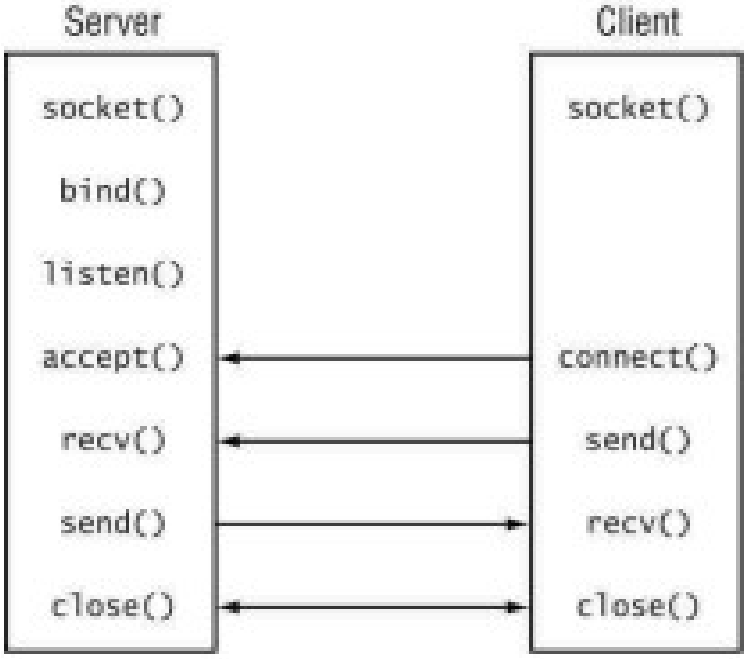
SOCKET PROGRAMMING

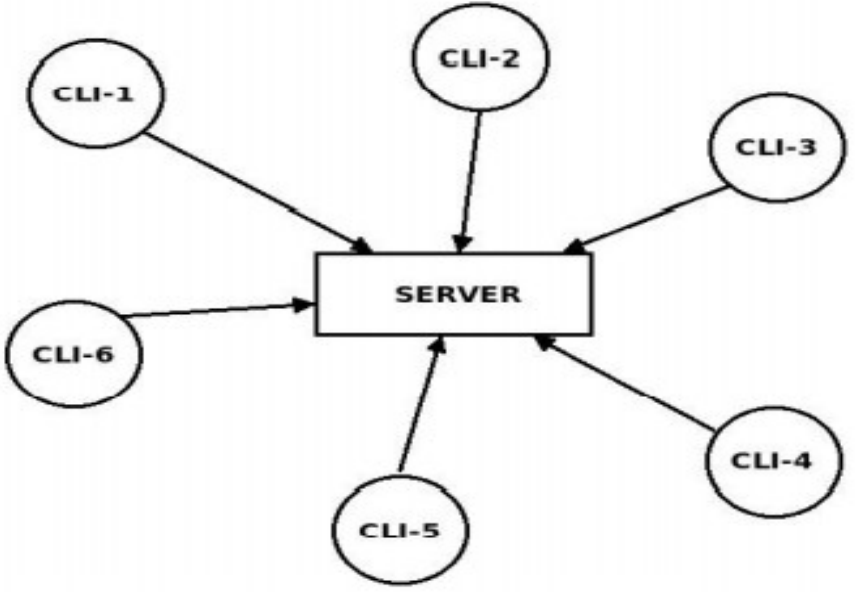
CONCURRENT SERVERS

Socket Programming Framework/API

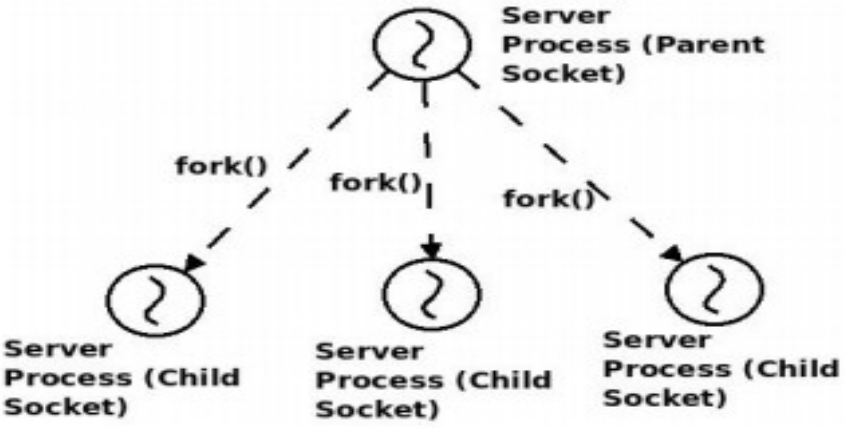
A set of **system calls** to get the service from TCP/IP protocol stack (`net` module in the OS kernel).



Concurrent Servers



Extending the Server Socket for Multiple Connections



Iterative Server

```
/*
 * listen: make this socket ready to accept connection requests
 */
if (listen(parentfd, 5) < 0) /* allow 5 requests to queue up */
  error("ERROR on listen");

/*
 * main loop: wait for a connection request, echo input line,
 * then close connection.
 */
clientlen = sizeof(clientaddr);
while (1) {

  /*
   * accept: wait for a connection request
   */
  childfd = accept(parentfd, (struct sockaddr *) &clientaddr, &clientlen);
  if (childfd < 0)
    error("ERROR on accept");
```

How Iterative Server Works

- The `listen()` call sets a flag that the socket is in listening state and set the maximum number of backlog connections.
- The `accept()` call blocks a listening socket until a new connection comes in the connection queue and it is accepted.
- Once the new connection is accepted, a new socket file descriptor (say `connfd`) is returned, which is used to read and write data to the connected socket.
- All other connections, which come in this duration, are backlogged in the connection queue.
- Once the handling of the current connected socket is done, the next `accept()` call accepts the next incoming connection from the connection queue (if any), or blocks the listening socket until the next connection comes.

Parallel processing of each incoming sockets

```
pid_t pid;
int listenfd, connfd;
listenfd = Socket( ... );
/* fill in sockaddr_in{} with server's well-known port */
Bind(listenfd, ... );
Listen(listenfd, LISTENQ);
for ( ; ; ) {
connfd = Accept (listenfd, ... ); /* probably blocks */
if( (pid = Fork()) == 0) {
Close(listenfd); /* child closes listening socket */
doit(connfd); /* process the request */
Close(connfd); /* done with this client */
exit(0); /* child terminates */
}
Close(connfd); /* parent closes connected socket */}
```

THANK YOU