**SEMAPHORE**

**Semctl():** control operations on a semaphore

**Syntax:**

int semctl(int semid, semnum, int cmd, union semun arg);

**semop():** perform semaphore operations(lock/unlock)(-1 for locking / +1 for unlocking)

**Syntax:**

Int semop(int semid, struct sembuf\*sops, size\_t nsops);

**Parameters:**

Semid: semaphore id returned by semget()

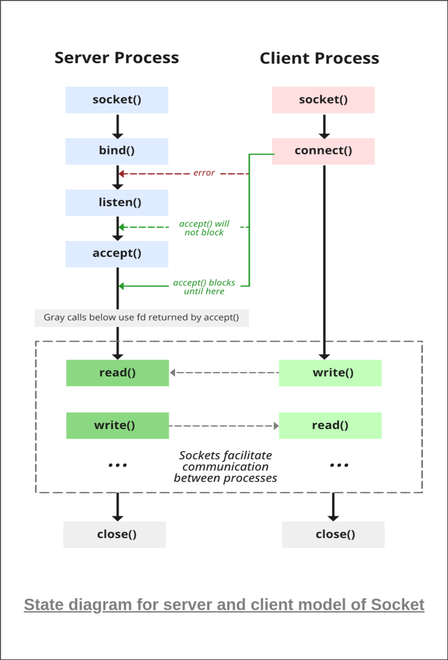
Sops: pointer to an array of struct sembuf operations to perform

nsops: no.of operations perform in an array

**SOCKET PROGRAMMING**

* Socket programming is a way of connecting two nodes on a network to communicate with each other.
* One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection.

Example: Listener will treated as server and the sender is treated as client.



**Stages for Server**

The server is created using the following steps:

**1. Socket Creation**

int sockfd = socket(domain, type, protocol)

* **sockfd:** socket descriptor, an integer (like a file handle)
* **domain:** integer, specifies communication domain. We use AF\_ LOCAL as defined in the POSIX standard for communication between processes on the same host. For communicating between processes on different hosts connected by IPV4, we use AF\_INET (AF🡪 address family) and AF\_I NET 6 for processes connected by IPV6.
* **type:** communication type  
  SOCK\_STREAM: TCP(reliable, connection-oriented)  
  SOCK\_DGRAM( datagram) : UDP(unreliable, connectionless)
* **protocol:**Protocol value for Internet Protocol(IP), which is 0. This is the same number that appears on the protocol field in the IP header of a packet.(man protocols for more details)
* **3. Bind**
* int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);
* After the creation of the socket, the bind function binds the socket to the address and port number specified in addr(custom data structure). In the example code, we bind the server to the localhost, hence we use INADDR\_ANY to specify the IP address.

A screen shot of a computer

Description automatically generated

**4. Listen**

int listen(int sockfd, int backlog);

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

* The backlog argument defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives
* when the queue is full, the client may receive an error with an indication of ECONNREFUSED or, if the underlying protocol supports retransmission,
* the request may be ignored so that a later reattempt at connection succeeds.

**\*\*\*Backlog argument is defined as the pending connections wgich are in queue**

**For example , in backlog arg if we take 5 as arg and it has 1000 clients limit and what will happen here is after establishing 1010 servers with those 1000 clients, those 5 backlog args will be in pending queue after completing 1000 which is full and remaining 1006 to 1010 are getting refused.**

**Now if one is getting outside from the queue which is full then one backlog arg will be entered to make connection.**

**5. Accept( client’s address, ports, address length)**

int new\_socket= accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen);

It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. At this point, the connection is established between client and server, and they are ready to transfer data.

🡪If the server is closed, then client should also be closed.