Computer Network Programming

TCP Sockets

Dr. Sam Hsu
Computer Science & Engineering
Florida Atlantic University

TCP Sockets

- The socket() Function
- The connect() Function
- The bind() Function
- The listen() Function
- The accept() Function
- The fork() and exec() Function
- Concurrent Servers
- Server/Client Connection Status & Example
- The close() Function
- The getsockname() and getpeername() Functions
- Wrapper Functions

The socket() Function

- To open a socket for performing network I/O.

```
#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
AF_ROUTE	Routing protocols
AF_KEY	Key socket

type	Description
SOCK_STREAM	stream socket (TCP/SCTP)
SOCK_DGRAM	datagram socket (UDP)
SOCK_SEQPACKET	sequenced packet socket (SCTP)
SOCK_RAW	raw socket (talk to IP directly)

- For *protocol*, one may select the system default (0), since not all combinations of *family* and *type* are valid.

The connect() Function (1/3)

- Is used by a client to establish a connection with a server via a 3-way handshake.

```
#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.
  - *servaddr* contains the IP address and port number of the server.
  - *addrlen* has the length (in bytes) of the server socket address structure.

The connect() Function (2/3)

- This function returns only when the connection is established or an error occurs.
- Some possible errors:
  - If the client TCP receives no response to its SYN segment, ETIMEDOUT is returned.
    - The connection-establishment timer expires after 75 seconds (4.4 BSD).
      - The client will resend SYN after 6 seconds later, and again another 24 seconds later. If no response is received after a total of 75 seconds, the error is returned.

The connect() Function (3/3)

- If a reset (RST) is received from server, ECONNREFUSED is returned. This is a **hard error**.
  - This indicates that there is no process running at the server host at the port specified.
- If an ICMP "destination unreachable" is received from an intermediate router, EHOSTUNREACH or ENETUNREACH is returned. This is a **soft error.**
  - Upon receiving the first ICMP message, the client kernel will keep sending SYNs at the same time intervals as mentioned earlier, until after 75 seconds have elapsed (4.4BSD).

The bind() Function (1/2)

- Is used primarily by a server to assign 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
```

  - *sockfd* is a socket descriptor returned by the socket() function.
  - *myaddr* is a pointer to a protocol-specific address. With TCP, it has the IP address and port number of the server.
  - *addrlen* has the length (in bytes) of the server socket address structure.

The bind() Function (2/2)

- IP address/Port number assignment:

Process specifies		Results
IP address	Port	
Wildcard	0	Kernel chooses IP address and port
Wildcard	Nonzero	Kernel chooses IP address, process specifies port
Local IP addr	0	Process specifies IP address, kernel chooses port
Local IP addr	Nonzero	Process specifies IP address and port

  - Wildcard address: INADDR_ANY (IPv4), in6addr_any (IPv6).
  - TCP servers typically bind their well-known port, and clients let the kernel choose an ephemeral port.

The listen() Function (1/2)

- Is used by a server to convert an unconnected socket to a passive socket.

```
#include <sys/socket.h>

int listen(int sockfd,  int backlog);

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

- *sockfd* is a socket descriptor returned by the socket() function.
- *backlog* specifies the maximum number of connections the kernel should queue for this socket.

The listen() Function (2/2)

- For a given listening socket, the kernel maintains 2 queues.
  - An *incomplete connection queue*
    - It contains an entry for each SYN received from a client, for which the server is awaiting completion of the TCP 3-way handshake.
  - A *completed connection queue*
    - It contains an entry for each client with whom the TCP 3-way handshake process has completed.
- *backlog* is the sum of these two queues.
  - *backlog* has not been well-defined so far.
  - One may select any number other than 0.

Two TCP Waiting Queues

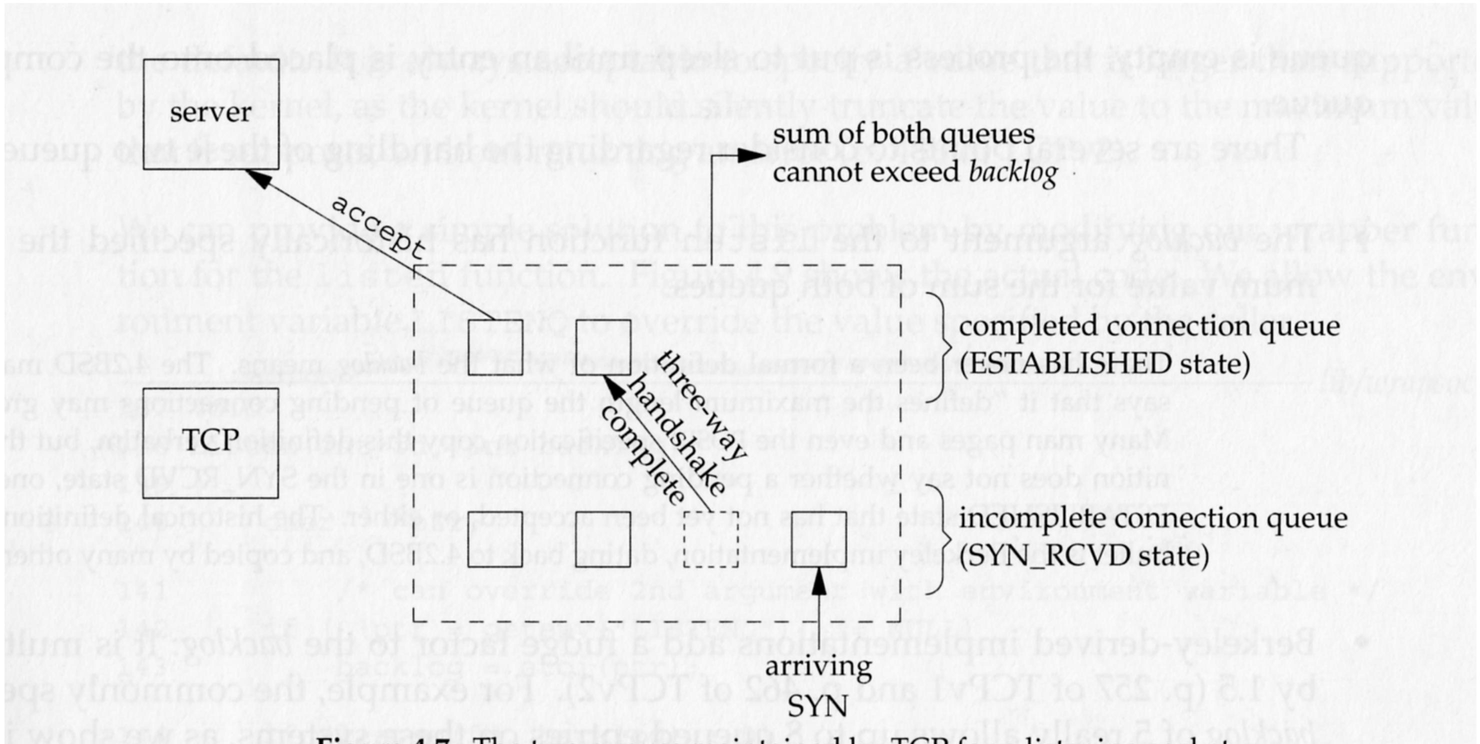
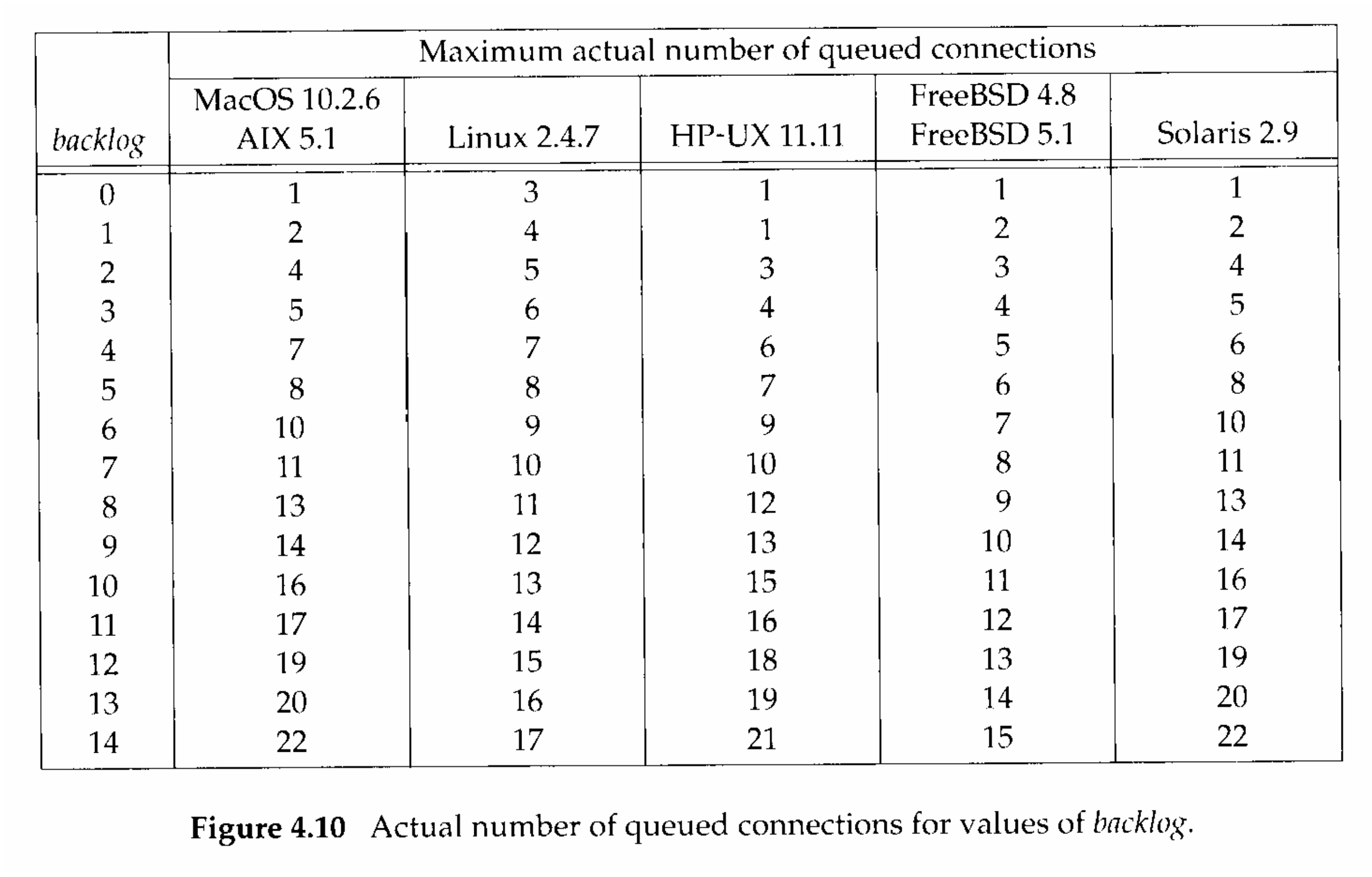


Figure 4.7 The two queues maintained by TCP for a listening socket.

Connections for Various *backlogs*



backlog	Maximum actual number of queued connections				
	MacOS 10.2.6 AIX 5.1	Linux 2.4.7	HP-UX 11.11	FreeBSD 4.8 FreeBSD 5.1	Solaris 2.9
0	1	3	1	1	1
1	2	4	1	2	2
2	4	5	3	3	4
3	5	6	4	4	5
4	7	7	6	5	6
5	8	8	7	6	8
6	10	9	9	7	10
7	11	10	10	8	11
8	13	11	12	9	13
9	14	12	13	10	14
10	16	13	15	11	16
11	17	14	16	12	17
12	19	15	18	13	19
13	20	16	19	14	20
14	22	17	21	15	22

Figure 4.10 Actual number of queued connections for values of *backlog*.

SYN Flooding

- SYN Flooding: A type of attack aiming at *backlog*.
  - A program sends bogus SYNs at a high rate to a server, filling the incomplete connection queue for one or more TCP ports.
  - The source IP address of each SYN is set to a random number so that the server's SYN/ACK goes nowhere.
    - This is called *IP spoofing*.
  - This leaves no room for legitimate SYNs.
    - TCP ignores an arriving SYN if the queues are full.
- *backlog* should specify just the max number of completed connections for a listening socket.

The accept() Function (1/2)

- Is called by a server to return a new descriptor, created automatically by the kernel, for the connected socket.

```
#include <sys/socket.h>

int accept(int sockfd,  struct sockaddr *cliaddr,
                 socklen_t  *addrlen);

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

- *sockfd* is a socket descriptor returned by the socket() function.
- *cliaddr* contains the IP address and port number of the connected client. (a value-result argument)
- *addrlen* has the length (in bytes) of the returned client socket address structure. (a value-result argument)

The accept() Function (2/2)

- The new socket descriptor returned by accept() is called a *connected socket*, whereas the one returned by socket() is called a *listening socket*.
  - A given server usually creates only one *listening socket*. It exists for the lifetime of the server.
  - A *connected socket* is created for each client connection that is accepted. It exists only for the duration of the connection.
- Both *cliaddr* and *addrlen* may be set to the NULL pointer, if the server is not interested in knowing the identity of the client.

The UNIX fork() Function (1/2)

- Is used in UNIX to create a new process.

```
#include <unistd.h>

pid_t fork(void);

     Returns: 0 in child, process ID of child in parent, -1 on error
```

- *fork()* is called once, but returns *twice*.
  - Once in the calling process, called the *parent*.
  - Once in the newly created process, called the child.
- A parent may have more than 1 child process.

The UNIX fork() Function (2/2)

- All descriptors open in the parent before fork() are shared with the child after fork().
  - The connected socket is then shared between the parent and the child.
- Two typical uses of fork():
  - A process makes a copy of itself so that one copy can handle one operation, and the other copy does something else.
    - This is typical for network servers.
  - A process want to execute a new program by calling exec() in the child process.
    - User commands in UNIX are typically handled this way.
- fork() can be used to implement *concurrent* servers.

The UNIX exec() Function (1/3)

- Is used in UNIX to execute a program.
- Is a family name for six like functions virtually doing the same thing, only slightly different in syntax.

```
#include <unistd.h>

int  execl(…), execv(…), execle(…), execve(…),
     execlp(…), execvp(…);

                        Returns: -1 on error, no return on success
```

  - Descriptors open in the process before calling exec() normally remain open in the new program.

The UNIX exec() Function (2/3)

- Meaning of different letters:

  l: needs a list of arguments.

  v: needs an *argv[]* vector (*l* and *v* are mutually exclusive).

  e: needs an *envp[]* array.

  p: needs the PATH variable to find the executable file.

The UNIX exec() Function (3/3)

$$\begin{tikzcd}
\boxed{\text{execlp}} \arrow[d, "\text{argv}"] & \boxed{\text{execl}} \arrow[d, "\text{argv}"] & \boxed{\text{execle}} \arrow[d, "\text{argv}"] \\
\boxed{\text{execvp}} \arrow[r, "\text{PATH}"'] & \boxed{\text{execv}} \arrow[r, "\text{envp}"'] & \boxed{\text{execve}}
\end{tikzcd}$$

Relationship of the *exec()* functions.

Concurrent Servers

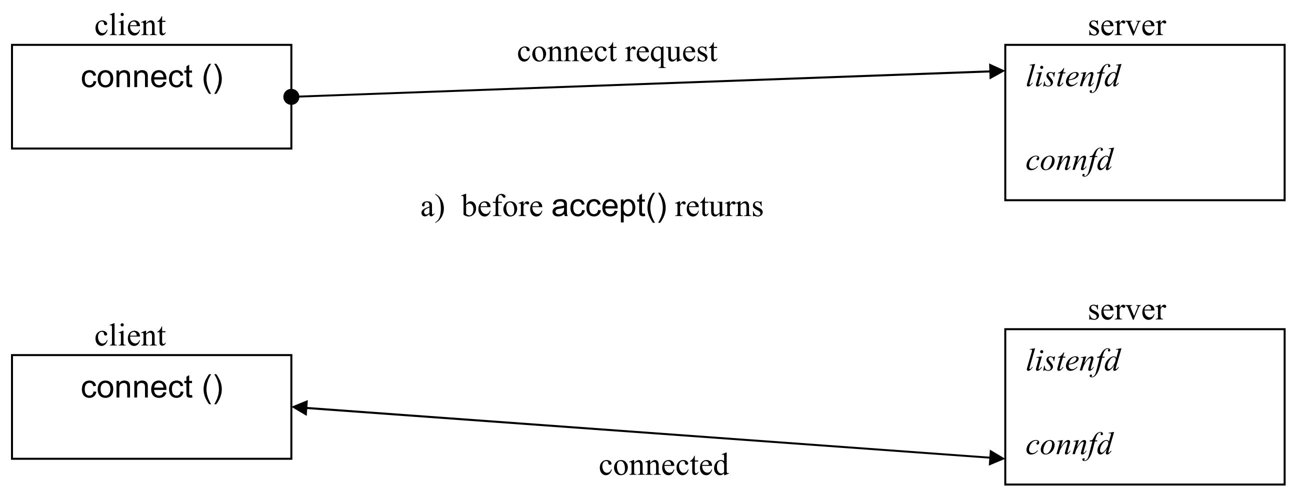
- Outline of a typical concurrent server (Fig 4.13)

```
pid_t       pid;
int         listenfd, connfd;

listenfd = Socket (...);
            /* fill in socket_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 */
}
```

Server/Client Connection Status (1/2)

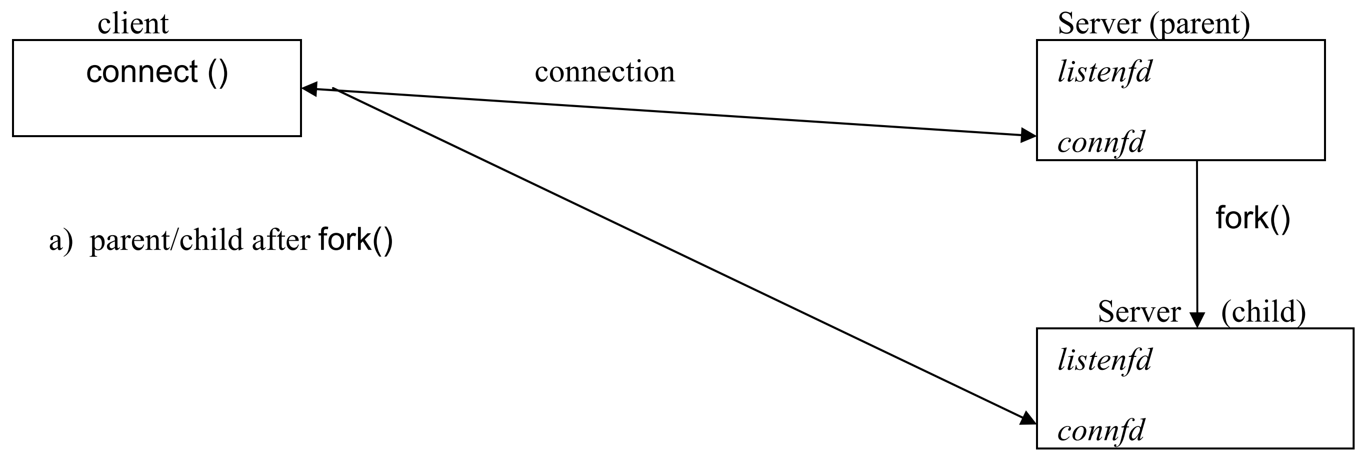


a) before accept() returns

b) after accept() returns

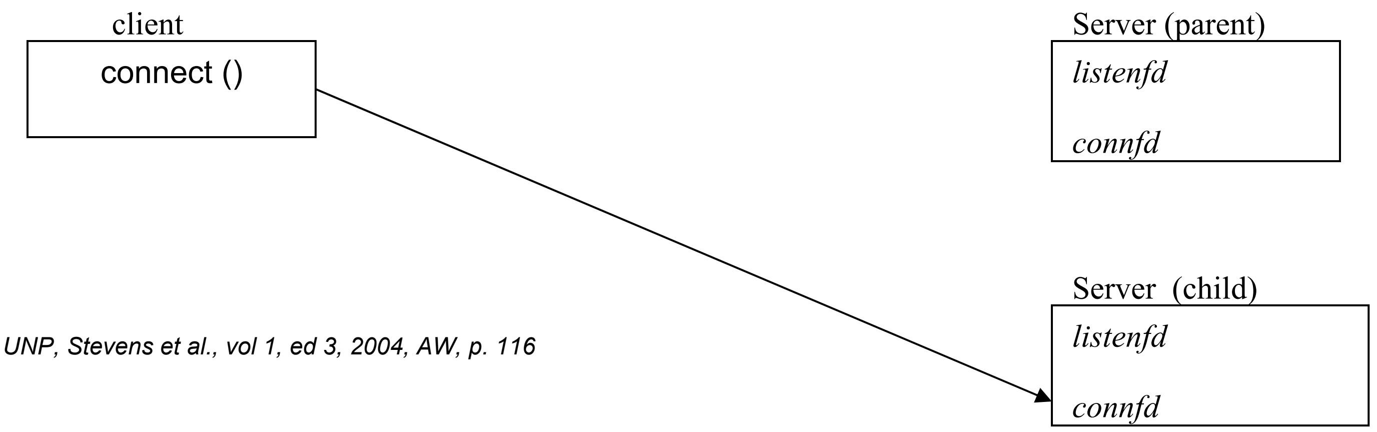
Ref: UNP, Stevens et al., vol 1, ed 3, 2004, AW, p. 115

Server/Client Connection Status (2/2)



a) parent/child after fork()

b) parent/child after closing appropriate sockets



Ref: UNP, Stevens et al., vol 1, ed 3, 2004, AW, p. 116

Server/Client Example (1/2)

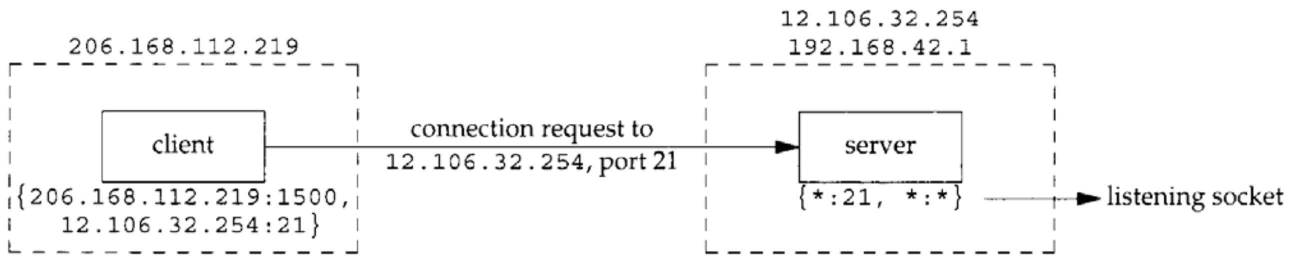


Figure 2.12 Connection request from client to server.

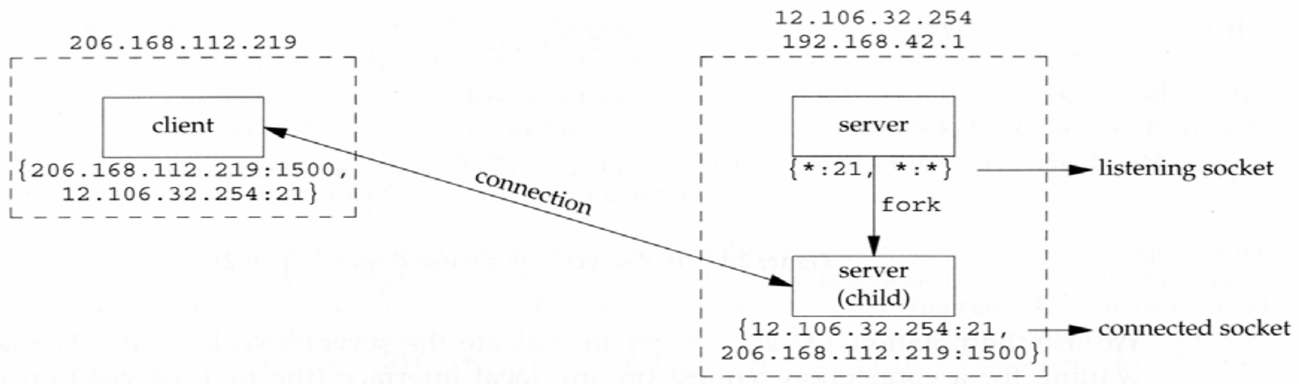


Figure 2.13 Concurrent server has child handle client.

Server/Client Example (2/2)

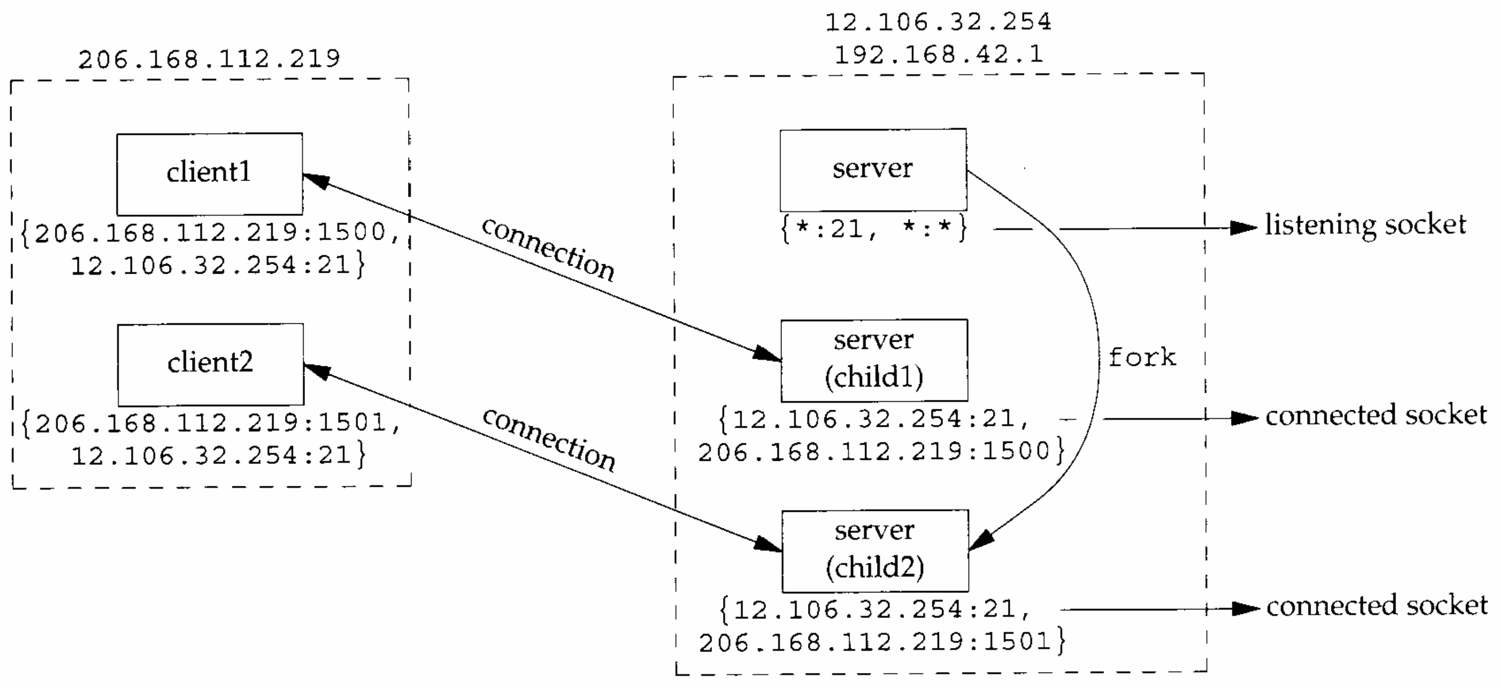


Figure 2.14 Second client connection with same server.

The UNIX close() Function (1/2)

- Is used to close a socket and terminate a TCP connection.

```
#include <unistd.h>

int close(int sockfd);

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

- *sockfd* is a socket descriptor returned by the socket() function.

The UNIX close() Function (2/2)

- close() marks socket as closed and returns immediately.
  - *sockfd* is no longer usable.
  - TCP continues to try sending unsent data.
    - Hardly knows whether it was ever successful.
- close() simply decrements the reference count.
  - Socket goes away when the reference count becomes 0.
- What if the parent does not close the connected socket for the client?
  - May run out of descriptors eventually.
  - No client connection will be terminated.
    - Reference count remains at 1.

getsockname()/getpeername() (1/2)

- Is used to get the local/foreign protocol address associated with a socket.

```
#include <sys/socket.h>

int getsockname(int sockfd,  struct sockaddr *localaddr,
                             socklen_t  *addrlen);

int getpeername(int sockfd,  struct sockaddr *peeraddr,
                             socklen_t  *addrlen);

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

  - *sockfd* is a socket descriptor returned by the socket() call.
  - All *localaddr/peeraddr/addrlen* are value-result arguments.

getsockname()/getpeername() (2/2)

- Reasons for using these two functions:
  - To use getsockname() by a TCP client that does not call bind() to get the local IP address and port number assigned.
  - To use getsockname() by a TCP server that called bind() with a port number 0 to get the local port number assigned.
  - To use getsockname() by a TCP server that called bind() with the wildcard IP address to get the local IP address assigned.
  - To use getsockname() to obtain the address family of a socket.
  - To use getpeername() by an *exec*ed TCP server that called accept() to get the identity of the client (its IP address and port number).

Wrapper Functions

- A wrapper function provides additional features to the function it embraces.
  - It enhances the functionality of the wrapped function.
- Quite a few wrapper functions have been defined in this textbook.
  - Primarily for better error handling purposes.
  - Each wrapper function begins with an uppercase letter.
    - A wrapper function calls a function whose name is the same but begins with the lowercase letter.
  - Each wrapper function performs the actual function call, tests the return value and terminates on an error.

Socket(): A Wrapper Example

- The wrapper function for socket()

```
#include “unp.h”

int  Socket(int family, int type, int protocol)
{
        int  n;

        if ( (n = socket(family, type, protocol)) < 0)
                err_sys("socket error");
        return(n);
}
/* end Socket */
```

Pthread_mutex_lock(): Another

- The wrapper function for pthread_mutext_lock()

```
#include “unp.h”

void  Pthread_mutex_lock(pthread_mutex_t *mptr)
{
        int   n;

        if ( (n = pthread_mutex_lock (mptr)) == 0)
                return;
        errno = n;
        err_sys(" pthread_mutex_lock error");
 }
/* end Pthread_mutex_lock */
```

Reading Assignment

- Read Chapter 4.