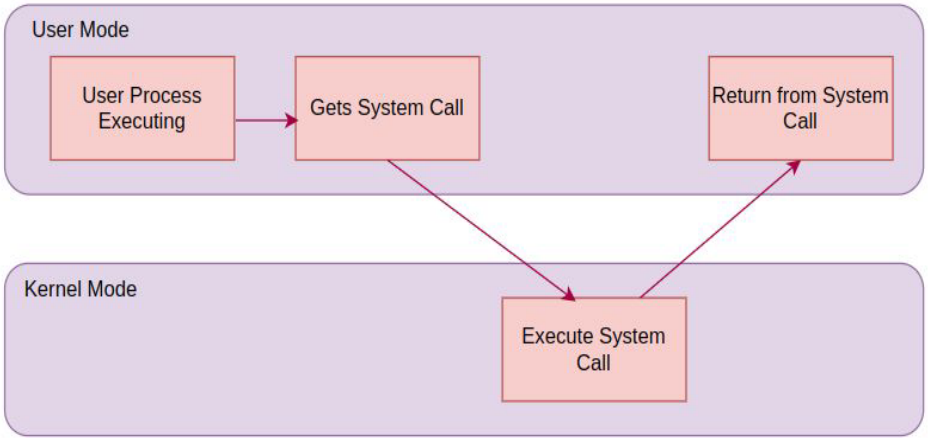
Socket Programming using blocking system calls (Server Client Architecture)

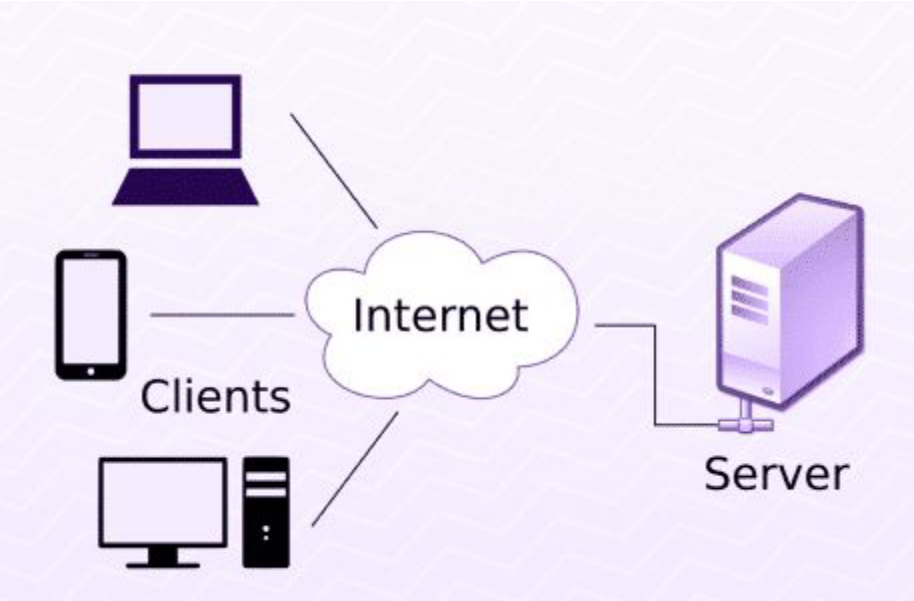
Design Lab (CS69202)
Feb. 5, 2025

Blocking System Calls



Blocking system calls halt the execution of a program until a specific event occurs, such as receiving data or establishing a connection. This simplicity can lead to performance issues when handling many clients.

Introduction to Server Client Architecture

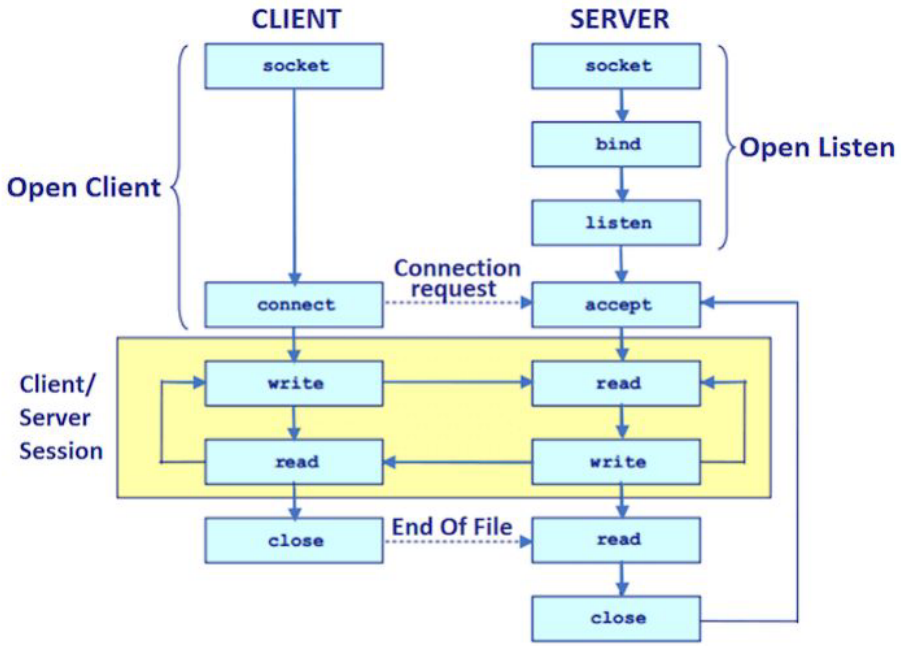


Introduction to Server Client Architecture

Server	Client
A server is a program that listens for requests from clients and responds accordingly. It acts as a centralized point of service.	A client is a program that initiates a request to a server and receives a response. Clients consume services provided by the server.

Network Programming

Network Programming : Client - Server Architecture



TCP (Transmission Control Protocol) Client-Server

- In this section we illustrate the use of sockets for inter-process communication across the network.
- We show the communication between a server process and a client process.

Network Programming : Server side

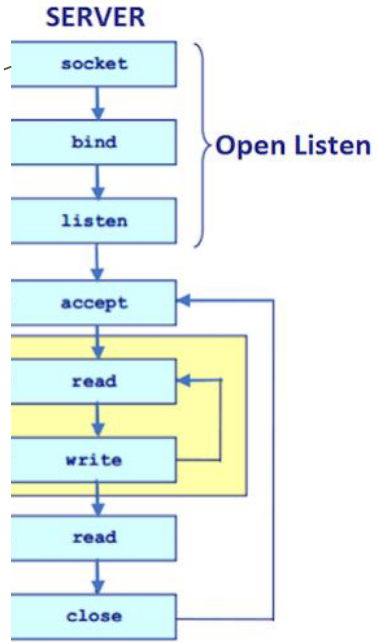
- Since many server processes may be running in a system, we identify the desired server process by a "port number". In the code snippets, we choose port number 6000 for our server process.

- To create a TCP server process, we first need to open a "socket" using the socket( ) system call.

- The following three files must be included for network programming -> <sys/socket.h>, <netinet/in.h> and <arpa/inet.h>

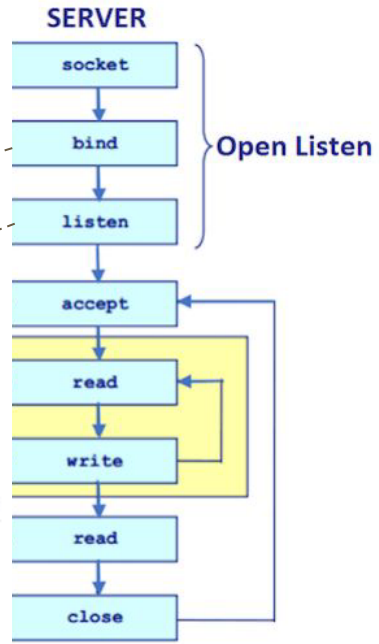
Network Programming : Server side

```c
main( )
{
    int sockfd, newsockfd; /* Socket descriptors */
    int clilen;
    struct sockaddr_in cli_addr, serv_addr;
    int i;
    char buf[100]; /* A buffer for communication */

    /* The following system call opens a socket. The first parameter
    indicates the family of the protocol to be followed. For internet
    protocols we use AF_INET. For TCP sockets the second
    parameter is SOCK_STREAM. The third parameter is set to 0
    for user applications. */

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0) ) < 0) {
        printf("Cannot create socket\n");
        exit(0);
    }
```



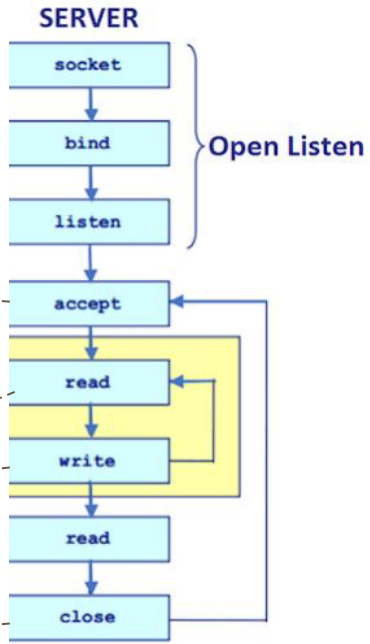
Network Programming : Server side

```
/* The structure "sockaddr_in" is defined in <netinet/in.h> for the
internet family of protocols. This has three main fields. The
field "sin_family" specifies the family and is therefore AF_INET
for the internet family. The field "sin_addr" specifies the internet
address of the server. This field is set to INADDR_ANY for machines
having a single IP address. The field "sin_port" specifies the port
number of the server. */

serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = 6000;

/* With the information provided in serv_addr, we associate the server
with its port using the bind() system call. */

if ( bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
    printf("Unable to bind local address\n");
    exit(0);
}

listen(sockfd, 2); /* This specifies that up to 2 concurrent client
requests will be queued up while the system is
executing the "accept" system call below. */
```



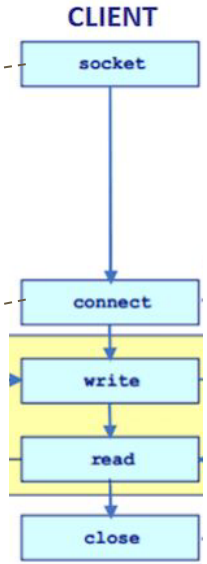
Network Programming : Server side

```
    while (1)
    {

        /* The accept() system call accepts a client connection. It blocks
        the server until a client request comes. The accept() system call
        fills up the client's details in a struct sockaddr which is passed
        as a parameter. The length of the structure is noted in clilen. */

        clilen = sizeof(cli_addr);
        newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen) ;
        if (newsockfd < 0) {
            printf("Accept error\n");
            exit(0);
        }

        /* Having successfully accepted a client connection, the server now
        sends message and loops back to accept the next connection. */

        for(i=0; i < 100; i++) buf[i] = '\0'; /* Initialize buffer */

        strcpy(buf,"Message from server"); /* Copy message */

        send(newsockfd, buf, 100, 0); /* Send message */

        for(i=0; i < 100; i++) buf[i] = '\0'; /* Initialize buffer */

        recv(newsockfd, buf, 100, 0); /* Receive message */

        printf("%s\n", buf);
        close(newsockfd);
    } /* End of while loop */
} /* End of main( ) */
```



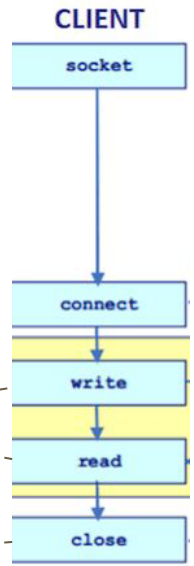
Network Programming : Client side

```
main( )
{
    int sockfd ;
    struct sockaddr_in serv_addr;
    int i;
    char buf[100];

    /* Opening a socket is exactly similar to the server process */

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("Unable to create socket\n");
        exit(0);
    }
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    serv_addr.sin_port = 6000;

    /* With the information specified in serv_addr, the connect( )
    system call establishes a connection with the server process. */

    if (connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
        printf("Unable to connect to server\n");
        exit(0);
    }
```

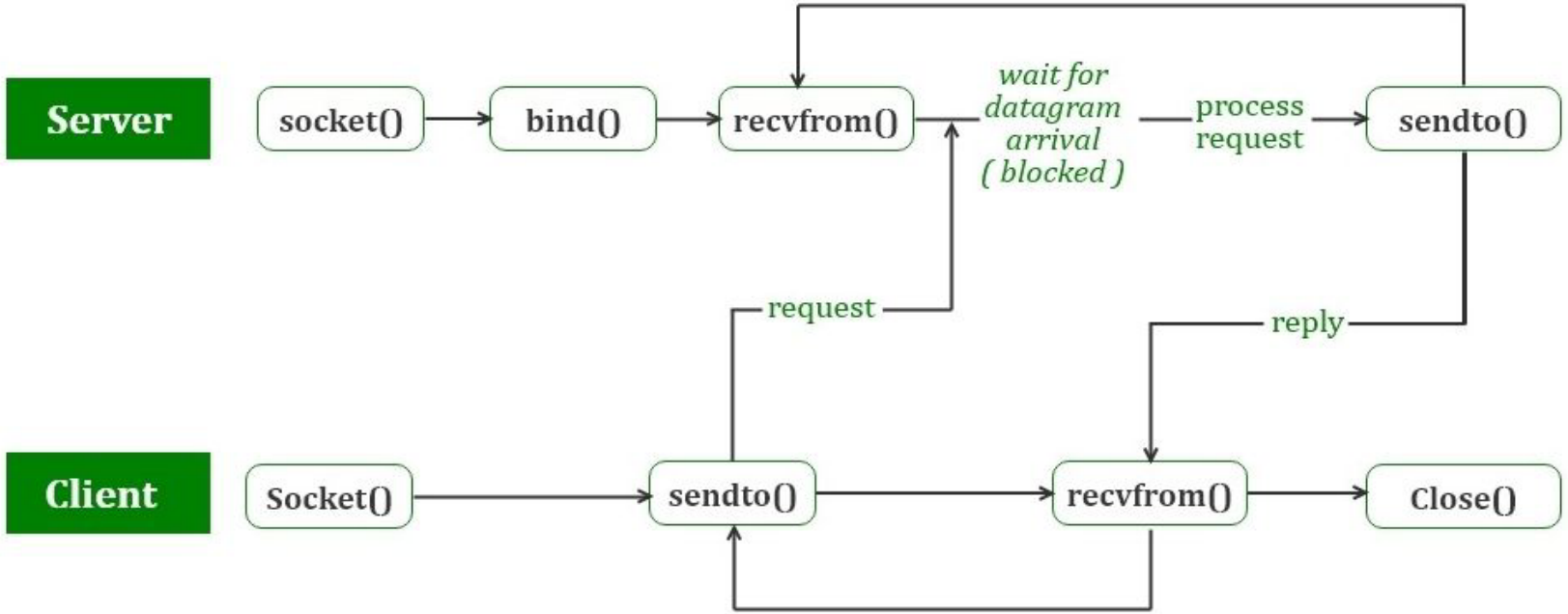


Network Programming : Client side

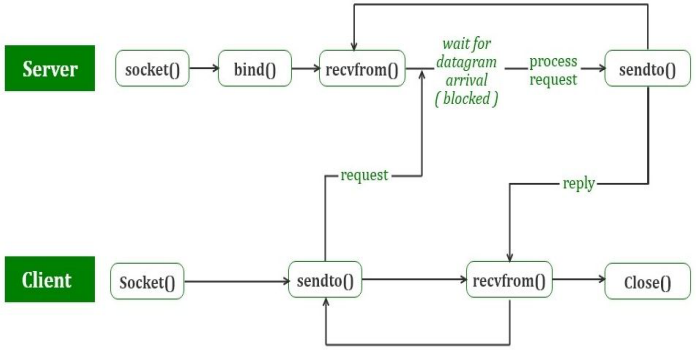
```
    /* After connection, the client can send or receive messages. However,
    please note that recv( ) will block when the server is not sending and
    vice versa. Similarly send( ) will block when the server is not receiving
    and vice versa. For non-blocking modes, refer to the online man pages.*/

    for(i=0; i < 100; i++) buf[i] = '\0';

    recv(sockfd, buf, 100, 0);

    printf("%s\n", buf);

    for(i=0; i < 100; i++) buf[i] = '\0';

    strcpy(buf,"Message from client");

    send(sockfd, buf, 100, 0);

    close(sockfd);
}
```



UDP (User Datagram Protocol) Sockets



UDP (User Datagram Protocol) Sockets



TCP is a **connection-oriented** protocol, where as UDP is a **connection-less** protocol.

- Does not have any connect( ) from client and any accept( ) from server (because no connection is established)

- Instead each message sent using sendto( ) must specify the destination IP address.

References

- Beej's Guide to Network Programming (http://www.cs.columbia.edu/~danr/courses/6761/Fall00/hw/pa1/6761-sockhelp.pdf)
- Tutorial - https://nikhilroxtomar.medium.com/tcp-client-server-implementation-in-c-idiot-developer-52509a6c1f59