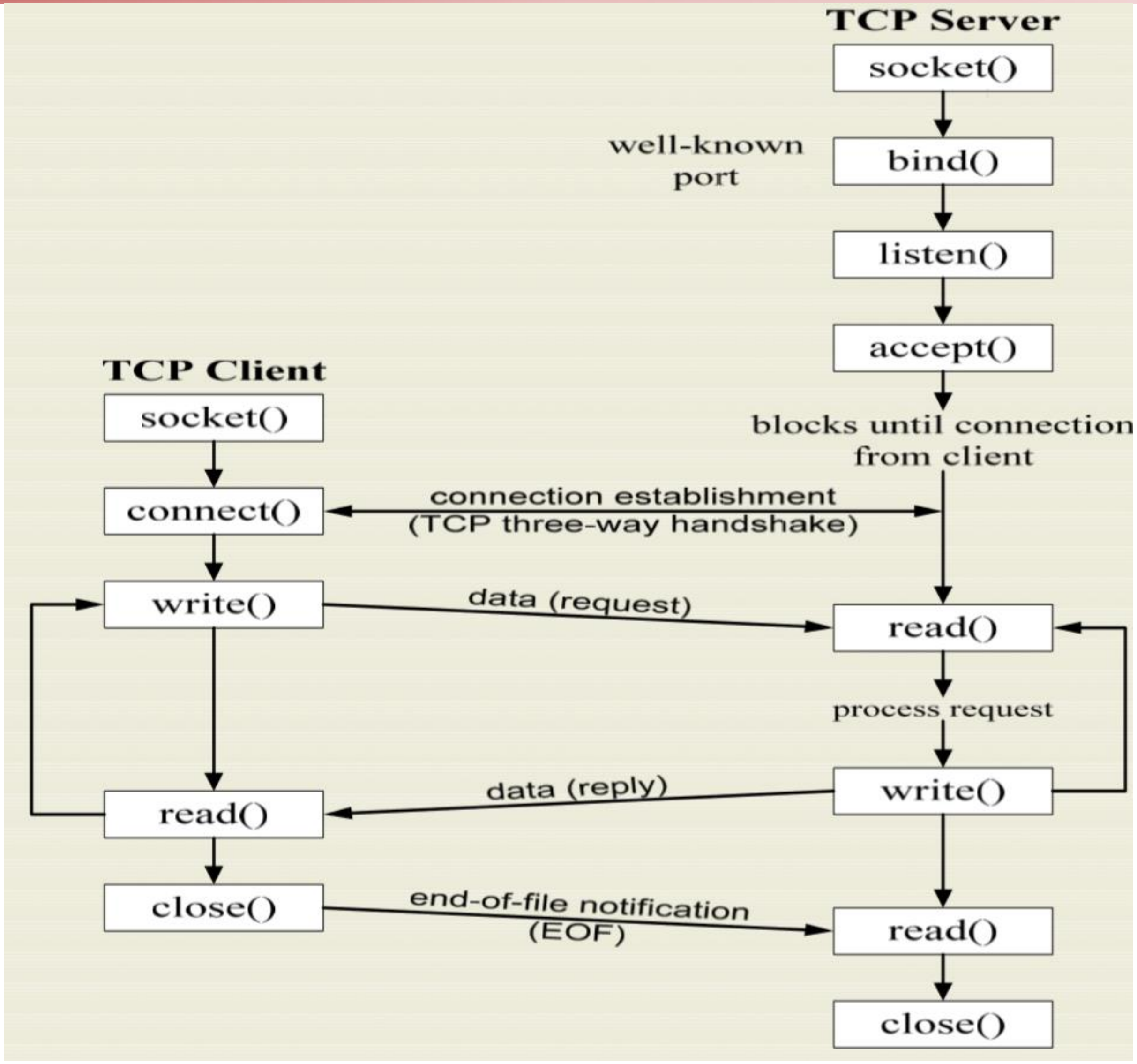
Chapter 4
Elementary TCP Sockets

Index

Introduction



socket Function

```
#include <sys/socket.h>
int socket (int family, int type, int protocol);
//Returns: non-negative descriptor if OK, -1 on error
```

family : protocol family constants
type : type of socket
protocol : protocol (TCP, UDP, SCTP)

family	Description
AF_INET	IPv4 protocols
AF_INET6	IPv6 protocol
AF_LOCAL	Unix domain protocols
AF_ROUTE	Routing sockets
AF_KEY	Key sokets

type	Description
SOCK_STREAM	Stream socket
SOCK_DGRAM	Datagram socket
SOCK_SEQPACKET	Sequenced packet socket
SOCK_RAW	Raw socket

Not all combination of socket *family* and *type* are valid

socket Function

Family/type	AF_INET	AF_INET6	AF_LOCAL	AF_ROUT	AF_KEY
STREAM	TCP | SCTP	TCP | SCTP	O		
DGRAM	UDP	UDP	O		
SEQPACKET	SCTP	SCTP	O		
RAW	IPv4	IPv6		O	O

On success, the *socket* Function returns integer value.
Called *socket descriptor / socketfd*

connect Function

```c
#include <sys/socket.h>
int connect(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);
// Returns: 0 if OK, -1 on error
```

Used by TCP client to establish a connection with a TCP server.

sockfd	: socket descriptor
servaddr	: socket address structure
addrlen	: size of socket address structure

The client does not have to call *bind* before calling connect.

return value :

success : 0

Error : -1

Error code of *connect* Function

ETIMEOUT

: client received no reponse to its SYN gegment

ECONNREFUSED

: no process is waiting for connections at server

: response of server is RST (reset)

: hard error

EHOSTUNREACH / ENETUNREACH

: response of router is ICNP (destination unreachable)

: soft error

: can fixed after seconds

bind Function

```
#include <sys/socket.h>
int bind (int sockfd, const struct sockaddr *myaddr, socklen_t addrlen);
// Returns: 0 if OK,-1 on error
```

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

protocol address : combination of IP address and port number.

sockfd	: socket descriptor
myaddr	: address structure
addrlen	: size of address structure

calling bind lets us specify a port number, an IP address, both, or neither.

bind Function

IP address	port	Result
Wildcard	0	Kernel choose IP address and port
Wildcard	Non-zero	Kernel choose IP address, process specify port
Local IP address	0	Process choose IP address, kernel specify port
Local IP address	Non-zero	Process specifies IP address and port

Port num = 0 -> kernel choose ephemeral port

Wildcard IP -> kernel does not choose the local IP address until either the socket is connected (TCP) or datagram is sent on the socket (UDP).

listen Function

```
#include <sys/socket.h>
int listen (int sockfd, int backlog);
//Returns: 0 if OK, -1 on error
```

called only by a TCP server and it performs two actions.

1. When socket create, it assumed as active socket.
2. listen function converts an unconnected socket into a passive socket
   -> socket moves CLOSED state to LISTEN state.

sockfd	: socket descriptor
backlog	: maximum number of connections the kernel should queue

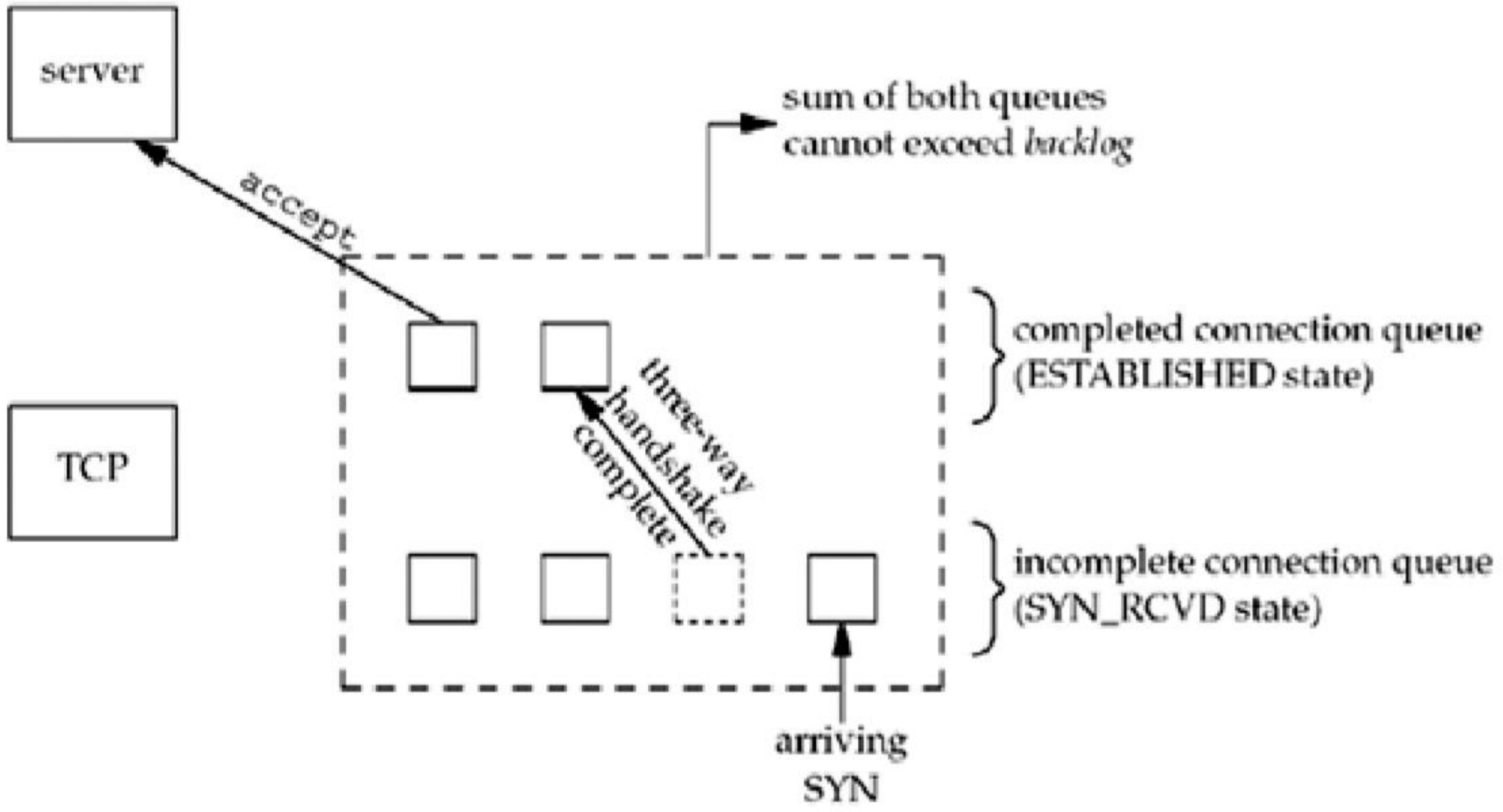
listen Function

kinds of *backlog* queue

1. incomplete connection queue
: contains entry for each SYN that has arrived from a client before completion of the TCP three-way handshake.
socket state : SYN_RCVD state

2. completed connection queue
: contains entry for each SYN, TCP three-way handshake has completed.
socket state : ESTABLISHED state

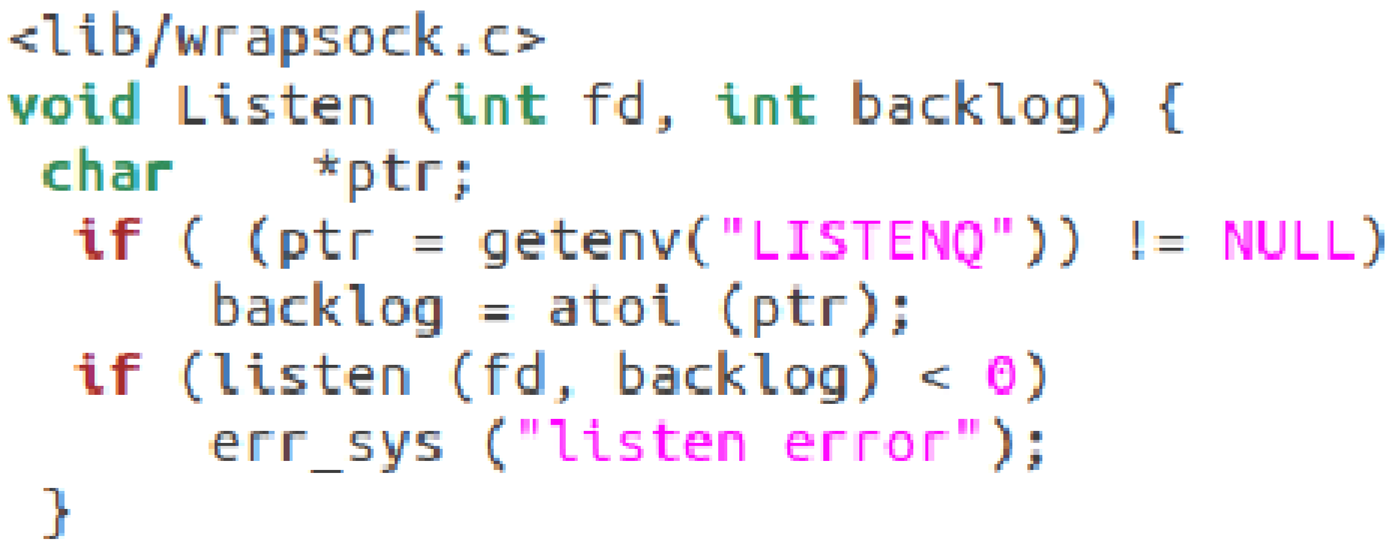
listen Function


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

listen Function

backlog : sum of both queues

Historically, backlog if 5
but, HTTP servers specify a larger value. -> problem occurred



```
<lib/wrapsock.c>
void Listen (int fd, int backlog) {
 char     *ptr;
  if ( (ptr = getenv("LISTENQ")) != NULL)
      backlog = atoi (ptr);
  if (listen (fd, backlog) < 0)
      err_sys ("listen error");
 }
```

getenv(“name”) : read defined value of name
atoi() : convert char type to int type

accept Function

```
#include <sys/socket.h>
int accept (int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);
//Returns: non-negative descriptor if OK, -1 on error
```

called by a TCP server
 return front of the completed connection queue

sockfd	: socket descriptor (listen socket)
cliaddr	: client address
addrlen	: size of client socket address structure

accept function return connect socket

listen socket : made only one by parent process
connect socket : creates for each client connection

fork Function

```
#include <unistd.h>
pid_t fork(void);
//Returns: 0 in child, process ID of child in parent, -1 on error
```

fork function create new process.

process that calling fork function called parent process.
new process called child process.

parent process return PID of child process.
child process return 0.

connected socket is shared by parent and child process

exec Function

```
#include <unistd.h>
int execl (const char *pathname, const char *arg0, ... /* (char *) 0 */ );
```

exec fuction replaces the current process image with the new program file.
PID does not change.

Concurrent Servers

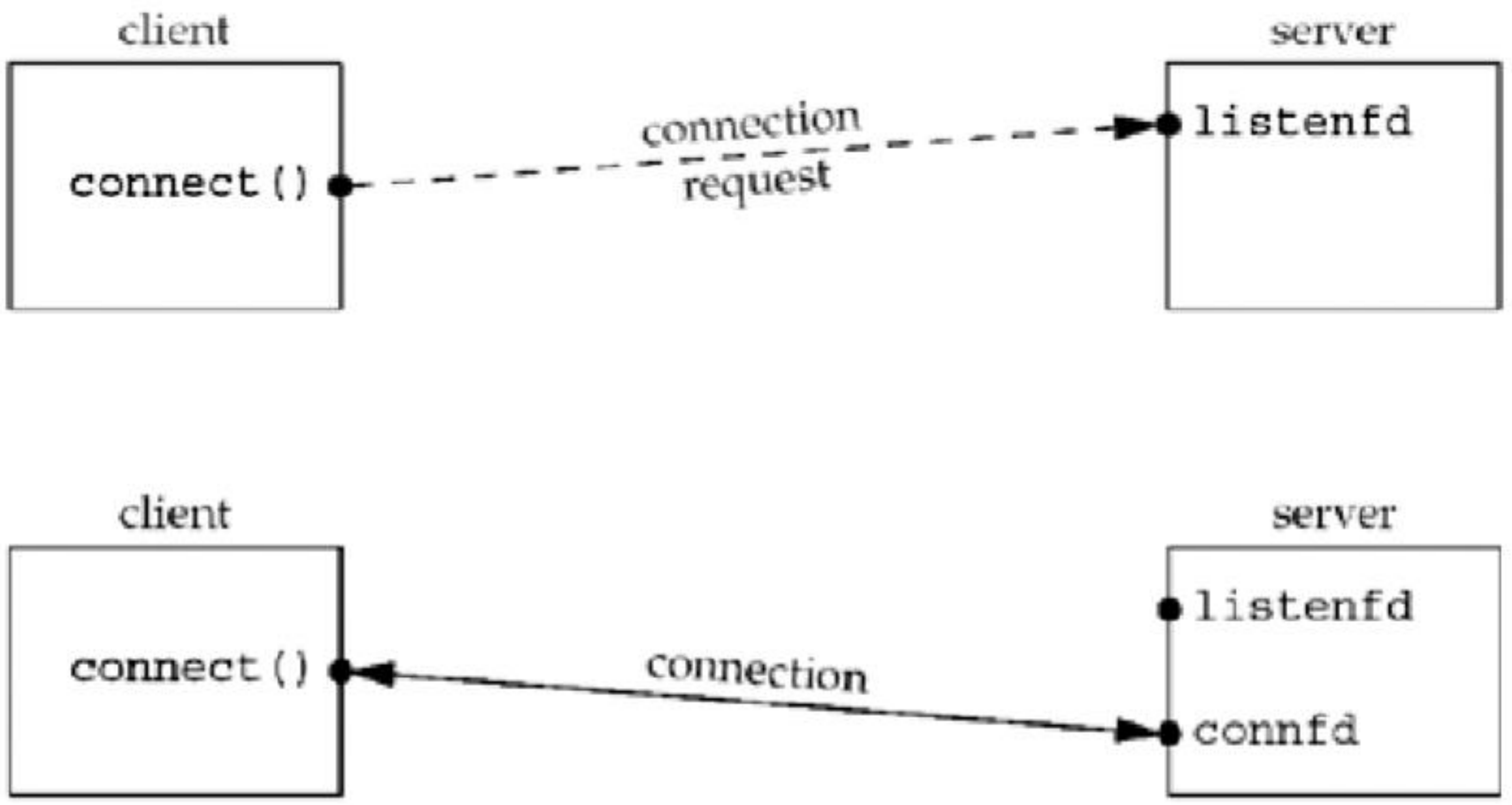
```
pid_t pid;
int   listenfd,  connfd;

 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 */
 }
```

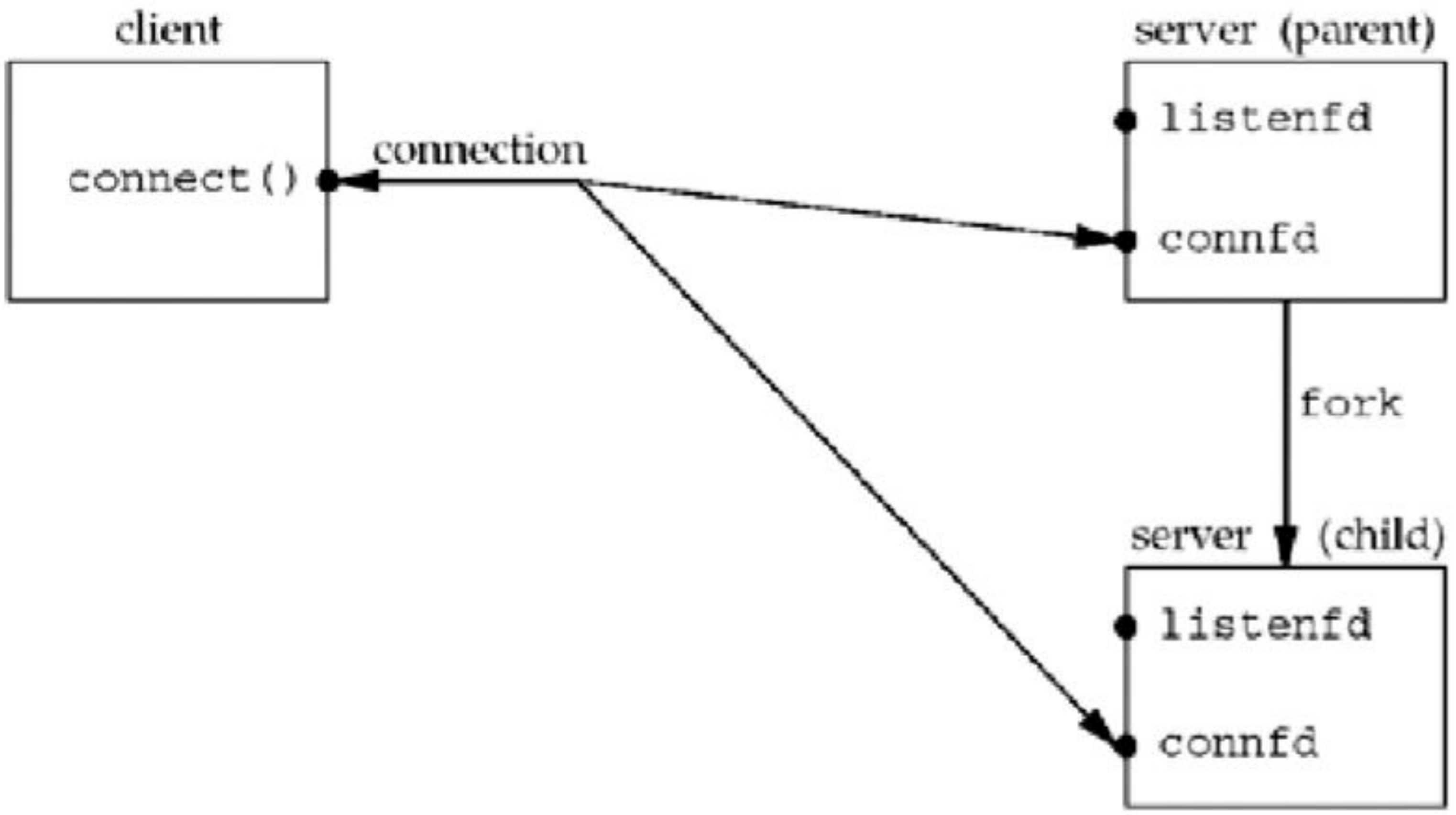
child process -> do client service

parent process -> close connected socket after child process died

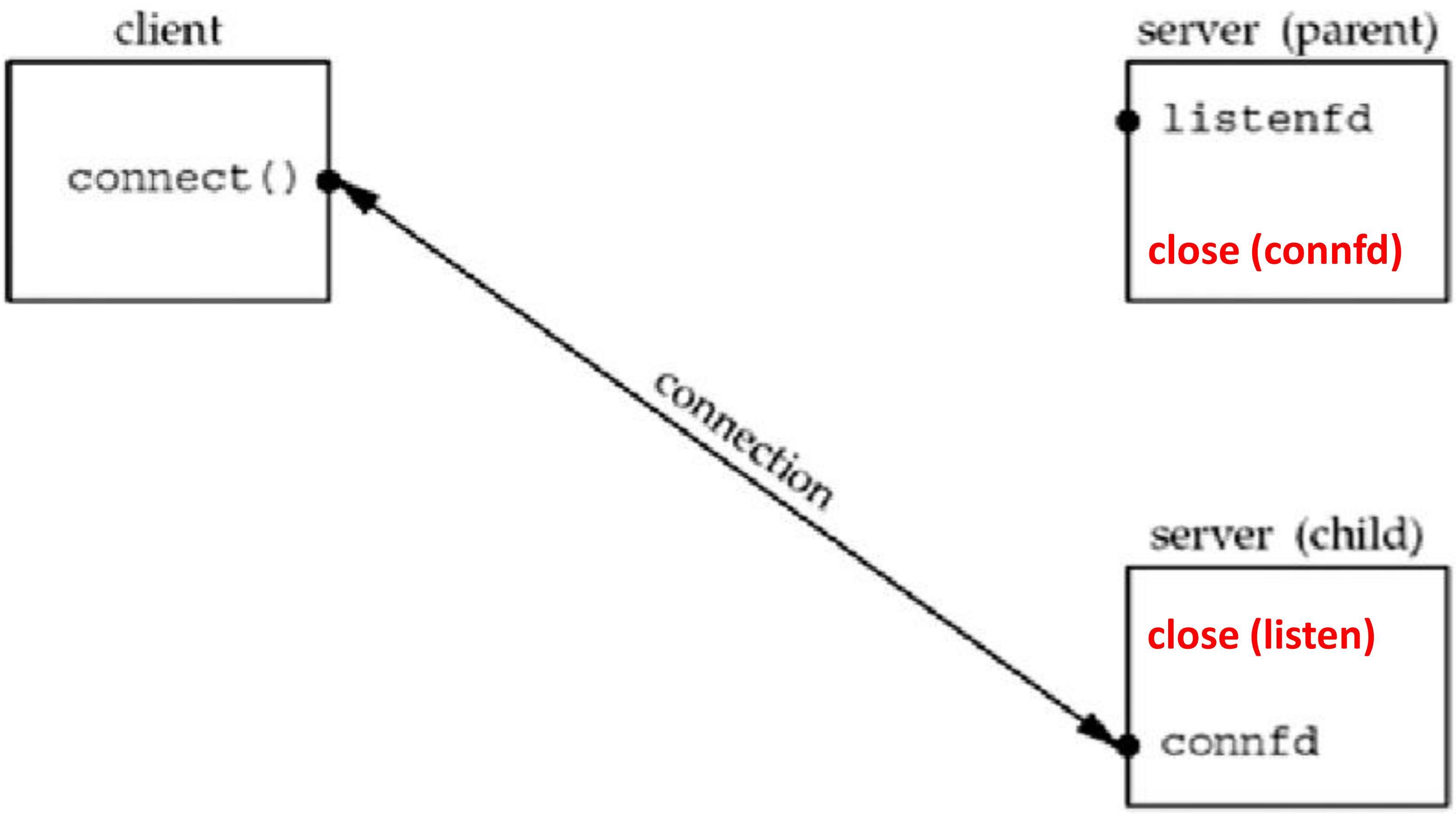
Concurrent Servers



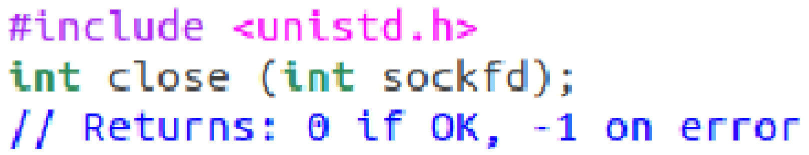
Concurrent Servers



Concurrent Servers



close Function

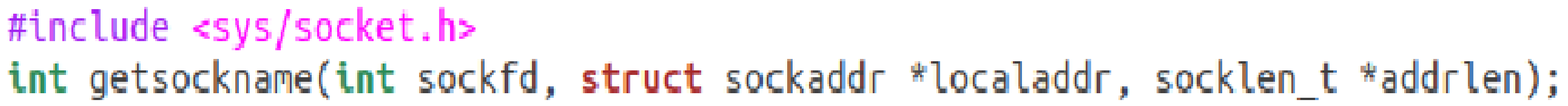


```
#include <unistd.h>
int close (int sockfd);
// Returns: 0 if OK, -1 on error
```

close function close socket and return to the process immediately.

after close, socket descriptor cannot be used as an argument to read, write

getsockname Function



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

Obtains the address assigned to the socket descriptor.

sockfd : socket descriptor
localaddr : buffer for save address
addrlen : size of buffer

getsockname Function

requirement for using getsockname function

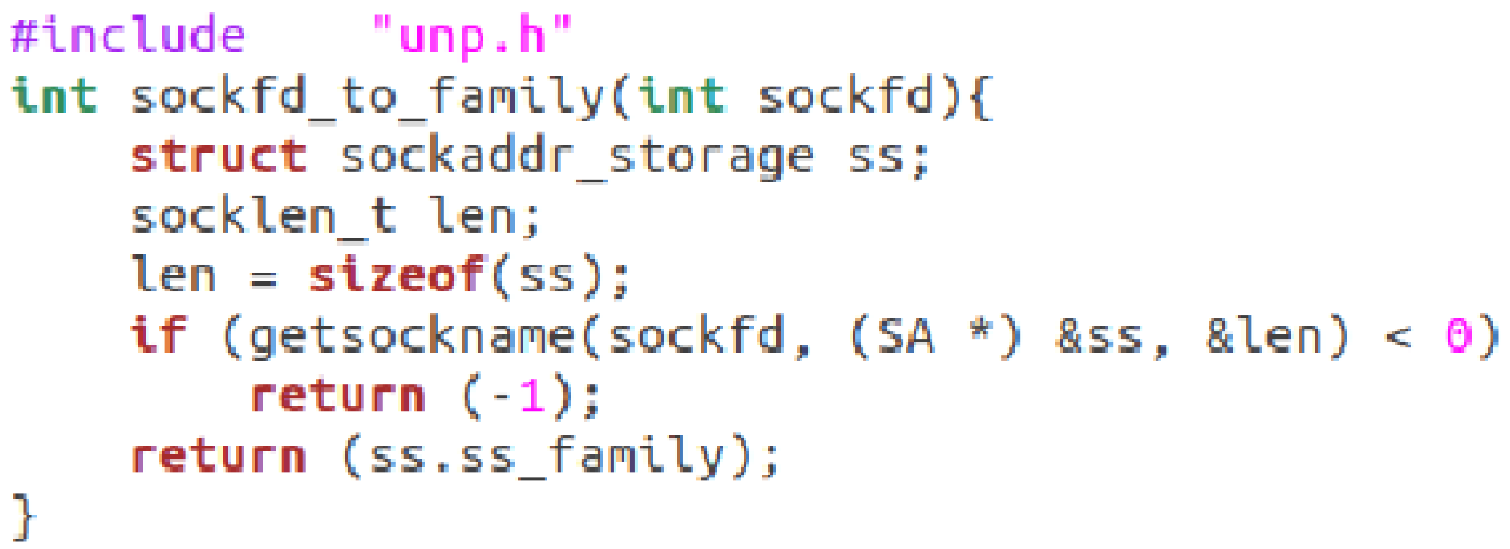
- successfully connected, does not call bind , getsockname returns the local IP address and local port number assigned to the connection by the kernel.

- After calling bind with a port number of 0, getsockname returns the local port number that was assigned.

- getsockname can be called to obtain the address family of a socket

- In a TCP server that binds the wildcard IP address the server can call getsockname to obtain the local IP address assigned to the connection. The socket descriptor argument in this call must be that of the connected socket, and not the listening socket.

getsockname Function



```
#include     "unp.h"
int sockfd_to_family(int sockfd){
    struct sockaddr_storage ss;
    socklen_t len;
    len = sizeof(ss);
    if (getsockname(sockfd, (SA *) &ss, &len) < 0)
        return (-1);
    return (ss.ss_family);
}
```

getpeername Function

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

Obtains the address assigned to the foreign socket descriptor that connected and accepted.

sockfd	: socket descriptor
peeraddr	: buffer for save address
addrlen	: size of buffer