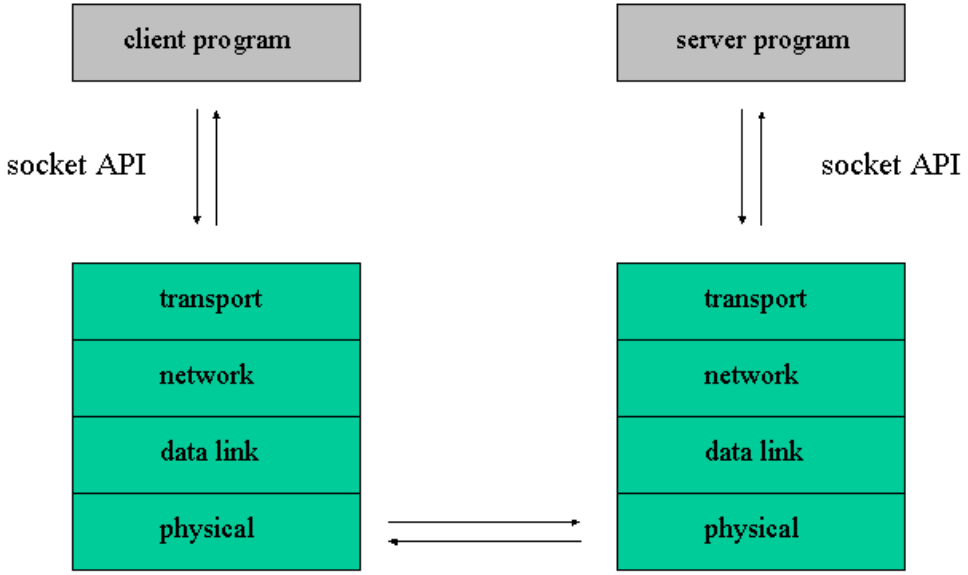
Homework 1

Socket programming

Mobile Computing and Communications Lab.

Socket API

- Socket : An endpoint of an inter-process communication across computer networks.



  - It allows user can make application programs access networks without profound knowledge of network architecture.
  - It is provided by the operating system as a form of file descriptor.

Components of Socket

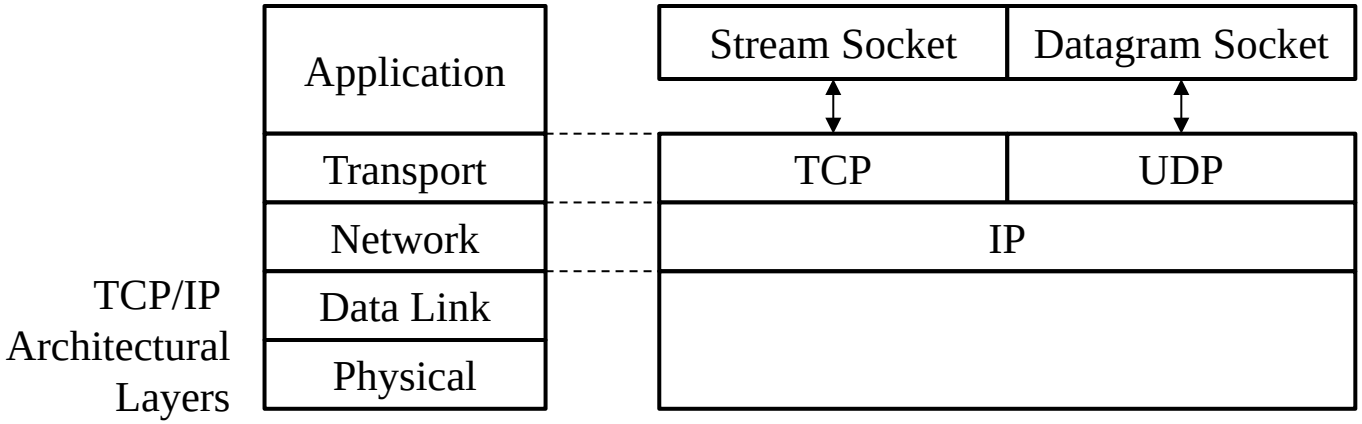
- A socket is defined by these parameters
  - Protocol
  - IP Address
  - Port

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

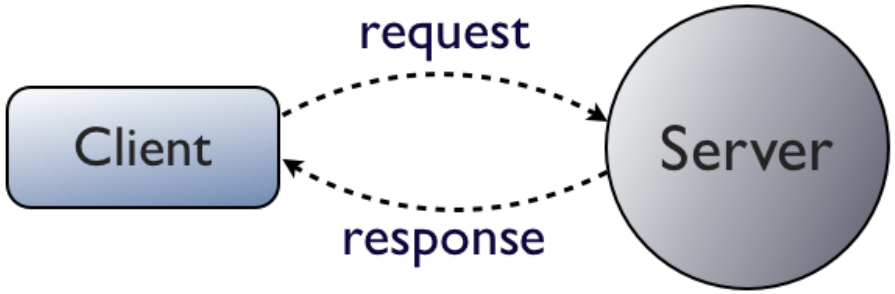
Socket Types

- Stream sockets (TCP)
  - Reliable : connection-oriented, ordered, error-checked
  - 4 tuples (source IP, port, destination IP, port)
- Datagram sockets (UDP)
  - Unreliable : connectionless, no guarantee of delivery and ordering
  - 2 tuples (source IP, port)

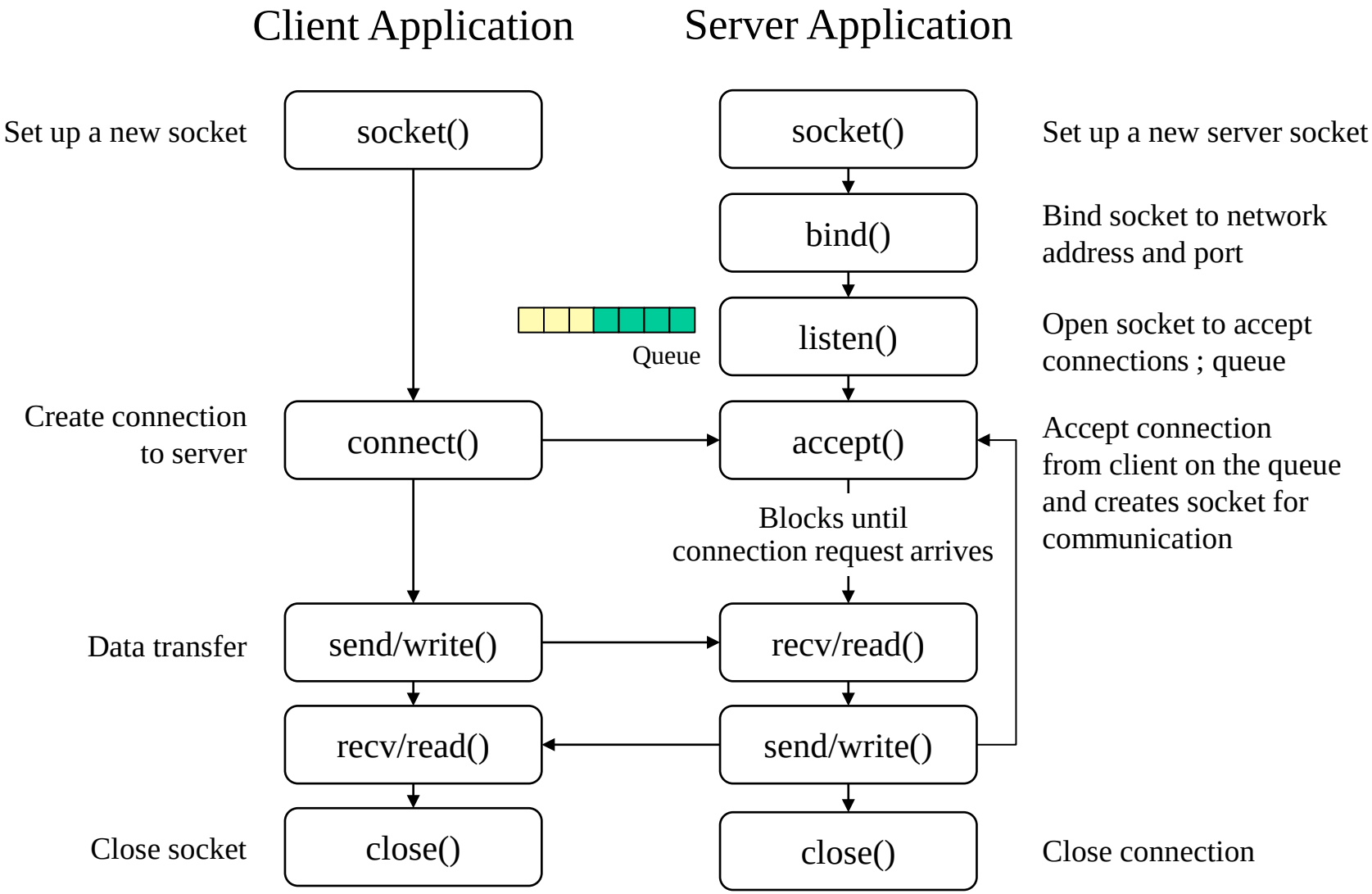


Client and Server Communication

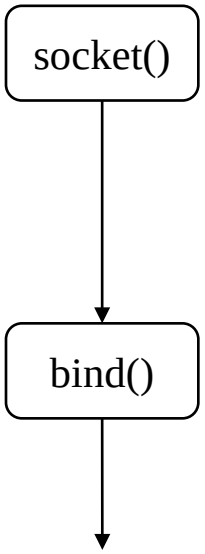
- Socket enables communication between a client and server process
  - Server: responder
    - Always ready for unscheduled incoming calls keeping TCP/UDP port open
  - Client: initiator of communication
    - Locate server



Stream(TCP) Socket Communication

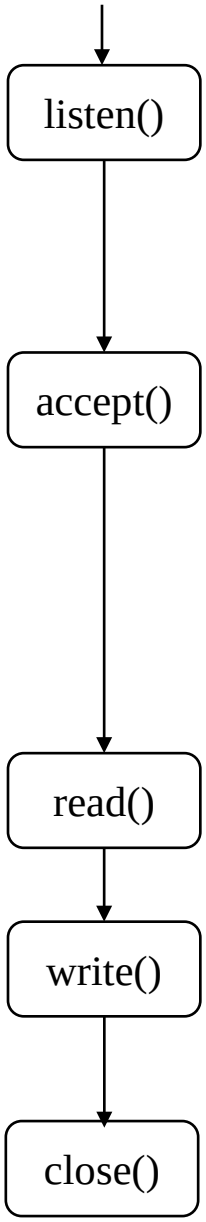


Echo Server Example



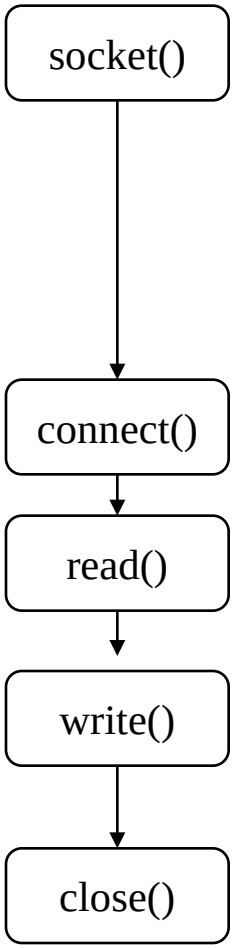

```
#include <sys/socket.h>
#include <sys/stat.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#define MAXBUF 1024

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

   if( 0 > (server_sockfd = socket(AF_INET, SOCK_STREAM, 0))) {
      printf("Error creating socket\n");
   }

   server_addr.sin_family = AF_INET;
   server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
   server_addr.sin_port = htons(3500);

   if(0 > (bind(server_sockfd, (struct sockaddr *)&server_addr,
               sizeof(server_addr))))
   {
      printf("Error binding\n");
   }
```

Echo Server Example (Cont.)




```
if(0 > listen(server_sockfd, 5)) {
   printf("Error lisenting\n");
}
client_addr_len = sizeof(client_addr);

while(1)
{
    if (0 > (client_sockfd = accept(server_sockfd, (struct sockaddr *)
                                    &client_addr, &client_addr_len)))
    {
        printf("Error accepting\n");
    }
    memset(buf, 0x00, MAXBUF);

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

    close(client_sockfd);

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

Echo Client Example




```
struct sockaddr_in server_addr;
int server_sockfd;
int server_addr_len;
char buf[MAXBUF];

if (0 > (server_sockfd=socket(AF_INET, SOCK_STREAM, 0))) {
   printf("Error creating socket\n");
}

server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
server_addr.sin_port = htons(3500);
server_addr_len = sizeof(server_addr);

connect(server_sockfd, (struct sockaddr_in *) &server_addr,
        server_addr_len);
memset(buf, 0x00, MAXBUF);

fgets(buf, MAXBUF, stdin);
if(write(server_sockfd, buf, MAXBUF) <= 0) {
  return 0;
}
if(read(server_sockfd, buf, MAXBUF) <= 0) {
  return 0;
}
fputs(buf, stdout);

close(server_sockfd);
return 0;
```

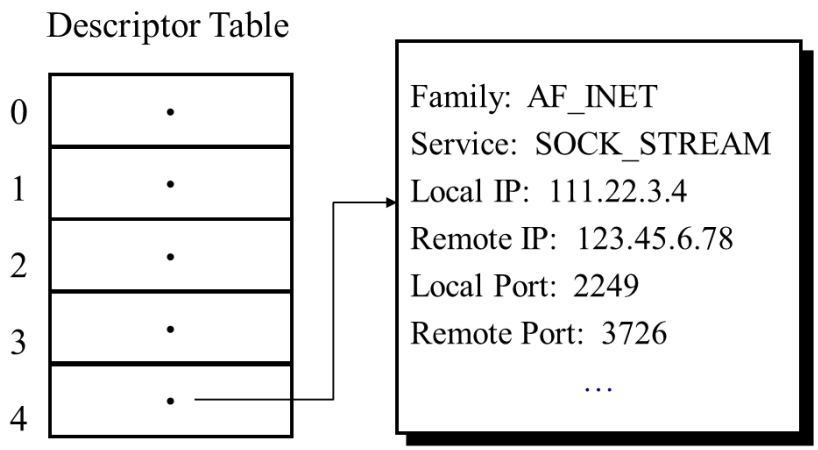
Server-side Socket Programming

❖ socket( )

Domain Type Protocol

```
int server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

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



- Domain
  - AF_INET : IPv4
  - AF_INET6 : IPv6
  - AF_UNIX : on one system
- Type
  - SOCK_STREAM : TCP
  - SOCK_DGRAM : UDP
- Protocol
  - IPPROTO_IP (0) : default
  - IPPROTO_TCP (6)
  - IPPROTO_UDP (17)

Server-side Socket Programming

❖ sockaddr_in

```
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(8080);
```

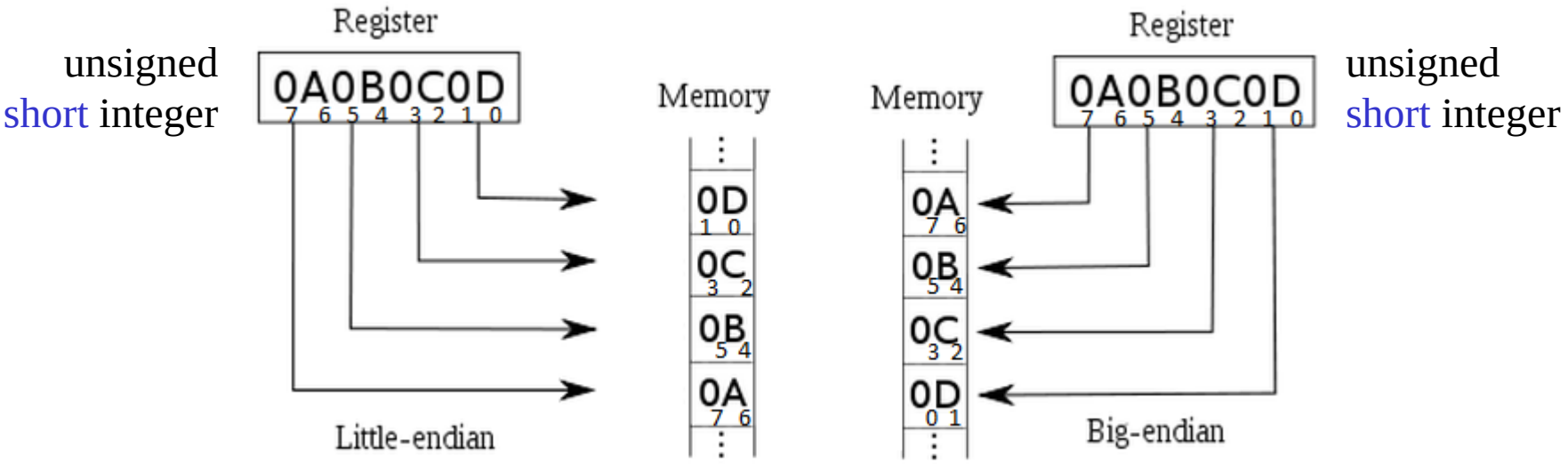
- sockaddr_in structure used to store addresses with IPv4 address family
- member variable
  - sin_family
  - sin_addr.s_addr : 32 bits IP address
    - inet_addr( ) : converts a string containing an IPv4 dotted-decimal address into a proper addres ↔ inet_ntoa( )
  - sin_port : 16 bits port number
 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

Server-side Socket Programming

❖ Network Byte Order

Host(CPU) byte order —— htons ——→ Network byte order

Host(CPU) byte order ←—— ntohs —— Network byte order



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(sockfd, 5);
```

- Backlog : The number of connections to queue

Server-side Socket Programming

❖ accept ( )

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

- accept( ) blocks the process until a connection request arrives
- It creates 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(client_sockfd, buf, MAXBUF) <= 0) {
   close(client_sockfd);
}
...
if(write(client_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 server_sockfd;
struct sockaddr_in server_addr;
int server_addr_len;
...
int server_addr_len = sizeof(client_addr);
connect(server_sockfd, (struct sockaddr_in *) &server_addr,
        server_addr_len);
```

- Client needs only to create socket and connection
- Return value :
  - -1 : error
  - 0 : successfully connceted

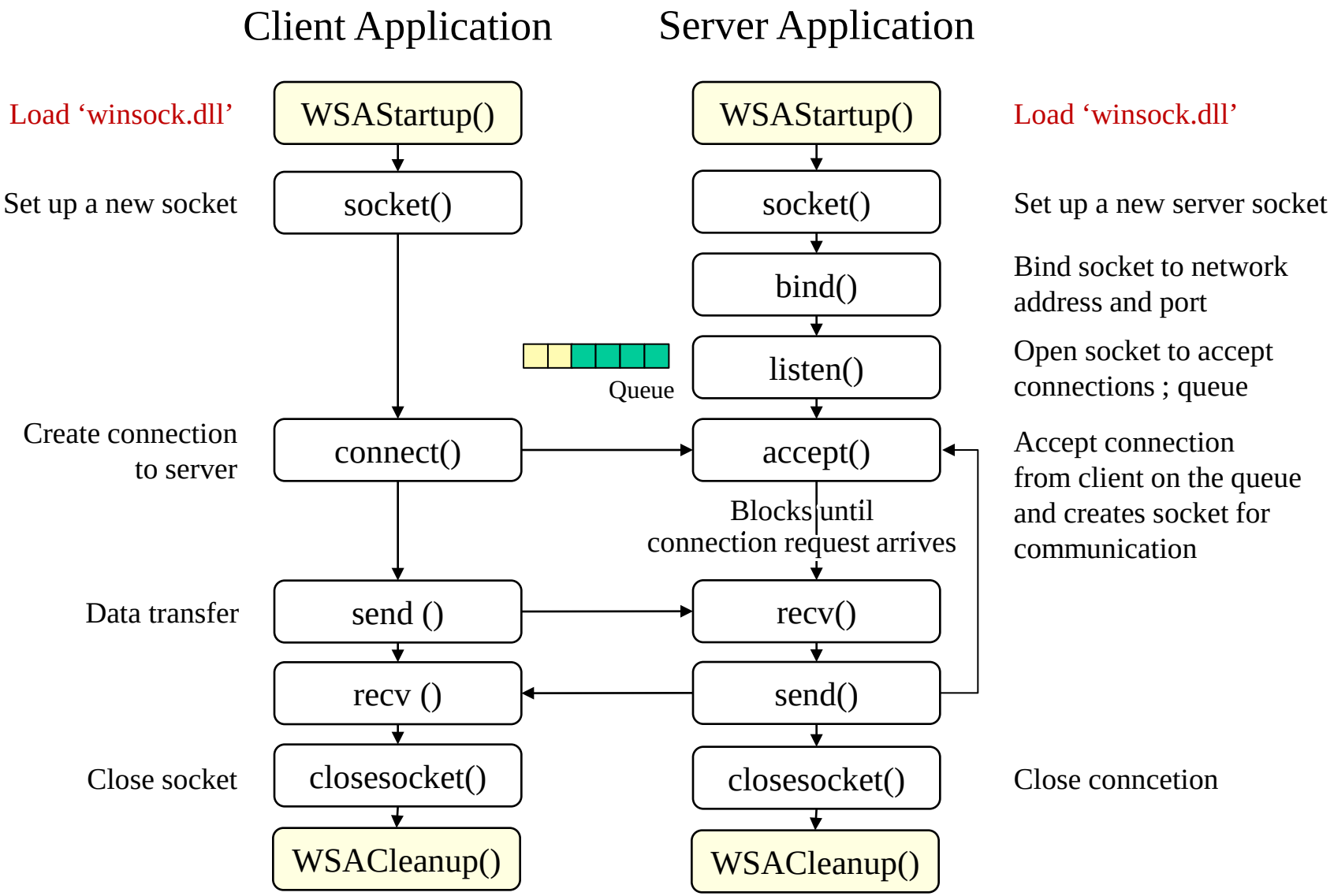
Linux C Based Socket API

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, ICMP)
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 **int 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_in remoteaddress** : Remote socket address where the connection was initiated **int 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_in remoteaddress** : Remote socket address where the connection is to be opened **int 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 **char* data** : Data to be sent, or buffer into which the read data will be placed **int datalength** : Length of the data to be written, or amount of data to be read
close ()	Close a socket	**int sockfd** : Socket which is to be closed

Socket API of Other Programming Languages

Windows Sockets API (C, C++)
Java
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 wVersionRequred,** **LPWSADATA ipWSAD ata);**
socket( )	socket(**PF_INET,** SOCK_STREAM, 0);
close( )	closesocket(socket);
	WSACleanup();
read( )	recv(SOCKET socket, char* buf, int len, int flags);
write( )	send(SOCKET socket, char* buf, int len, int flags);

Socket Programming using Windows Sockets API

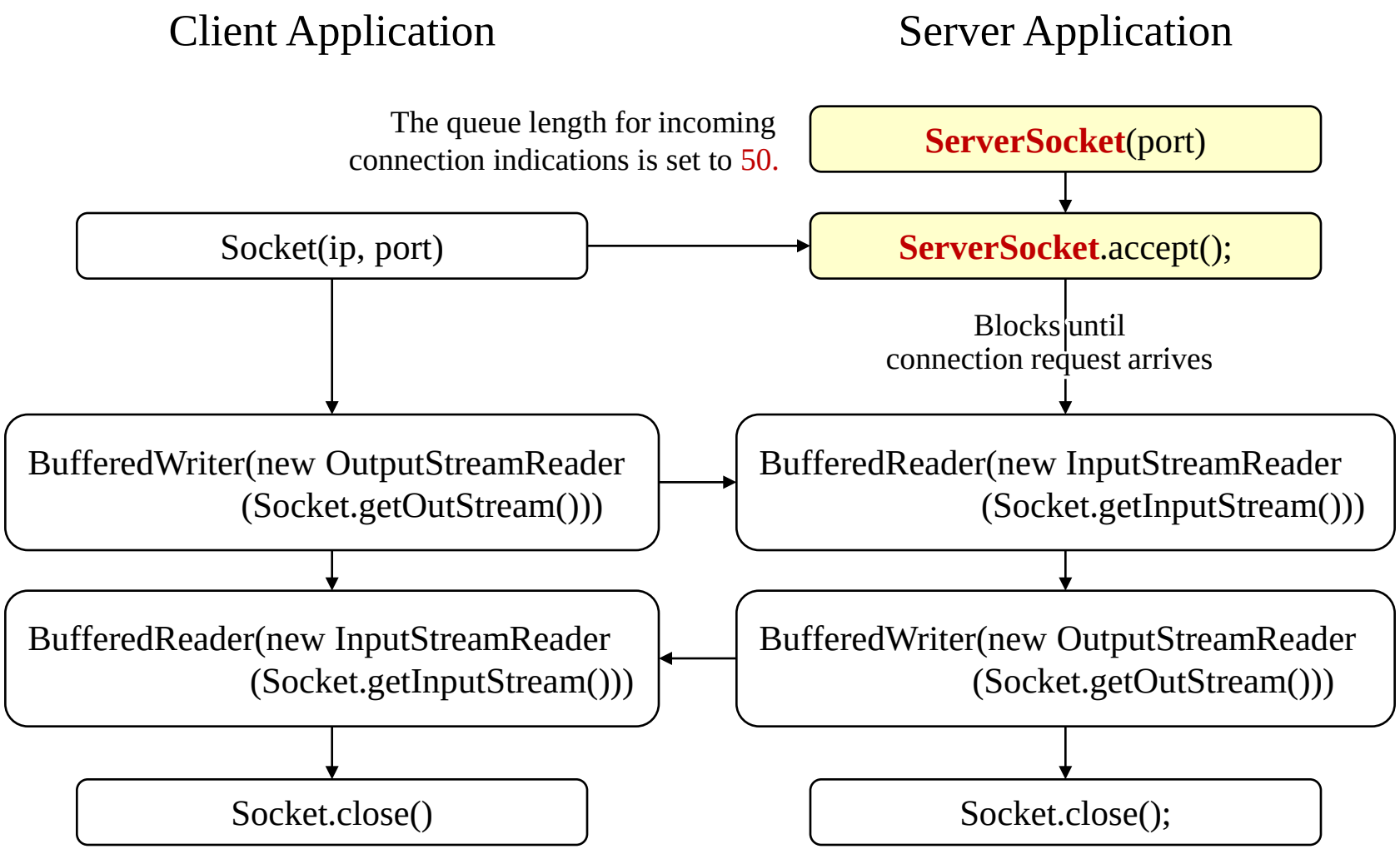
```
SOCKET server_socket = socket(PF_INET, SOCK_STREAM, 0);
```

- Return value : **Socket Handle**
  - Simliar 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 receive details of the WinSocket 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 ( ) : must called for every successful time the WSAStartup() is called.

Socket Programming in Java



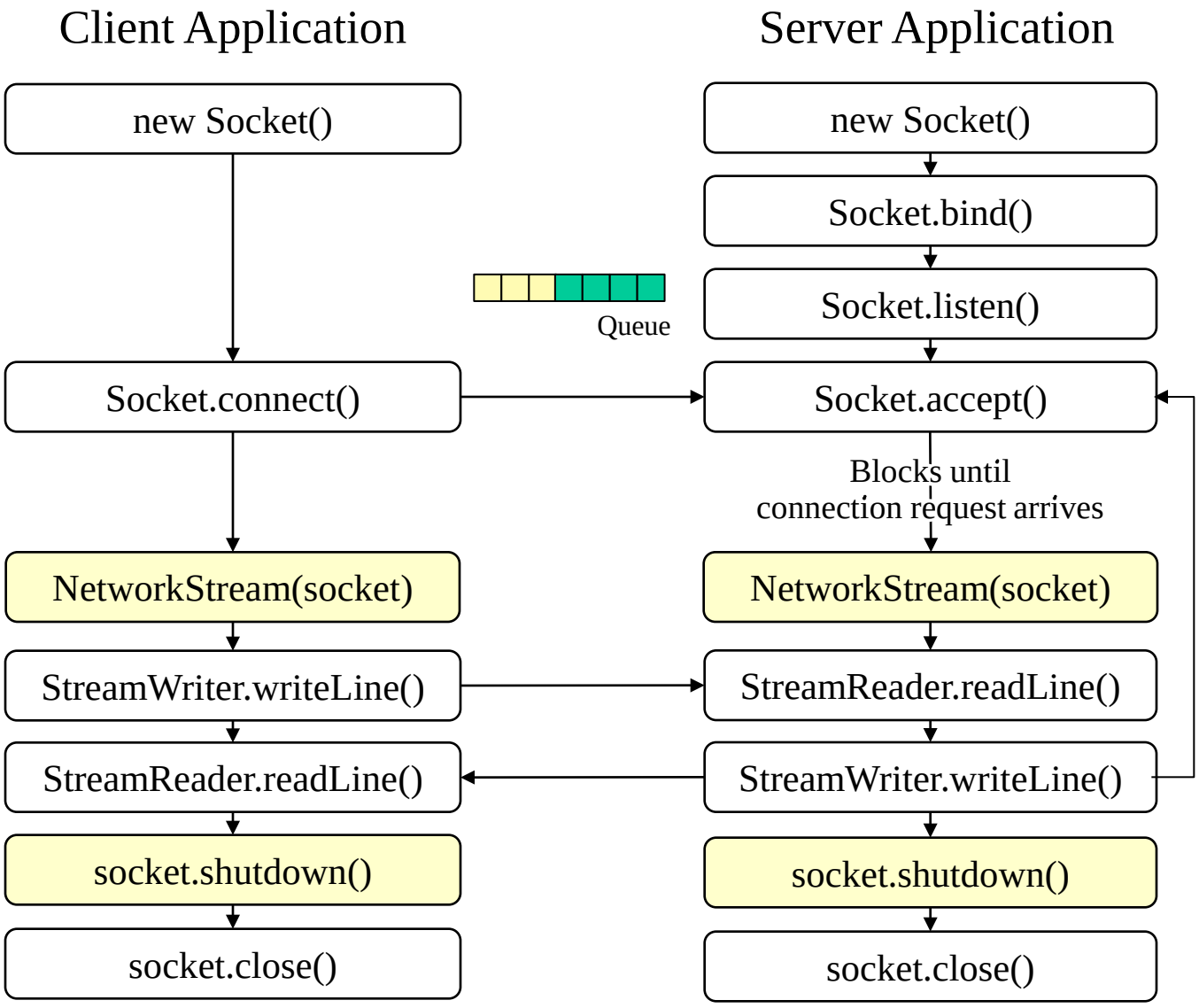
Socket Programming in Java

Linux C		Java
#include <sys/socket.h>		import java.net.*;
Server	sockaddr_in, int socket_fd, socket( ), bind( ), accept( ), listen( )	**ServerSocket s_sock** = new ServerSocket(int **port**);
Client	sockaddr_in, int socket_fd socket( ), connect( )	**Socket** sock = new Socket(String ip, int port);
accept( )		Socket **c_sock** = **s_sock**.accept();
read( )		**BufferedReader** buffer_read = new BufferedReader(new InputStreamReader(**sock**.getInputStream())); String message = buffer_read.readLine();

write( )		**BufferedWriter** buffer_write = new BufferedWriter(
		new OutputStreamReader(**sock**.getOutputStream()));
		buffer_write.writeLine(message);

close()		sock.close();

Socket Programming in C#



Socket Programming in C#

Linux C	C#
#include <sys/socket.h>	using System.Net.Sockets; using System.Net;
sockaddr_in	**IPEndPoint ipep = new IPEndPoint(IPAddress.Any, int port);**
int socket_fd	Socket **socket**;
socket( )	**socket** = new Socket(**AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp**);
bind( )	**socket**.Bind(**ipep**);
listen( )	**socket**.Listen(10);
accept( )	Socket **client_socket = socket.**Accept( );
close( )	**socket.Shutdown(SocketShutdown.Both);** **socket.**Close( );
connect( )	**socket.Connect(ipep);**
read/write( )	**NetworkStream ns = new NetwrokStream(socket);** StreamReader(**ns**).readLine(); StreamWriter(**ns**).writeLine(String buf);

Socket Programming in C#

```
IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 9050);
```

- Represents a network endpoint as an IP address and a port number.
- IPAddress.Any : server listen for client activity on all network interfaces.
  - IPAddress.Any returns “0.0.0.0” (Broadcast Addresss)

    cf. ipaddress.parse(0.0.0.0)

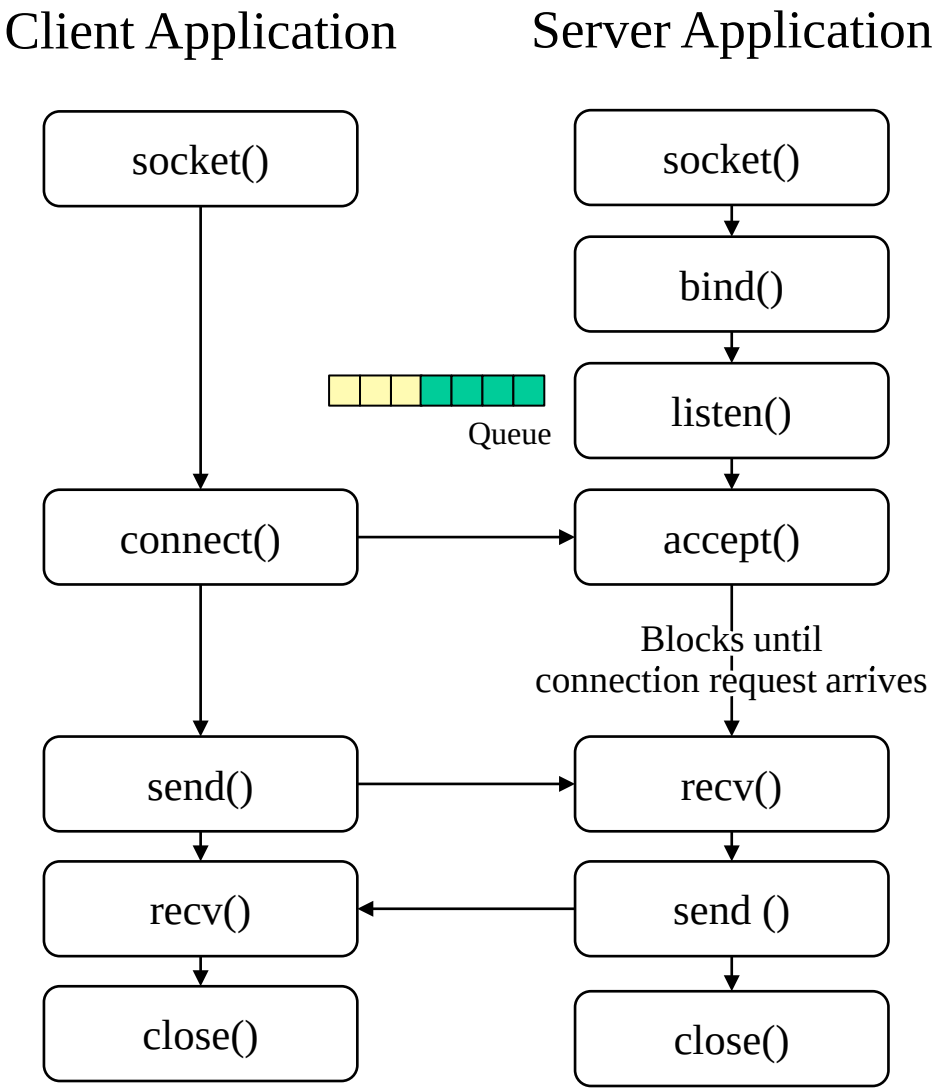
```
NetworkStream ns = new NetwrokStream(socket);
```

- It provides methods for sending and receiving data over Stream sockets in blocking mode.

```
socket.Shutdown(SocketShutdown.Both);
```

- Always call the method before closing the connection-oriented socket.
- This ensures that all data is sent and received on the connected socket before it is closed.

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”, 8080))
listen( )	**s_sock**.listen(5)
accept( )	**c_sock** = s_sock.accept()
close( )	**c_sock**.close()
connect( )	**c_sock**.connect((“localhost”, 8080))
read/write( )	data = c_sock.recv(1028) c_sock.send(data)

References

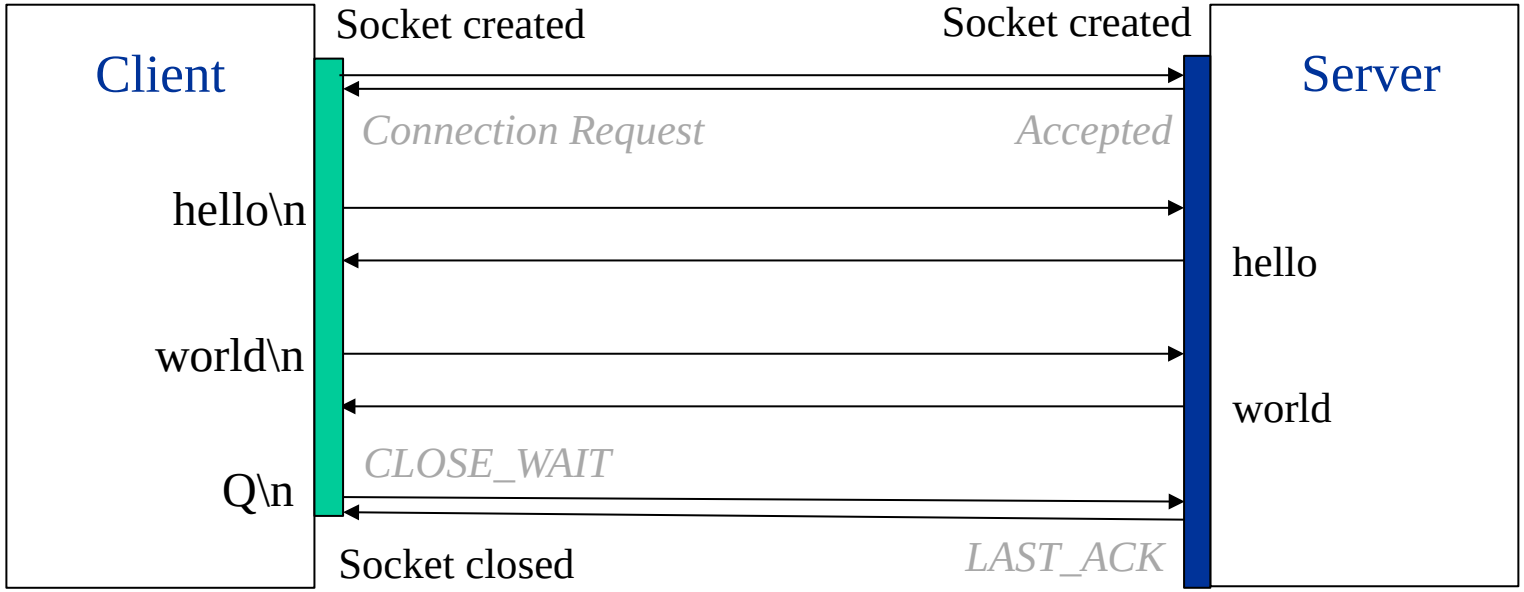
- Linux C Socket API
  - http://manpages.ubuntu.com/manpages/hardy/man7/socket.7.html
- WinSock
  - https://msdn.microsoft.com/en-us/library/aa925696.aspx
  - https://msdn.microsoft.com/en-us/library/windows/desktop/ms741394(v=vs.85).aspx
- Java Socket API
  - http://docs.oracle.com/javase/7/docs/api/java/net/Socket.html
  - http://download.java.net/jdk7/archive/b123/docs/api/java/net/ServerSocket.html
- C# Socket API
  - https://msdn.microsoft.com/en-us/library/system.net.sockets.socket(v=vs.110).aspx
  - https://msdn.microsoft.com/en-us/library/system.net.sockets.networkstream(v=vs.110).aspx
- Python Socket API
  - https://docs.python.org/2/library/socket.html
  - https://docs.python.org/3/library/socket.html

Practical Homework

Homework 1:

A stream-based echo server

- A client sends messages to the server and receives back each message in turn.
  - The client process sends multiple message until it terminates the connection
  - The connection of the client process is terminated when the user press ‘Q\n’
  - The server echoes the message orderly and immediately



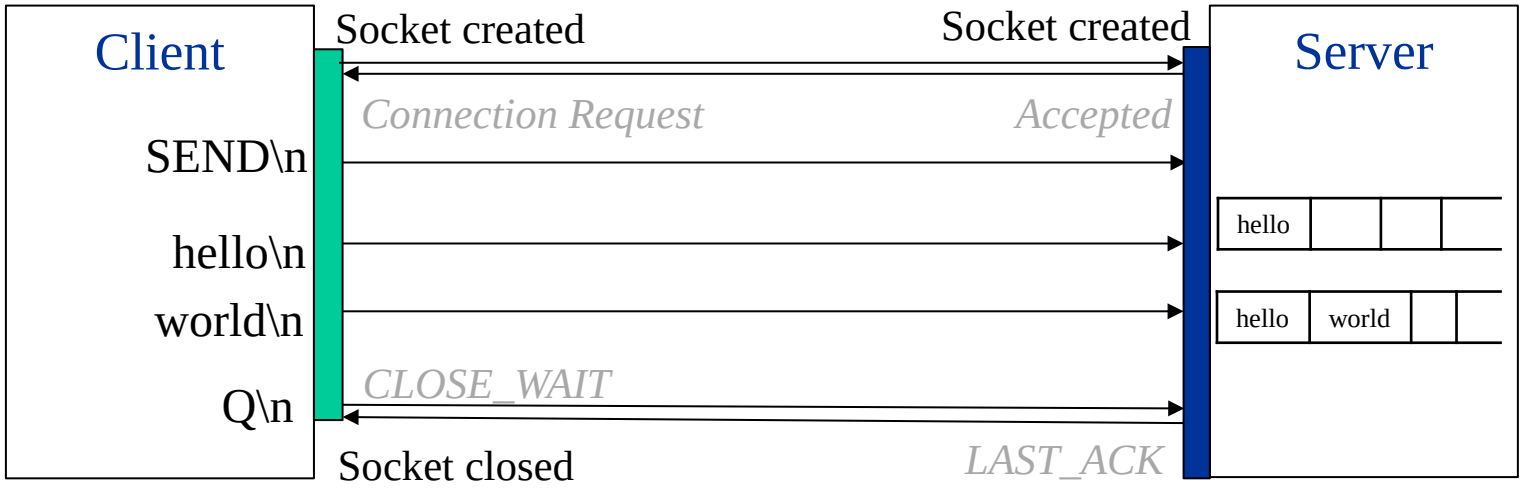
Homework 2:

A stream-based messaging service

- After connection, the client process first sends ‘SEND\n’ or ‘RECV\n’ message to the server
- If the client selects sending, the client process sends multiple message until it terminates the connection
  - All receiving messages are queued into the server
  - The connection of the client process is terminated when the user press ‘Q\n’
- If the client selects receiving, the server process sends all received messages during sending orderly
  - The server sends “LAST_MSG” message to the client after sending all messages in the queue
  - The connection of the client process is terminated after receiving “LAST_MSG” message
  - Queue at the server becomes empty after transmission
- Otherwise, the connection of the client process is terminated

Homework 2

- Sending example



- Receiving example