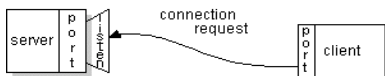


## Socket Programming

## Assignment

- What is a socket ?  
A socket is a communication mechanism.  
A socket is normally identified by a small integer which called the socket descriptor.
- Ports – like a channel in a network.
- Server side sockets – switchboard operator.
- Client side sockets – end point of conversation.



## System calls

- Socket() – creates new socket  

```
int socket(int domain, int type, int protocol);
```

Domain – Usually set to PF\_INET.  
Type – Set to SOCK\_STREAM or SOCK\_DGRAM.  
Protocol – Set to 0.

```
sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

- **Bind()** - Associate the socket with port.

```
int bind(int sockfd, struct sockaddr *serv_addr,
        int addrlen);
```

sockfd - Socket descriptor returned by the socket call  
 serv\_addr - structure containing the local details  
 addrlen - size of the structure.  
 Returns -1 on error.

The port number can be any number between 2000 and 65535.

```
bind(sockfd, (struct sockaddr *) &serv_addr,
      sizeof(serv_addr))
```

- **Connect()** – Connect to a remote host.

```
int connect(int sockfd, struct sockaddr *serv_addr,
           int addrlen);
```

Returns -1 on error

eg : connect(sockfd,&serv\_addr,sizeof(serv\_addr))

- **Listen()** – Wait for incoming connection

```
int listen(int sockfd, int backlog);
```

backlog – Number of connections allowed on pending queue

Returns -1 on error

eg : listen(sockfd,5);

- **Accept()** – Accept the connection queued up on the pending queue.

```
int accept(int sockfd, struct sockaddr *addr,
          socklen_t *addrlen);
```

Returns a new file descriptor for use on this connection.

Returns -1 on error.

eg : newsockfd = accept(sockfd, (struct sockaddr \*)  
 &cli\_addr, &cliilen);

## Exchanging messages

- **Read / Write :**

Standard read and write commands to read from and write to a file descriptor.

```
ssize_t read(int fd, const void *buf, size_t count);
```

```
ssize_t write(int fd, const void *buf, size_t count);
```

## Exchanging messages

- **Send()** –

```
int send(int sockfd, const void *msg, int len, int flags);
```

- **Recv()** –

```
int recv(int sockfd, void *buf, int len, unsigned int flags);
```

The above calls can only be made in the connected state

<http://www.penguin-soft.com/penguin/man2/send.html?manpath=/man/man2/send.2.inc>

## Closing Connections

- **close(sockfd)** : Regular unix close function .
- **shutdown(sockfd, flags)** : Same function but more control.

Flags : 0 – Further receives disallowed

1 – Further sends disallowed

2 – Further sends and receives are disallowed ( like close ).

## Structs

```
struct sockaddr {
    unsigned short sa_family; // address family, AF_XXX
    char sa_data[14]; // 14 bytes of protocol address
};
struct sockaddr_in {
    short int sin_family; // Address family
    unsigned short int sin_port; // Port number
    struct in_addr sin_addr; // Internet address
    unsigned char sin_zero[8]; // memset to zero
};
Eg:
server.sin_family = PF_INET;
server.sin_port = htons(portno);
inet_aton("10.12.110.57", &(server.sin_addr));
```

## Handling IP Addresses

- 2 conventions – Big Endian and Little Endian
- Big Endian – MSB first – Networks, Motorola, IBM
- Little Endian – LSB first – intel, VAX
- Exception eg : inet\_addr, gethostbyname
- Take care of sin\_port and sin\_addr
- Address conversions –  
ntohl , htonl, ntohs, htons

- Inet\_aton –  
`int inet_aton(const char *strptr, struct in_addr *addrptr);`  
converts the string pointed to by *strptr* into its 32-bit binary network byte
- Inet\_ntoa –  
`char *inet_ntoa(struct in_addr inaddr);`  
converts a 32-bit binary network byte ordered IPv4 address into its corresponding dotted-decimal string.

### Example – Client server application Client side

1. Create socket
2. Bind port to socket
3. Connect to server
4. Read / write
5. Close the socket at the end

### Server Side

1. Create a socket
2. Bind to a port
3. Listen for incoming connections
4. Accept incoming connections and spawn a new socket descriptor
5. Read / Write on the new socket fd
6. Close the connections

### References and some useful links

- Unix Network programming – Richard Stevens
- <http://www.cs.rpi.edu/courses/sysprog/sockets/sock.html>
- <http://beej.us/guide/bgnet/>
- <http://www.amk.ca/python/howto/sockets/>

Questions?