill your code to bind the socket to the address data structure;
    fill your code to make the socket to listen;

    /* wait for connection, then receive and print text */
    while(1) {
            fill your code to accept connections;
            fill your code to receive and print text;
    }
}
```