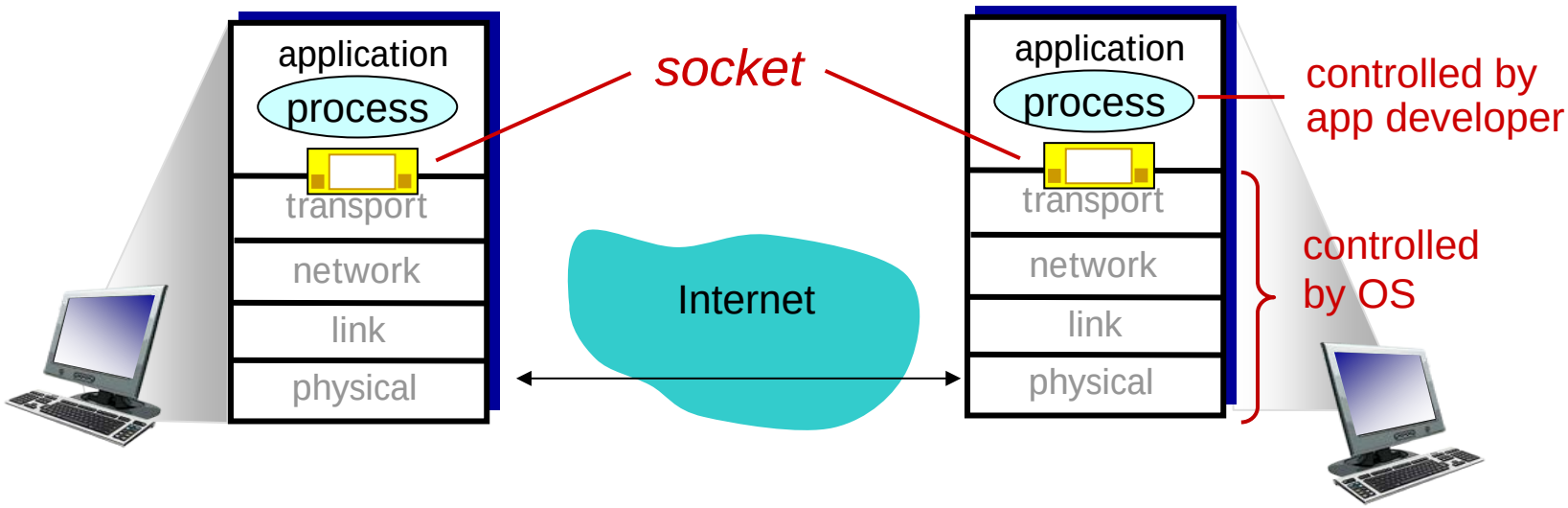
Lecture-2

Socket programming with UDP and TCP

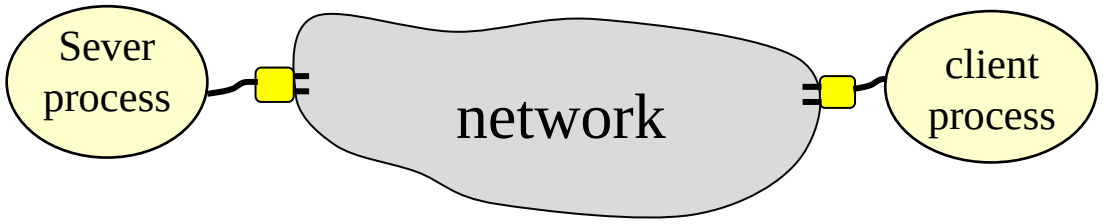
Socket

- Concept was developed in the 1980s in the UNIX environment as the Berkeley Sockets Interface
  - De facto standard application programming interface (API)
  - Basis for Window Sockets (WinSock)
- It allows user can make application programs access networks without profound knowledge of network architecture



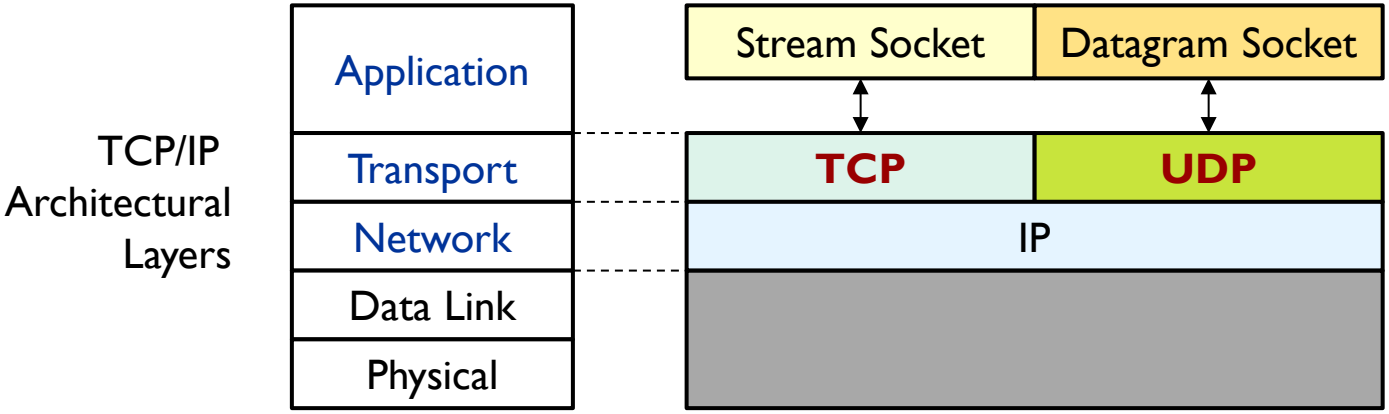
Socket

- Socket extend the convectional UNIX I/O facilities
  - It is provided by the operating system as a form of file descriptor
    - file descriptors for network communication
    - Extended read/write system calls
- Client and Server Communication: Socket enables communication between a client process and a server process
  - Client: initiator of communication
    - Locate the server
  - Server: responder
    - Always ready for unscheduled incoming calls, keeping TCP/UDP port open



Socket Types

- Stream sockets (TCP)
  - Reliable : connection-oriented, ordered, error-checked
- Datagram sockets (UDP)
  - Unreliable : connectionless, no guarantee of delivery and ordering



Components of Socket

- A socket is defined by these parameters
  - Protocol (Layer 4)
  - IP Address
  - Port

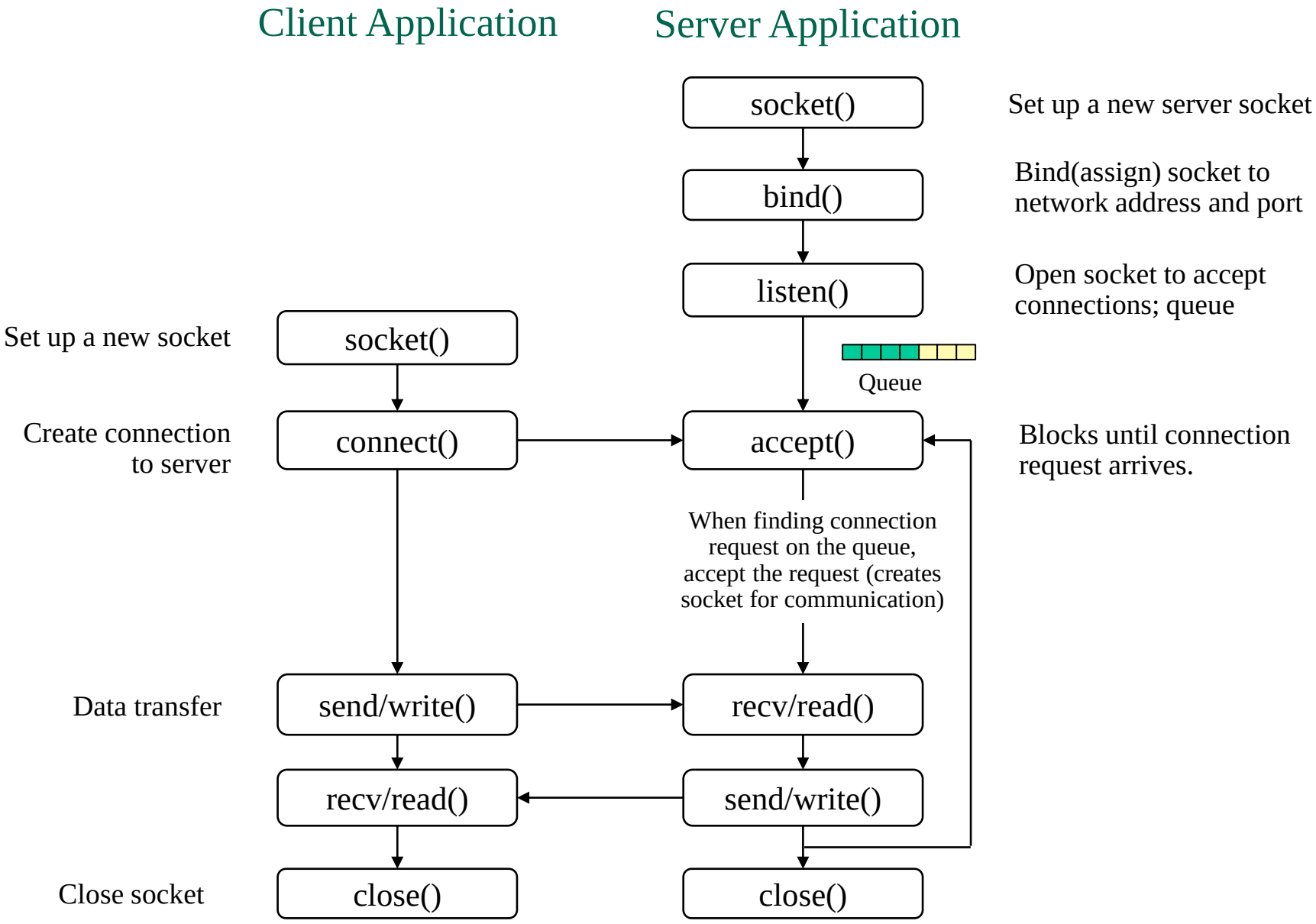
data control

Port Number		Application Layer Protocol	Transport Layer	
			TCP	UDP
0~1023 Well-known port	20, 21	FTP	●	
	22	SSH	●	
	23	TELNET	●	
	25	SMTP	●	
	80	HTTP	●	●

* 1024~49151 : Registered Port (IANA) * 49152 ~ 65535 : Dynamic/Private Port

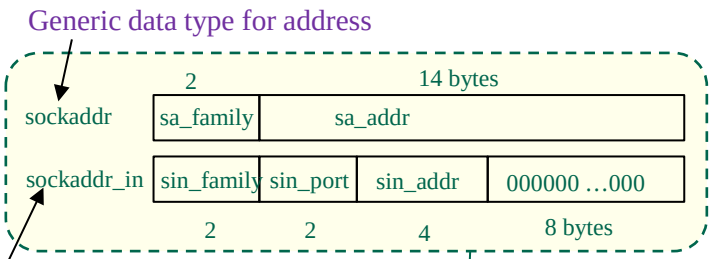
Internet Assigned Numbers Authority

Stream(TCP) Socket Communication

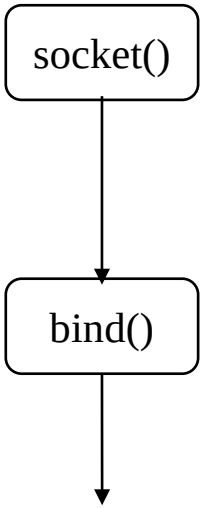


Stream socket comm. Example: Echo Server (1/2)

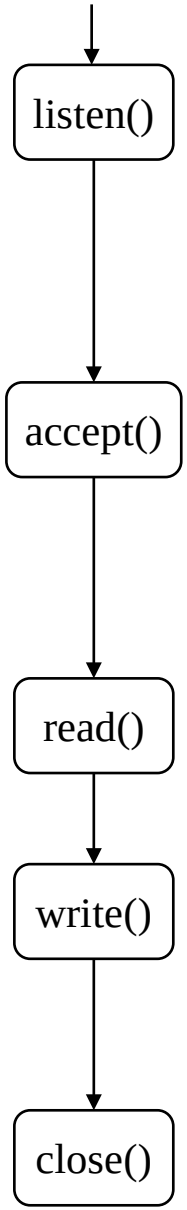
```
#include <sys/socket.h>   // socket, AF_INET, sockadrr, …, etc
#include <arpa/inet.h>    // inet_addr, sockadrr_in, htons, htonl, INADDR_ANY
#include <stdio.h>        // stdin, stdout, fputs
#include <string.h>       // memset
#include <unistd.h>       // read, write, close
#include <stdlib.h>       // exit, EXIT_FAILURE
#define MAXBUF 1024

void error_handling(char *msg) {
    fputs(msg, stderr);
    exit(EXIT_FAILURE);
}

int main(int argc, char **argv)
{
   int server_sockfd, sockfd, pid;
   struct sockaddr_in  server_addr, client_addr;
   socklen_t  client_addr_len;
   char buf[MAXBUF];

   if((server_sockfd = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
      error_handling("socket error\n");
   }
   server_addr.sin_family = AF_INET;
   server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
   server_addr.sin_port = htons(50000);

   if(bind(server_sockfd, (struct sockaddr *)&server_addr,
               sizeof(server_addr)) < 0){
      close(server_sockfd);
      error_handling("bind error\n");
   }
```



htonl(INADDR_ANY)



Stream socket comm. Example: Echo Server (2/2)



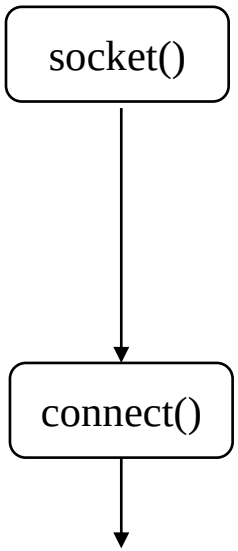

```
if(listen(server_sockfd, 5) < 0) {
   close(server_sockfd);
   error_handling("listen error\n");
}
client_addr_len = sizeof(client_addr);

while(1)
{
    if ((sockfd = accept(server_sockfd,(struct sockaddr *)&client_addr,
                                        &client_addr_len)) < 0){
       close(server_sockfd);
       error_handling("accept error\n");
    }

    memset(buf, 0x00, MAXBUF);
    if(read(sockfd, buf, MAXBUF) <= 0){
       close(sockfd);
       error_handling("read error\n");
    }
A   if(write(sockfd, buf, MAXBUF) <= 0){
       close(sockfd);
       error_handling("write error\n");
    }
    close(sockfd);

}
close(server_sockfd);
return 0;
}
```

- 서버 소켓은 클라이언트 소켓의 연결 요청을 받아들이는 역할만 수행
- 직접적인 데이터 송수신은 서버 소켓의 연결 요청 수락의 결과로 만들어지는 새로운 소켓을 통해 처리

```
int pid=fork();
if (pid==0){  //child process
  close(server_sockfd);
    A
  return 0;
}
else {
  close(sockfd);
  if (pid==-1)
    error_handling("fork error\n");
}
```

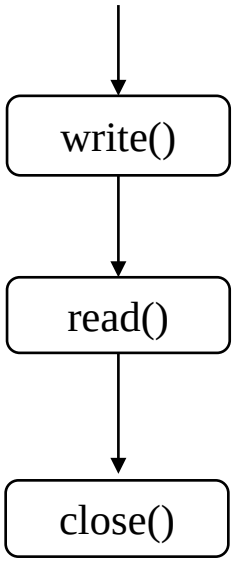
multi process 기반의 다수의 클라이언트 지원

Stream socket comm. Example: Echo Client (1/2)




```
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#define MAXBUF 1024
void error_handling(char *msg);

int main(int argc, char **argv)
{
     struct sockaddr_in server_addr;
     int sockfd;
     socklen_t server_addr_len;
     char buf[MAXBUF];

     if ((sockfd=socket(PF_INET, SOCK_STREAM, 0)) < 0){
         error_handling("socket error\n");
     }

     server_addr.sin_family = AF_INET;
     server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
     server_addr.sin_port = htons(50000);
     server_addr_len = sizeof(server_addr);

     if (connect(sockfd, (struct sockaddr *)&server_addr,
      server_addr_len) < 0){
         close(sockfd);
         error_handling("connect error\n");
     }
```

Stream socket comm. Example: Echo Client (2/2)




```
    memset(buf, 0x00, MAXBUF);
    fgets(buf, MAXBUF, stdin);

    if(write(sockfd, buf, MAXBUF) <= 0){
        close(sockfd);
        error_handling("write error\n");
    }
    if(read(sockfd, buf, MAXBUF) <= 0) {
        close(sockfd);
        error_handling("read error\n");
    }

    close(sockfd);
    return 0;
}


void error_handling(char *msg) {
   fputs(msg, stderr);
   exit(EXIT_FAILURE);
}
```

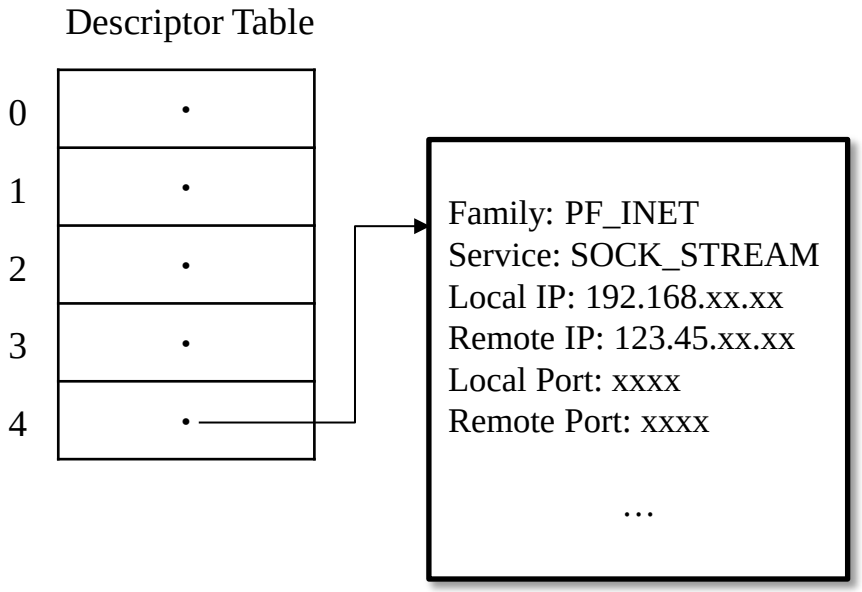
Server-side Socket Programming

❖ socket( )

Protocol Family Type Protocol

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

- Return value : **File Descriptor**
  - -1 : error
  - more than 0 : successfully created



- Family
  - PF_INET : IPv4
  - PF_INET6 : IPv6
  - PF_UNIX : between processes on one system
- Type
  - SOCK_STREAM : TCP
  - SOCK_DGRAM : UDP
  - SOCK_LAW: allows a direct-access to lower-layer protocols
- Protocol
  - 0 : default (Type에서 지정된 것)
  - IPPROTO_TCP: TCP
  - IPPROTO_UDP: UDP
  - is to allow more than one transport-level protocol in a future implementation.

Server-side Socket Programming

❖ sockaddr_in

localhost (loopback)

```
struct sockaddr_in server_addr;

server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
server_addr.sin_port = htons(50000);
```

- sockaddr_in structure used to store addresses with IPv4 address family

  AF_INET
- member variable
  - sin_family
  - sin_addr.s_addr : 32 bits IP address
    - inet_addr( ) : converts a string containing an IPv4 dotted-decimal address into a proper address ↔ inet_ntoa( )
  - sin_port : 16 bits port number
    - htons( ) : converts the unsigned short integer from host byte order to network byte order.

Server-side Socket Programming

❖ Network Byte Order

- According to CPU, Little Endian or Big Endian

Host byte order

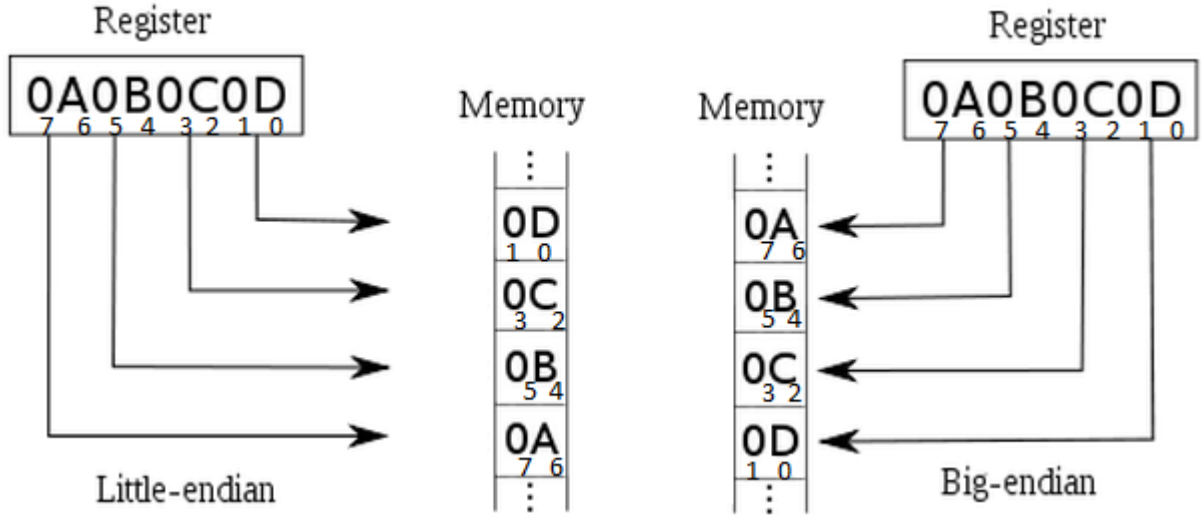
htons →

← ntohs

Network byte order

- Big Endian

unsigned short integer



unsigned short integer

Server-side Socket Programming

❖ bind( )

```
bind(server_sockfd,(struct sockaddr *)&server_addr,
     sizeof(server_addr));
```

- Return value
  - -1 : error
  - 0 : successfully bound

  * The same applies to listen( ), accept ( ), read ( ), write ( ), and connect ( ).

❖ listen( )

```
listen(server_sockfd, 5);
```

- Backlog : The number of connections to queue

Server-side Socket Programming

❖ accept( )

```
int sockfd;
struct sockaddr_in client_addr;
socklen_t client_addr_len;
...
client_addr_len = sizeof(client_addr);
sockfd = accept(server_sockfd, (struct sockaddr *)
                &client_addr, &client_addr_len);
```

- accept( ) blocks the process until a connection request arrives
- It creates a new socket(file descriptor) that contains client's IP address and port number from the queue

❖ close( )

```
close(server_sockfd);
```

Server-side Socket Programming

❖ read( ) / write( )

```
#define MAXBUF 1024
...
char buf[MAXBUF];
...
memset(buf, 0x00, MAXBUF);

if(read(sockfd, buf, MAXBUF) <= 0) {
   close(client_sockfd);
}
...
if(write(sockfd, buf, MAXBUF) <= 0) {
   close(client_sockfd);
}
```

- read( ) and write( ) blocks the process until data arrives to file descriptor
- memset( ) : initialize a block of memory to a specified value.

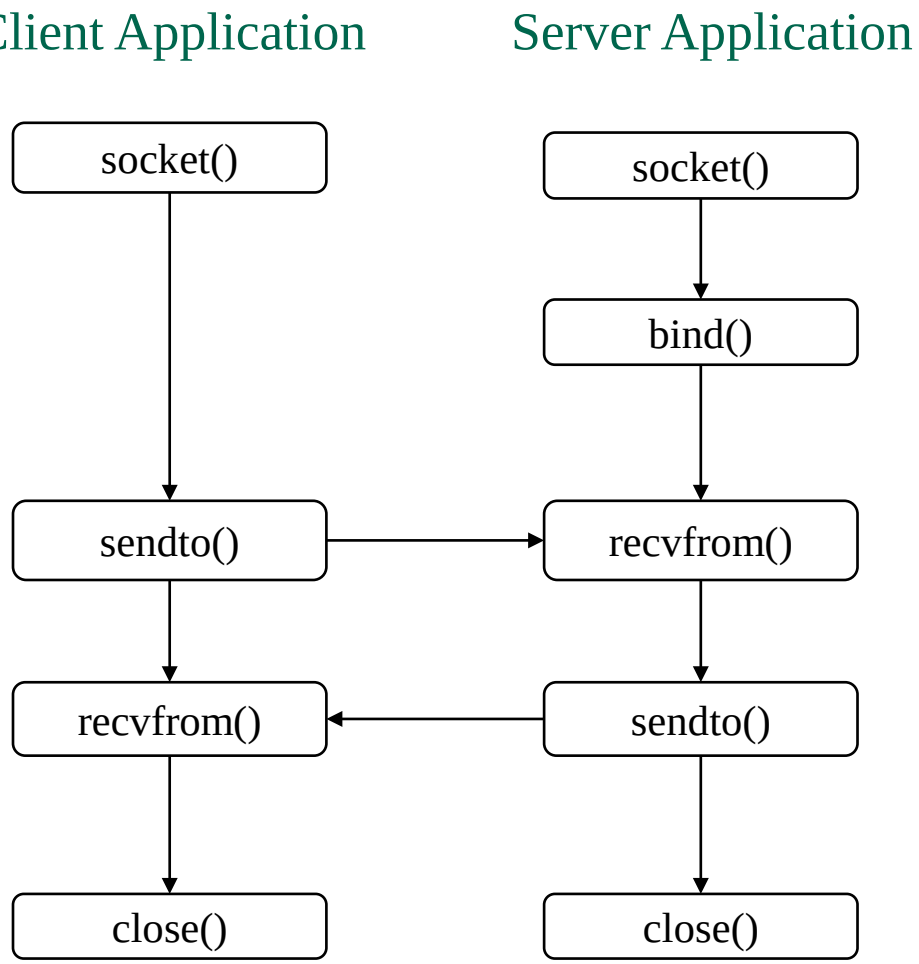
Client-side Socket Programming

❖ connect ( )

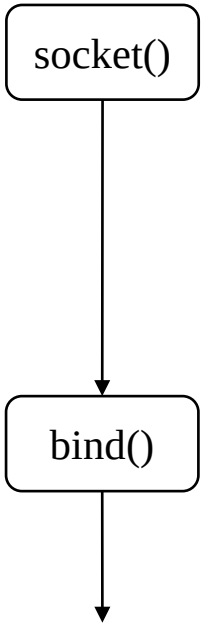
```
int sockfd;
struct sockaddr_in server_addr;
socklen_t server_addr_len;
...
server_addr_len = sizeof(server_addr);
connect(sockfd, (struct sockaddr *)&server_addr, server_addr_len);
```

- Client needs only to create socket and connection (without binding)
- Return value :
  - -1 : error
  - 0 : successfully connected

Datagram(UDP) Socket Communication

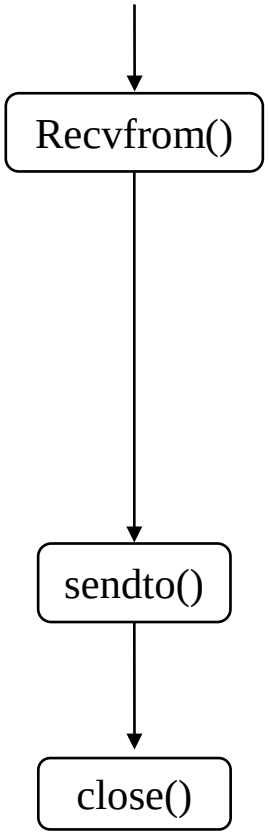


Datagram socket comm. Example: Echo Server (1/2)




```
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAXBUF 1024

int main(void){
    int socket_sd;
    struct sockaddr_in server_addr, client_addr;
    char server_message[MAXBUF], client_message[MAXBUF];
    int client_struct_length = sizeof(client_addr);

    memset(server_message, 0x00, MAXBUF);
    memset(client_message, 0x00, MAXBUF);

    socket_sd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if(socket_sd < 0){
        printf("Error while creating socket\n");
        return -1;
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(2000);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    if(bind(socket_sd, (struct sockaddr*)&server_addr,
            sizeof(server_addr)) < 0){
        printf("Couldn't bind to the port\n");
        return -1;
    }
```

Datagram socket comm. Example: Echo Server (2/2)

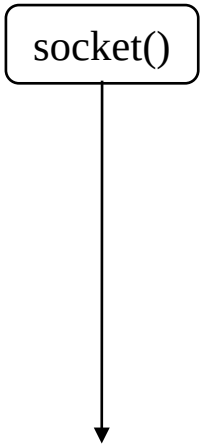



```
printf("Listening for incoming messages...\n\n");

if (recvfrom(socket_sd, client_message, sizeof(client_message), 0,
       (struct sockaddr*)&client_addr, &client_struct_length) < 0){
    printf("Couldn't receive\n");
    close(socket_sd);
    return -1;
 }

 printf("Received message from IP: %s and port: %i\n",
       inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));
 printf("Msg from client: %s\n", client_message);

 if (sendto(socket_sd, client_message, strlen(client_message), 0,
       (struct sockaddr*)&client_addr, client_struct_length) < 0){
    printf("Can't send\n");
    close(socket_sd);
    return -1;
 }

 close(socket_sd);
 return 0;
}
```

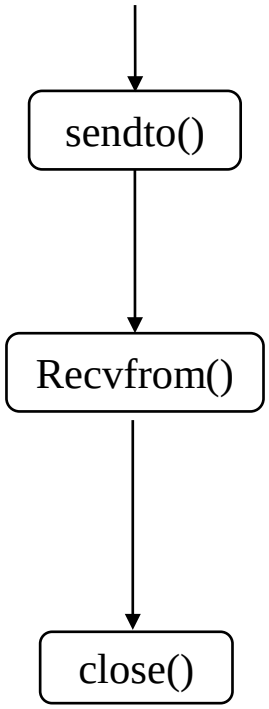
The server receives information about the client when it receives data using the recvfrom()

Datagram socket comm. Example: Echo Client (1/2)

```
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAXBUF 1024

int main(void){
    int socket_cd;
    struct sockaddr_in server_addr;
    char server_message[MAXBUF], client_message[MAXBUF];
    int server_struct_length = sizeof(server_addr);

    memset(server_message, 0x00, MAXBUF);
    memset(client_message, 0x00, MAXBUF);

    socket_cd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

    if(socket_cd < 0){
        printf("Error while creating socket\n");
        return -1;
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(2000);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
```



Datagram socket comm. Example: Echo Client (2/2)




```
    gets(client_message);

    if(sendto(socket_cd, client_message, strlen(client_message), 0,
         (struct sockaddr*)&server_addr, server_struct_length) < 0){
        printf("Unable to send message\n");
        return -1;
    }

    if(recvfrom(socket_cd, server_message, sizeof(server_message), 0,
         (struct sockaddr*)&server_addr, &server_struct_length) < 0){
        printf("Error while receiving server's msg\n");
        return -1;
    }

    printf("Server's response: %s\n", server_message);

    close(socket_cd);
    return 0;
}
```

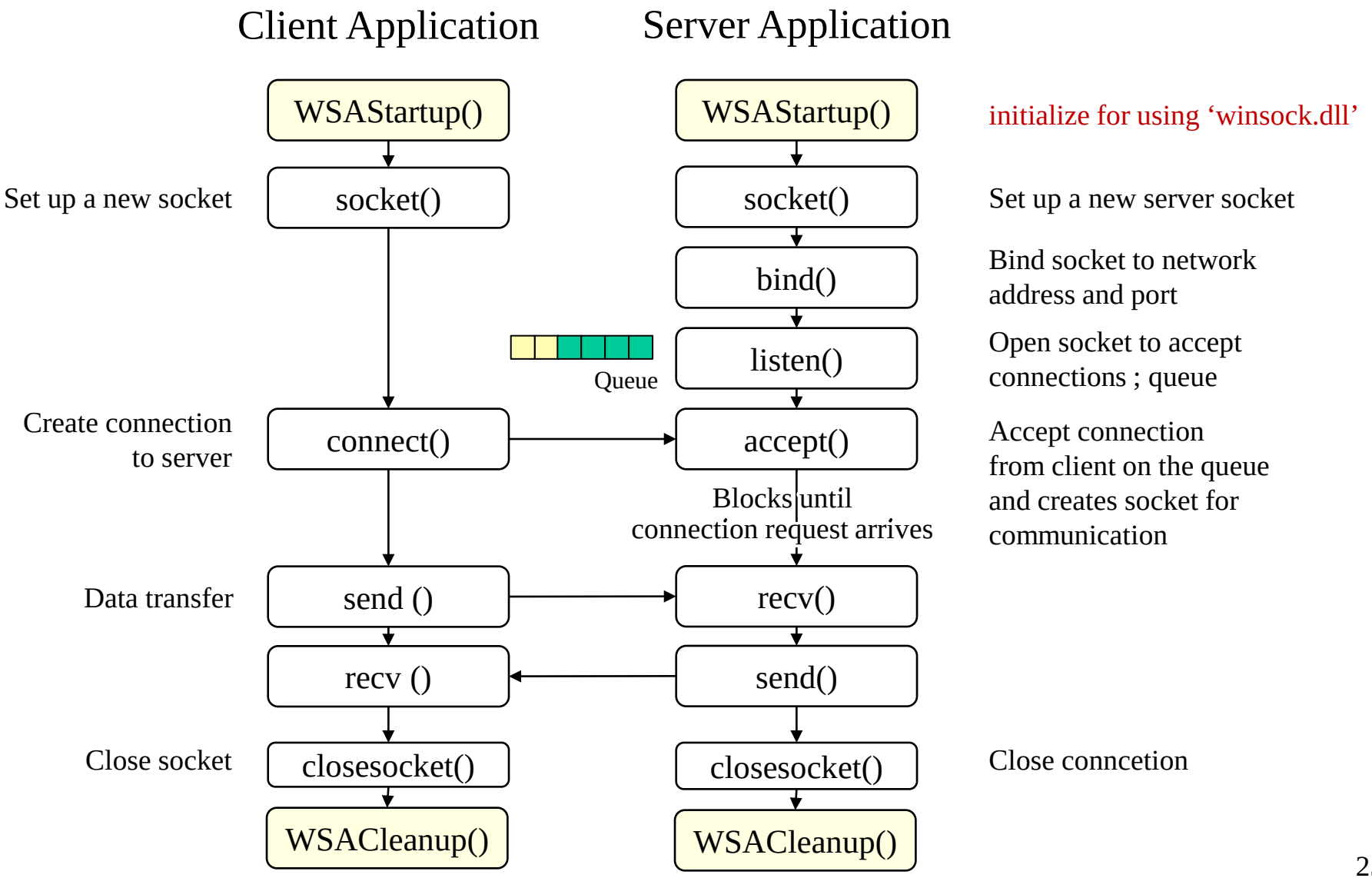
Linux C Based Socket API

Format	Function	Parameters
socket ()	Initialize a socket	**int domain** : Protocol family of the socket to be created (AF_INET, AF_INET6) **int type** : Type of socket to be opened (stream, datagram, raw) **int protocol** : Protocol to be used on socket (UDP, TCP)
bind ()	Bind a socket to a port address	**int sockfd** : Socket to be bound to the port address **struct sockaddr *localaddress** : Socket address to which the socket is bound **socklen_t addresslength** : Length of the socket address structure
listen ()	Listen on a socket for inbound connections	**int sockfd** : Socket on which the application is to listen **int queuesize** : Number of inbound requests that can be queued at any time
accept ()	Accept an inbound connection	**int sockfd** : Socket on which the connection is to be accepted **struct sockaddr *remoteaddress** : Remote socket address where the connection was initiated. **socklen_t *addresslength** : Length of the socket address structure
connect ()	Connect outbound to a server	**int sockfd** : Socket on which the connection is to be opened **struct sockaddr *remoteaddress** : Remote socket address where the connection is opened **socklen_t addresslength** : Length of the socket address structure
send/wirte () recv/read ()	Send and receive data on a stream socket	**int sockfd** : Socket across which the data will be sent or read **void* buf** : Data to be sent, or buffer into which the read data will be placed **size_t len** : Length of the data to be written, or amount of data to be read **int flags** : special option, usually just 0 (send, recv)
sendto() recvfrom()	Send and receive data on a datagram socket	**int sockfd** : Socket across which the data will be sent or read **void* buf** : Data to be sent, or buffer into which the read data will be placed **size_t len** : Length of the data to be written, or amount of data to be read **int flags** : special option, usually just 0 **struct sockaddr *remoteaddress** : Remote socket address **socklen_t addresslength** : Length of the socket address structure (sendto) **socklen_t *addresslength**: (recvfrom)
close ()	Close a socket	**int sockfd** : Socket which is to be closed

Socket API of Other Programming Languages

Windows Sockets API (C, C++)
Python

Socket Programming using Windows Sockets API



Socket Programming using Windows Sockets API

Linux C	Windows C++
#include <sys/socket.h>	#include <winsock2.h>
int socket_fd	SOCKET socket
sockaddr_in	SOCKADDR_IN
	WSADATA wsaData; WSAStartup(WORD wVersionRequested, WSADATA* wsaData);
socket( )	socket(PF_INET, SOCK_STREAM, 0);
close( )	closesocket(socket);
	WSACleanup();
read( )/recv()	recv(SOCKET socket, char* buf, int len, int flags);
write( )/send()	send(SOCKET socket, const char* buf, int len, int flags);

Socket Programming using Windows Sockets API

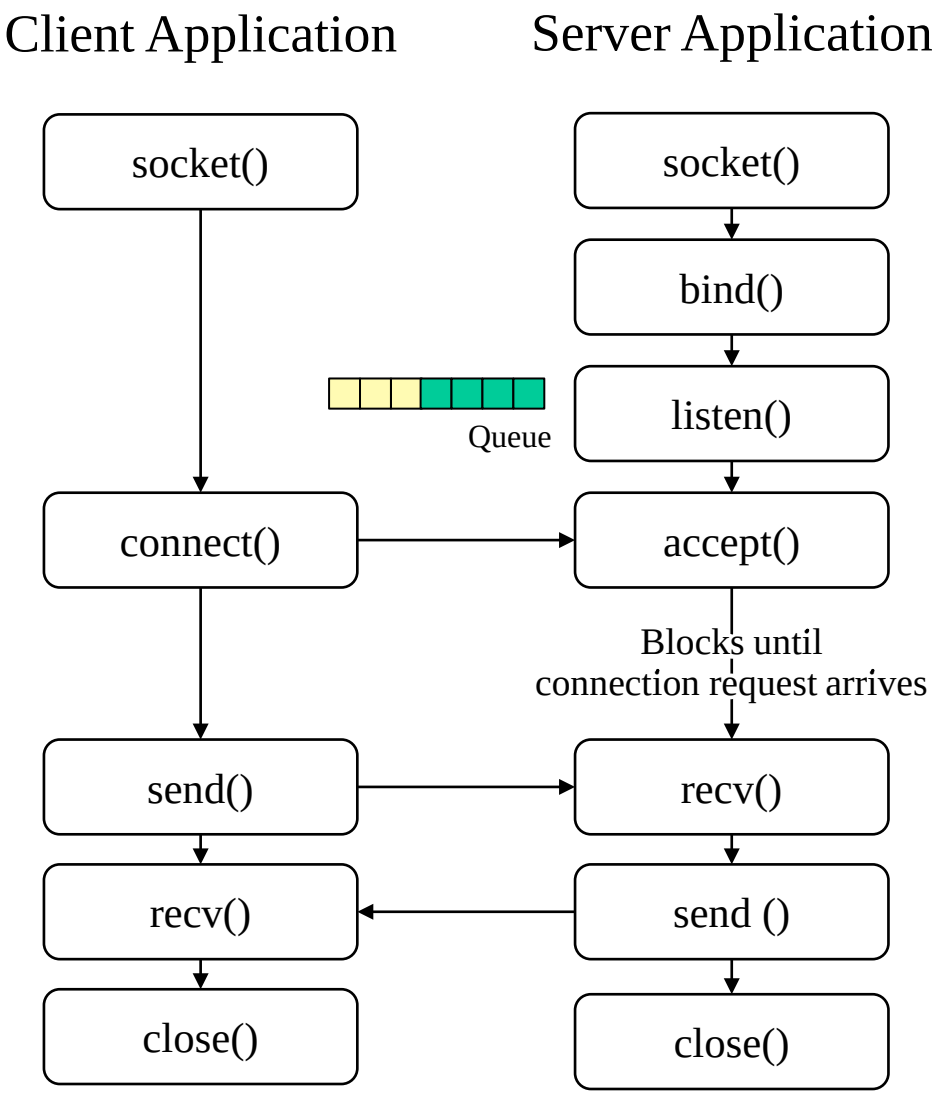
```
SOCKET server_socket = socket(PF_INET, SOCK_STREAM, 0);
```

- Return value : **Socket Handle**
  - Similar to File Descriptor in UNIX System
  - Windows differentiates between socket handle and file handle

```
WSADATA wsaData;
WSAStartup(MAKEWORD(2,2), &wsaData);
WSACleanup();
```

- WSADATA: data structure to contain information about the Windows Sockets implementation
- WSAStartup( )
  - initiates use of the 'winsock.dll' and retrieve details of the specific WinSokcet implementation.
  - specify the version of Windows Sockets required.
    - `MAKEWORD(2,2)` : WinSocket Version 2.2
- WSACleanup( ) : for every WSAStartup() that is successfully called.

Socket Programming in Python



Socket Programming in Python

Linux C	Python
#include <sys/socket.h>	import socket
sockaddr_in int socket_fd socket( )	s_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
bind( )	s_sock.bind((“localhost”, port))
listen( )	s_sock.listen(5)
accept( )	(c_sock, address)= s_sock.accept()
close( )	c_sock.close() s_sock.close()
connect( )	c_sock.connect((host_ip, port))
read()/write( )	data = c_sock.recv(1024) c_sock.send(data)