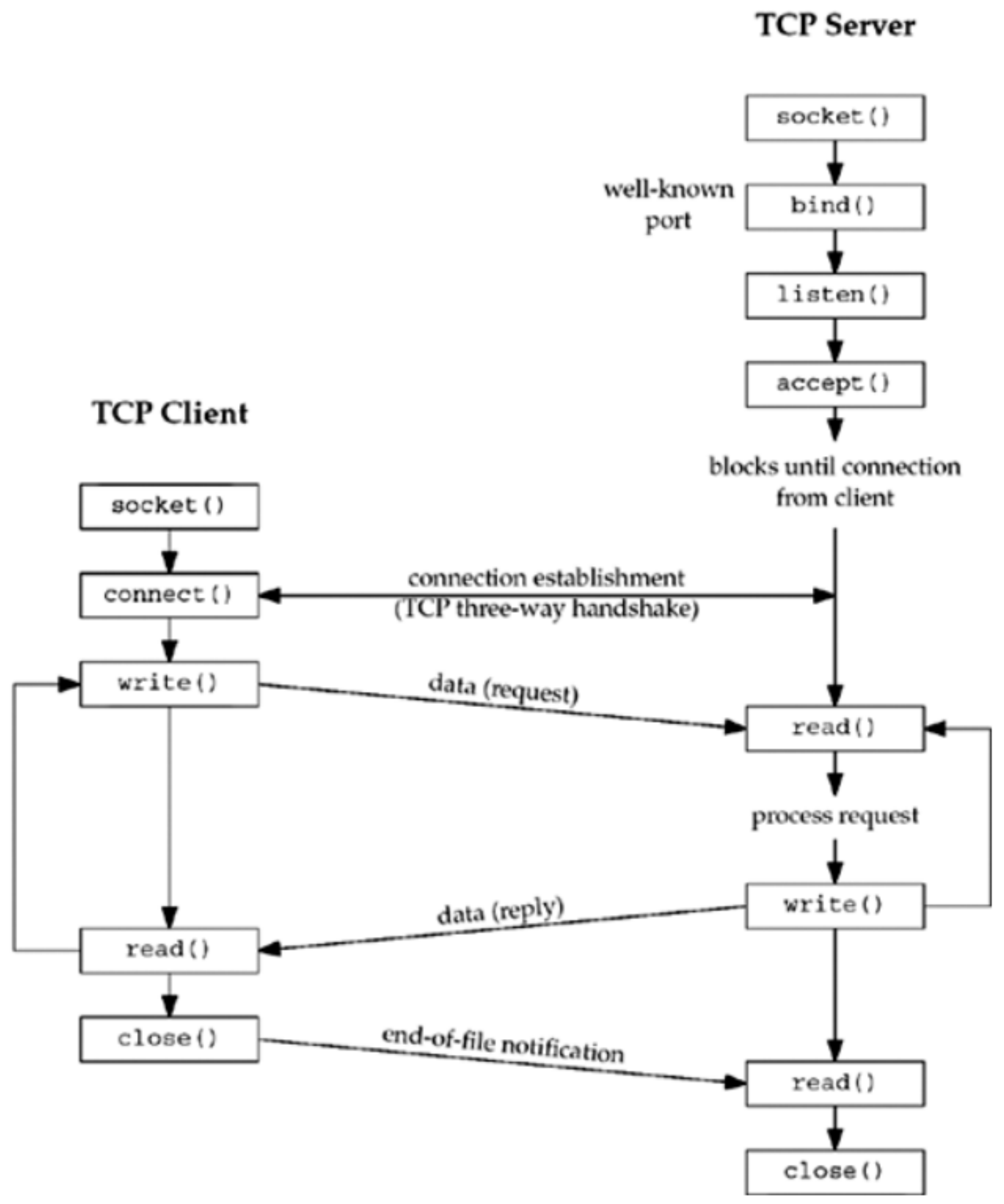
TCP sockets

Giảng viên: TS. Trần Hải Anh
Bộ môn Truyền Thông & Mạng máy tính
Khoa CNTT & TT

Introduction

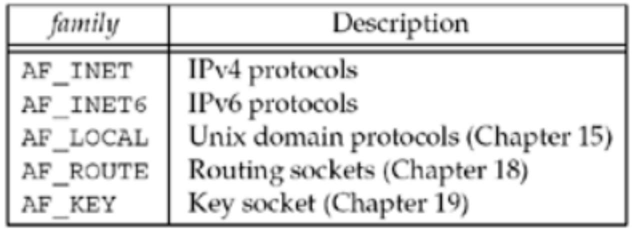


socket function

- To perform network I/O, the first thing a process must do is call the *socket* function

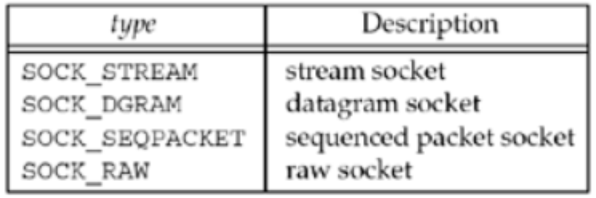
```
#include <sys/socket.h>
int socket (int family, int type, int protocol);
```

- Returns: non-negative descriptor if OK, -1 on error



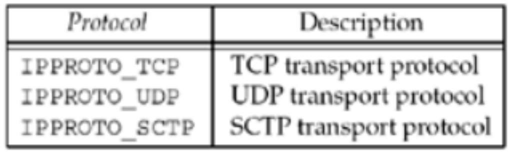
family	Description
AF_INET	IPv4 protocols
AF_INET6	IPv6 protocols
AF_LOCAL	Unix domain protocols (Chapter 15)
AF_ROUTE	Routing sockets (Chapter 18)
AF_KEY	Key socket (Chapter 19)

family



type	Description
SOCK_STREAM	stream socket
SOCK_DGRAM	datagram socket
SOCK_SEQPACKET	sequenced packet socket
SOCK_RAW	raw socket

socket



Protocol	Description
IPPROTO_TCP	TCP transport protocol
IPPROTO_UDP	UDP transport protocol
IPPROTO_SCTP	SCTP transport protocol

protocol

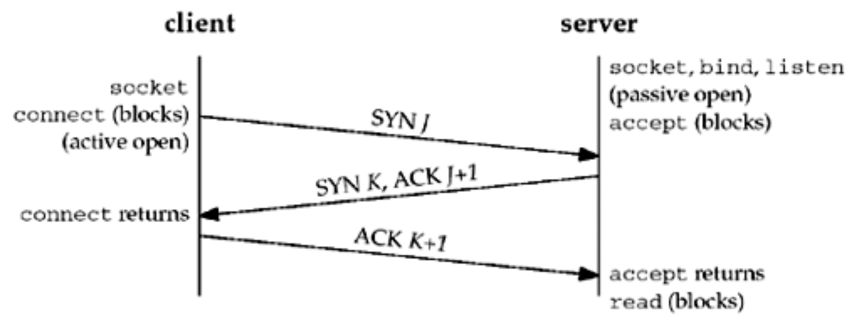
connect Function

- The connect function is used by a TCP client to establish a connection with a TCP server.

```
#include <sys/socket.h>
int connect(int sockfd, const struct sockaddr
  *servaddr, socklen_t addrlen);
```

- Returns: 0 if OK, -1 on error
- *sockfd* is a socket descriptor returned by the *socket* function
- The second and third arguments are a pointer to a socket address structure and its size.
- The client does not have to call *bind* before calling *connect*: the kernel will choose both an ephemeral port and the source IP address if necessary.

Nhắc lại: Thiết lập liên kết TCP : Giao thức bắt tay 3 bước



- Bước 1: A gửi SYN cho B
  - chỉ ra giá trị khởi tạo seq # của A
  - không có dữ liệu
- Bước 2: B nhận SYN, trả lời bằng SYNACK
  - B khởi tạo vùng đệm
  - chỉ ra giá trị khởi tạo seq. # của B
- Bước 3: A nhận SYNACK, trả lời ACK, có thể kèm theo dữ liệu

connect Function (2)

- Problems with *connect* function:
  1. If the client TCP receives no response to its SYN segment, ETIMEDOUT is returned. (If no response is received after a total of 75 seconds, the error is returned).
  2. If the server's response to the client's SYN is a reset (RST), this indicates that no process is waiting for connections on the server host at the port specified (i.e., the server process is probably not running). Error: ECONNREFSED.
  3. If the client's SYN elicits an ICMP "destination unreachable" from some intermediate router, this is considered a soft error. If no response is received after some fixed amount of time (75 seconds for 4.4BSD), the saved ICMP error is returned to the process as either EHOSTUNREACH or ENETUNREACH.

bind Function

- The bind function assigns a local protocol address to a socket.

```
#include <sys/socket.h>
int bind (int sockfd, const struct sockaddr
  *myaddr, socklen_t addrlen);
```

- Returns: 0 if OK,-1 on error
- Example:

```
struct sockaddr_in address;

/* type of socket created in socket() */
  address.sin_family = AF_INET;
  address.sin_addr.s_addr = INADDR_ANY;
/* 7000 is the port to use for connections */
  address.sin_port = htons(7000);
/* bind the socket to the port specified above */
```

listen Function

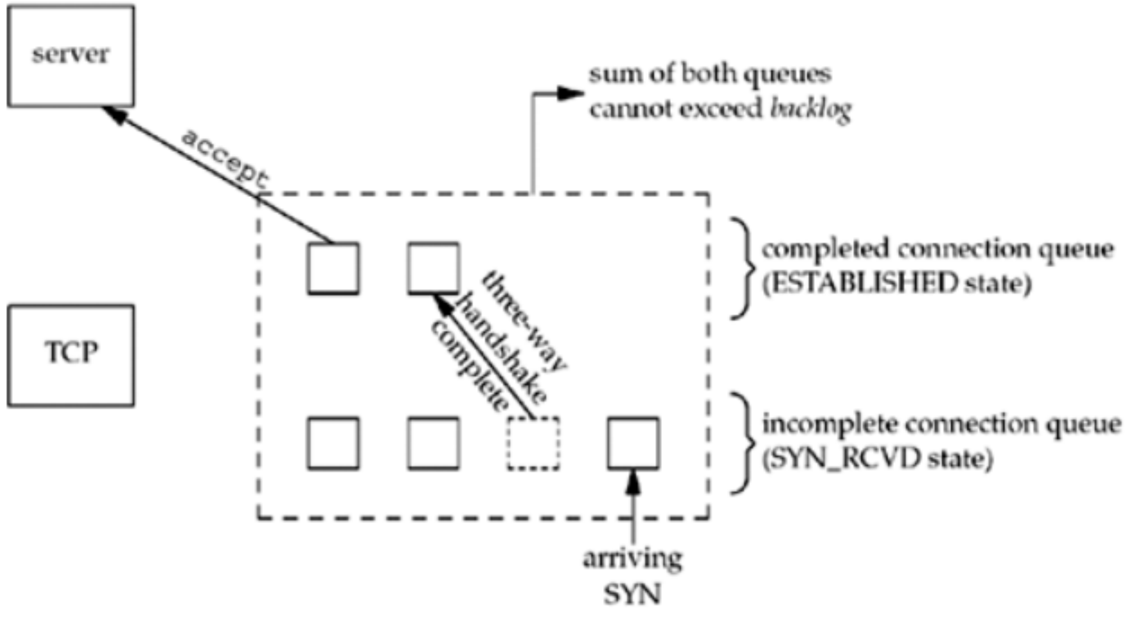
- The listen function is called only by a TCP server.
- When a socket is created by the *socket* function, it is assumed to be an active socket, that is, a client socket that will issue a *connect*.
- The *listen* function converts an unconnected socket into a passive socket, indicating that the kernel should accept incoming connection requests directed to this socket.
- Move the socket from the CLOSED state to the LISTEN state.

```
#include <sys/socket.h>
int listen (int sockfd, int backlog);
```

- Returns: 0 if OK, -1 on error

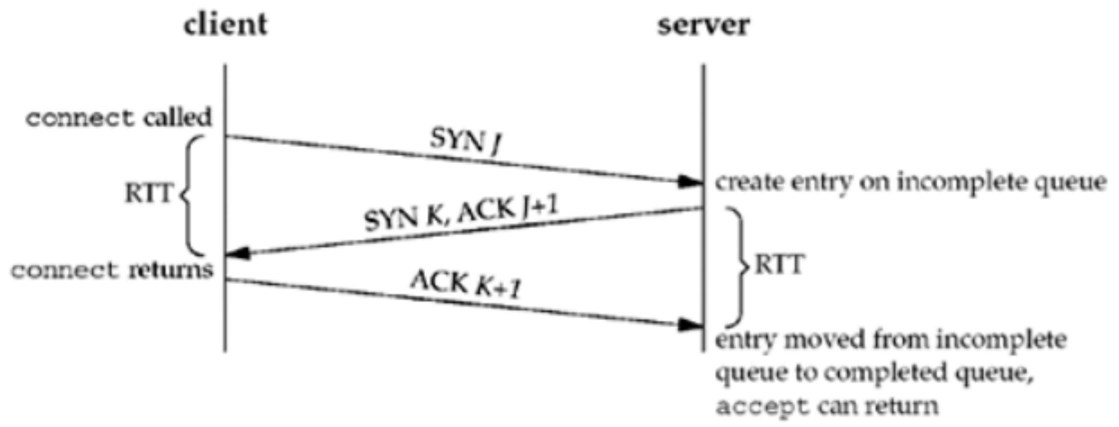
listen Function (2)

- The second argument (*backlog*) to this function specifies the maximum number of connections the kernel should queue for this socket.


server
accept
sum of both queues cannot exceed *backlog*
completed connection queue (ESTABLISHED state)
TCP
three-way handshake complete
incomplete connection queue (SYN_RCVD state)
arriving SYN

The two queues maintained by TCP for a listening socket

listen Function (3)



client
server
connect called
SYN J
create entry on incomplete queue
RTT
SYN K, ACK J+1
connect returns
RTT
ACK K+1
entry moved from incomplete queue to completed queue, accept can return

TCP three-way handshake and the two queues for a listening socket.

accept Function

- *accept* is called by a TCP server to return the next completed connection from the front of the completed connection queue.
- If the completed connection queue is empty, the process is put to sleep.

```
#include <sys/socket.h>
int accept (int sockfd, struct sockaddr *cliaddr,
  socklen_t *addrlen);
```

- Returns: non-negative descriptor if OK, -1 on error
- The *cliaddr* and addrlen arguments are used to return the protocol address of the connected peer process (the client).
- *addrlen* is a value-result argument

accept Function

- Example

```
int addrlen;
struct sockaddr_in address;

  addrlen = sizeof(struct sockaddr_in);
  new_socket = accept(socket_desc, (struct sockaddr *)&address, &addrlen);
  if (new_socket<0)
    perror("Accept connection");
```

fork and *exec* Functions

```
#include <unistd.h>
pid_t fork(void);
```

- Returns: 0 in child, process ID of child in parent, -1 on error
- *fork* function (including the variants of it provided by some systems) is the only way in Unix to create a new process.
- It is called once but it returns twice.
- It returns once in the calling process (called the parent) with a return value that is the process ID of the newly created process (the child). It also returns once in the child, with a return value of 0.
- The reason fork returns 0 in the child, instead of the parent's process ID, is because a child has only one parent and it can always obtain the parent's process ID by calling *getppid*.

Example

```
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    printf("--beginning of program\n");

    int counter = 0;
    pid_t pid = fork();

    if (pid == 0)
    {
        // child process
        int i = 0;
        for (; i < 5; ++i)
        {
            printf("child process: counter=%d\n", ++counter);
        }
    }
    else if (pid > 0)
    {
        // parent process
        int j = 0;
        for (; j < 5; ++j)
        {
            printf("parent process: counter=%d\n", ++counter);
        }
    }
    else
    {
        // fork failed
        printf("fork() failed!\n");
        return 1;
    }

    printf("--end of program--\n");

    return 0;
}
```

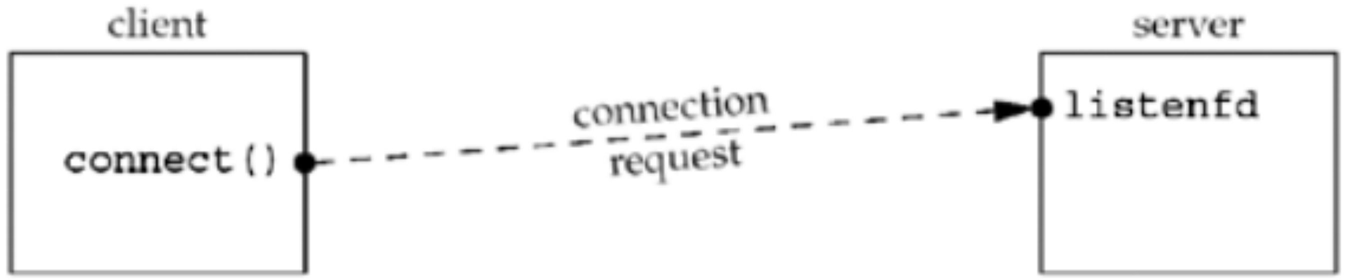
- 2 typical uses of fork:
  - A process makes a copy of itself so that one copy can handle one operation while the other copy does another task.
  - A process wants to execute another program.

Concurrent Servers

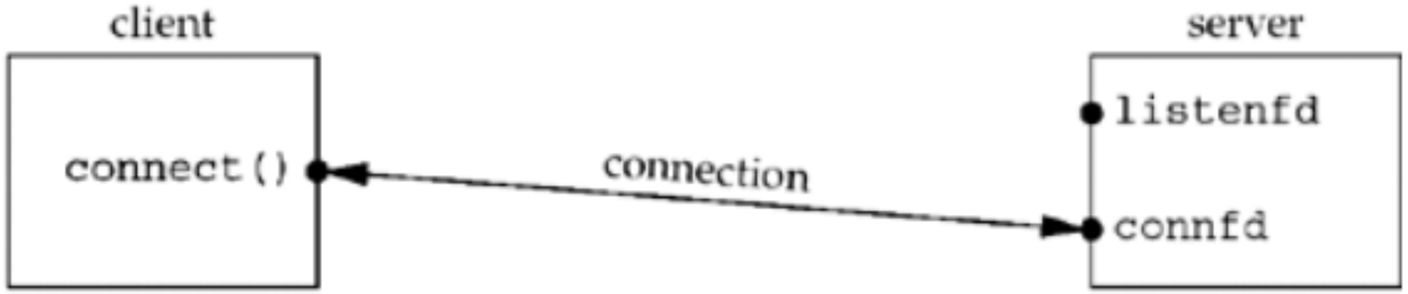
- *fork* a child process to handle each client.

```
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 */
}
```

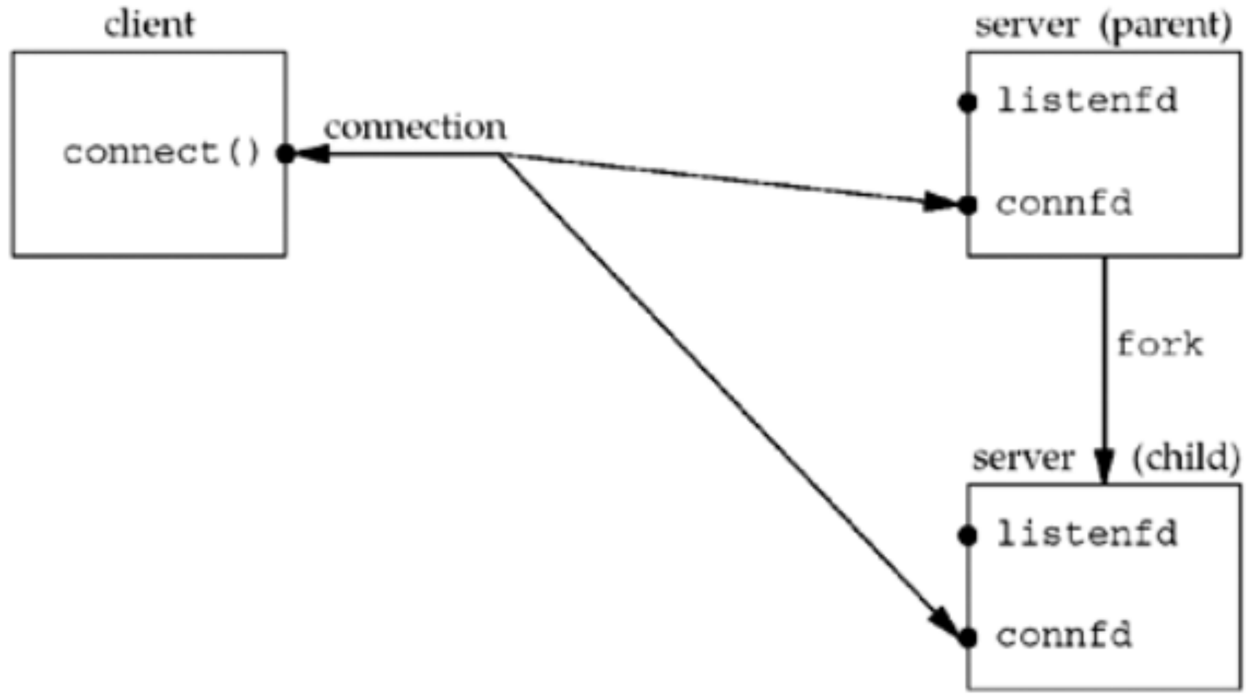
Status of client/server before call to *accept* returns.



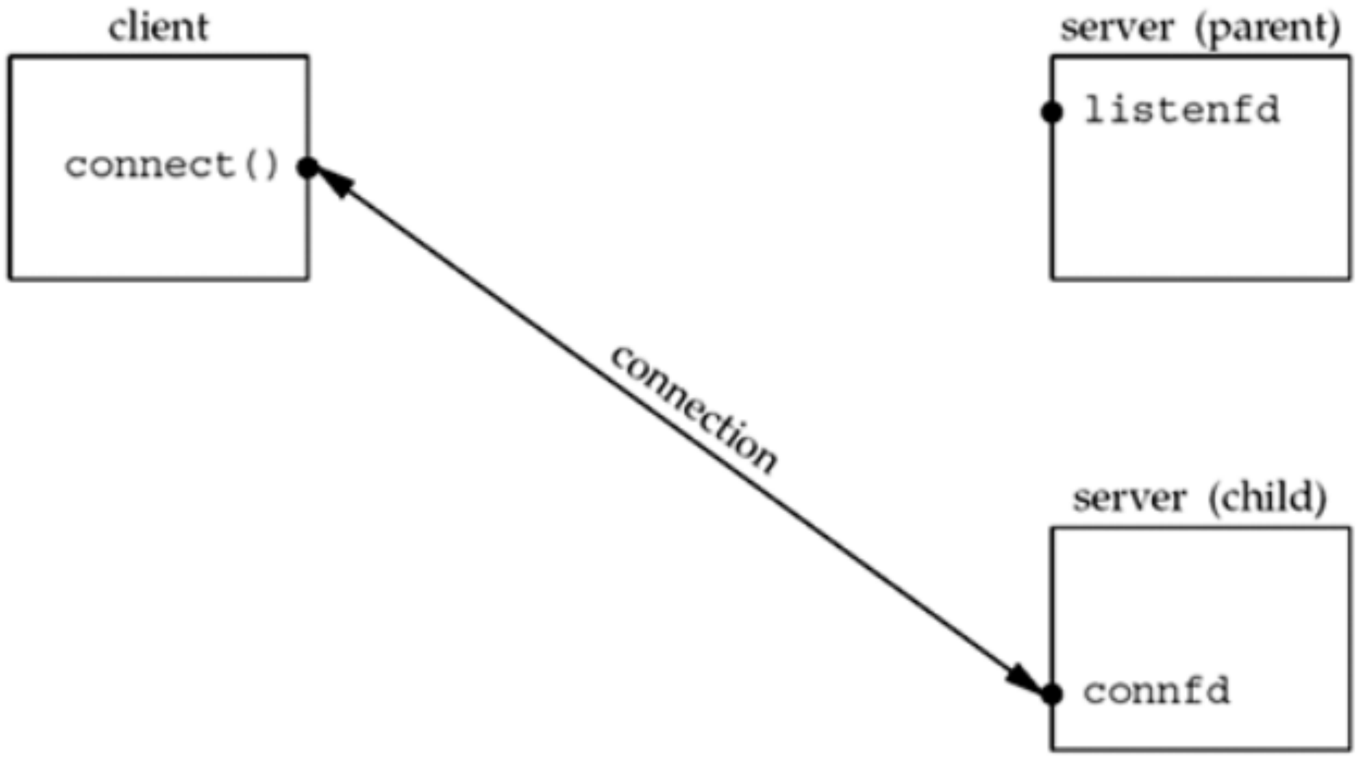
Status of client/server after return from *accept*.



Status of client/server after fork returns.



Status of client/server after parent and child close appropriate sockets.



close Function

- The normal Unix close function is also used to close a socket and terminate a TCP connection.

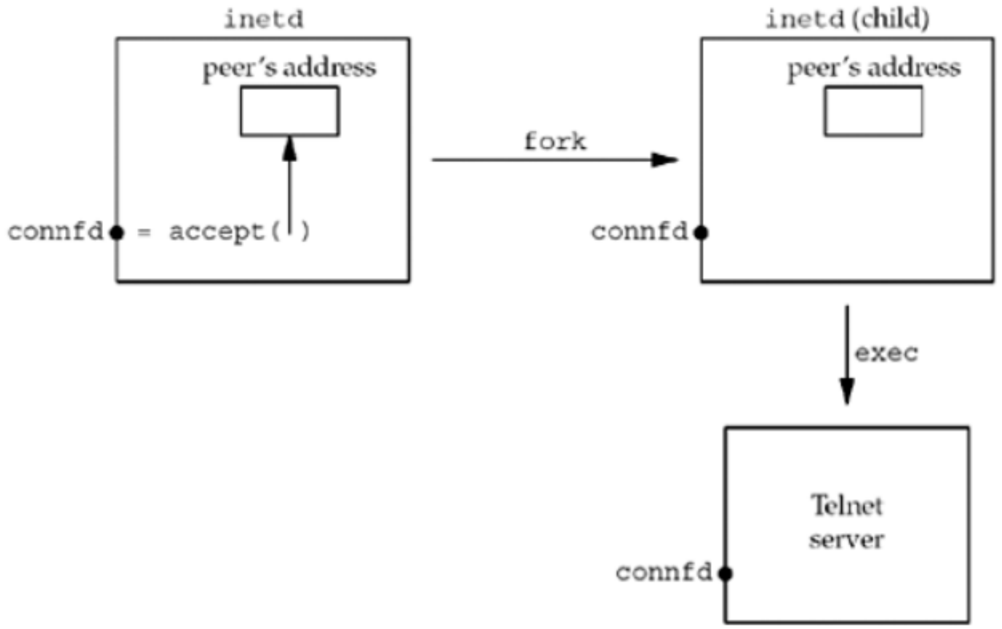
```
#include <unistd.h>
int close (int sockfd);
```

- Returns: 0 if OK, -1 on error
- If the parent doesn't close the socket, when the child closes the connected socket, its reference count will go from 2 to 1 and it will remain at 1 since the parent never closes the connected socket. This will prevent TCP's connection termination sequence from occurring, and the connection will remain open.

getsockname and *getpeername* Functions

```
#include <sys/socket.h>
int getsockname(int sockfd, struct sockaddr
  *localaddr, socklen_t *addrlen);
int getpeername(int sockfd, struct sockaddr
  *peeraddr, socklen_t *addrlen);
```

- Both return: 0 if OK, -1 on error


inetd
peer's address
connfd = accept( )
fork
inetd (child)
peer's address
connfd
exec
Telnet server
connfd