

Network Programming

Dr. Thaier Hayajneh

Computer Engineering Department

TCP Sockets

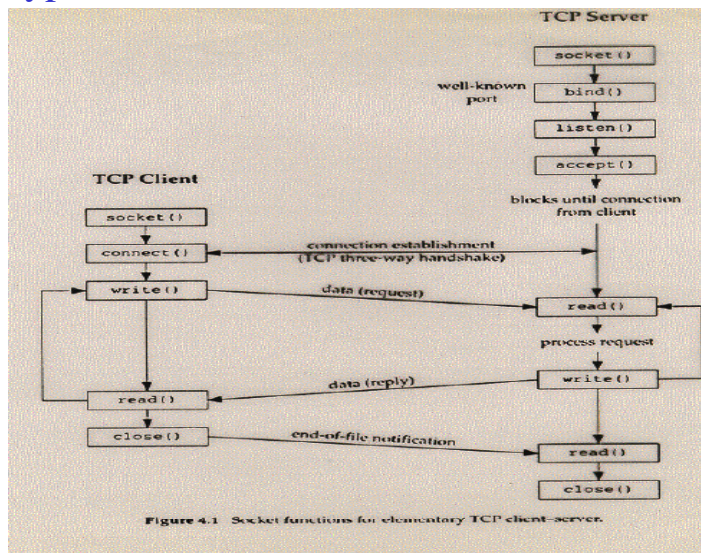
1

Outline

- TCP Sockets
 - Information to write a complete TCP client and server
 - Concurrent TCP sockets

2

Typical Scenario between TCP client/server



3

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 (AF_INET → IPv4 protocols, AF_INET6 → IPv6 Protocols) (see Fig. 4.2)

type

(SOCK_STREAM → stream socket, SOCK_DGRAM → Datagram socket) (see Fig. 4.3)

protocol

Use 0 to get system's default given combination of family and type (see Fig. 4.4)

4

connect Function 1/3

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

```
int connect (int sockfd, const struct sockaddr * servaddr , socklen_t  
addrlen)
```

// returns 0 if OK, -1 on error

- No need to specify client's source IP address or port
 - Kernel will choose an ephemeral port and source IP if necessary
- Connect function initiates TCP's three-way handshake
- Function returns only when connection is established or an error occurs

5

connect Function 2/3

Several possible errors (The following numbers for 4.4 BSD)

Send SYN....& after 6 seconds..& after 24 seconds

if after a total of 75 seconds no SYN-ACK received

➤ETIMETIMEOUT is returned

if server responds with RST

➤no process waiting at port → *hard error*

➤ECONNREFUSED is returned

if a router returns ICMP destination unreachable (*soft error*)

➤send after 6 and 24 seconds and if no connection after 75 seconds

➤EHOSTUNREACH is returned

- You can't **reconnect** the socket to another address unless you close and call socket again.

6

connect Function 3/3

•Try it out with the daytime TCP client/server

- Successful connection
- IP address on local subnet, but host nonexistent
 - ✓ *Connection timed out*
- Correct local IP address, not running a daytime server
 - ✓ *Connection refused*
- Unreachable Internet IP address
 - ✓ Intermediate router will return ICMP error
 - ✓ *No route to host*

•Reasons for RST segment

- SYN arrives for a port with no listening server
- TCP wants to abort an existing condition
- TCP receives a segment for a connection that does not exist

7

bind Function 1/2

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

```
int bind (int sockfd, const struct sockaddr * myaddr , socklen_t  
addrlen)
```

// assigns a local protocol address→ returns 0 if OK, -1 on error

Server (see *daytimetcpsrv3.c* in *intro* chapter)

- Normally bind to a well know port & *INADDR_ANY*
- Using port 0: kernel choose a free port and we use *getsockname* to find the selected port
- When a connection is accepted, the address of the connection is fixed and we use *getsockname* to find the interface IP address
- You can bind to specific IP address instead of *INADDR_ANY*, only connections to this address are accepted
- Can generate EADDRINUSE error

8

bind Function 2/2

Client (see *daytimetcpcli3.c* in *intro* chap)

- Normally do not bind to any specific port or address
- As part of **connect** → **bind** is implicitly called
- Any ephemeral port and interface IP address is filled based on the routing table
- Use **getsockname** to find out the port and address

```
struct sockaddr_in servaddr, cliaddr;
len = sizeof(cliaddr);
Getsockname(sockfd, (SA *) &cliaddr, &len);
printf("local addr: %s\n", sock_ntop((SA *) &cliaddr, sizeof(cliaddr)));
```

9

listen Function 1/4

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

```
int listen (int sockfd, int backlog)
```

```
//returns 0 if OK, -1 on error
```

- When a socket created → assumed active socket
 - A client socket that will issue a **connect**
- **listen** converts an unconnected socket into a passive socket
- **backlog** specifies maximum number of connections the kernel should queue for this socket
- If the queues are full when client SYN arrives, TCP server ignore the SYN, it does not send RST.
- Kernel maintains 2 queues
 - Incomplete connection queue (only SYN received from client)
 - Completed connection queue (three-way handshake done) ¹⁰

listen Function 2/4

Figure 4.6 depicts these two queues for a given listening socket.

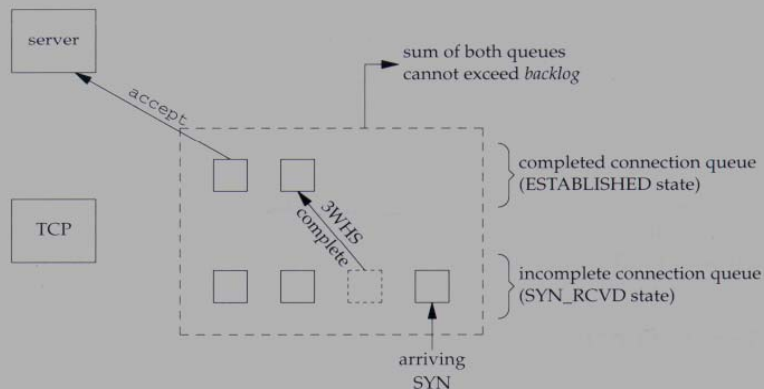


Figure 4.6 The two queues maintained by TCP for a listening socket.

11

listen Function 3/4

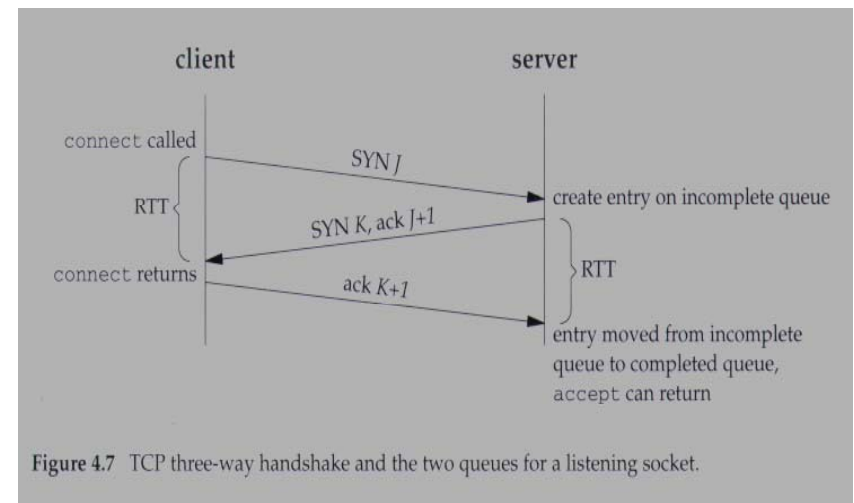


Figure 4.7 TCP three-way handshake and the two queues for a listening socket.

12

listen Function 4/4

- Berkeley-derived implementations add a fudge-factor to the backlog (multiplied by 1.5 → backlog of 5 allows up to 8 queued entries). *See figure 4.10*
- A backlog of 0 is not recommended (different implementations)
- Specifying a backlog inside source code is a problem! (growing number of connections to handle)
 - Specify a value larger than supported by kernel → kernel truncates value to maximum value that it supports
 - Textbook uses an environment variable for backlog (see `lib/wrapsock.c`)
- If queues are full when client SYN arrives
 - Ignore arriving SYN but do not send a RST (Why?)
- Data that arrives after 3WHS, but before a call to `accept` should be queued by TCP server

13

accept Function

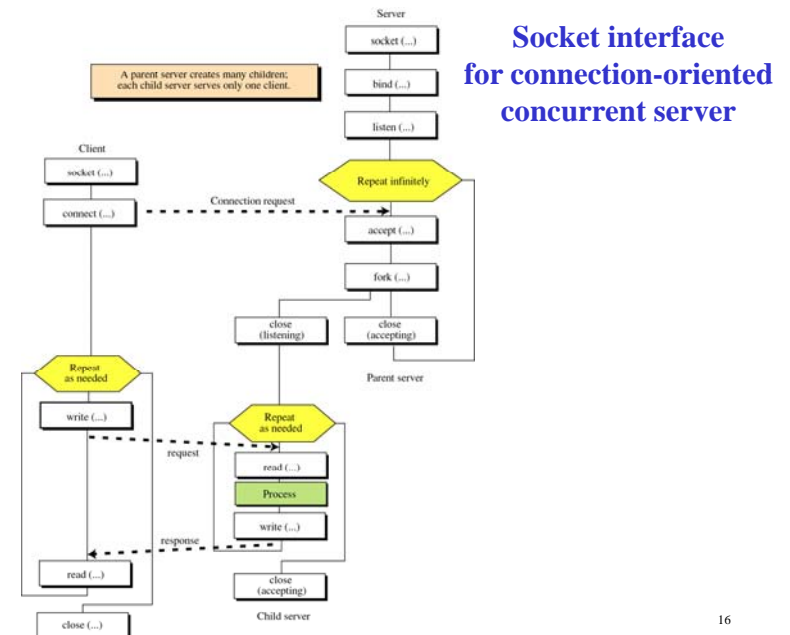
- ```
#include <sys/socket.h>
int accept (int sockfd, struct sockaddr * cliaddr, socklen_t * addrlen)
//returns non-negative descriptor if OK, -1 on error
```
- `cliaddr` and `addrlen` used to return protocol address of connected peer process
  - Set to `null` if not interested in identifying client
  - `addrlen` is a value-result argument
  - Difference between *listening socket* and *connected socket*

14

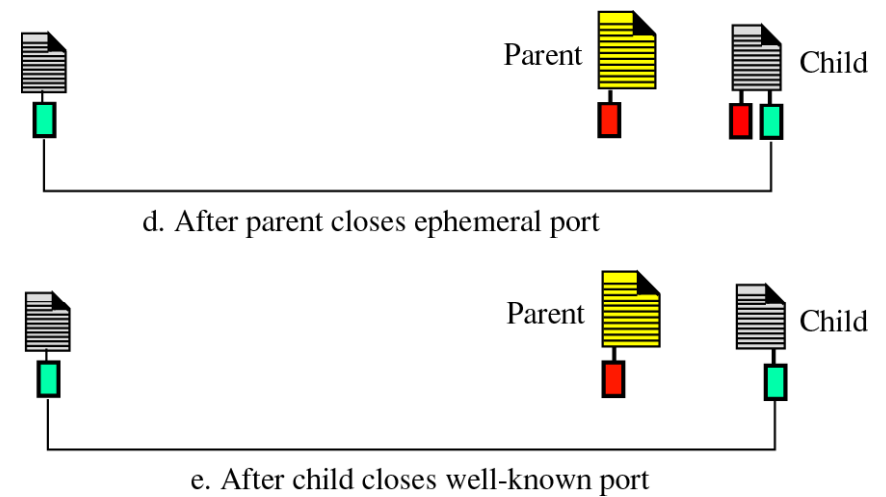
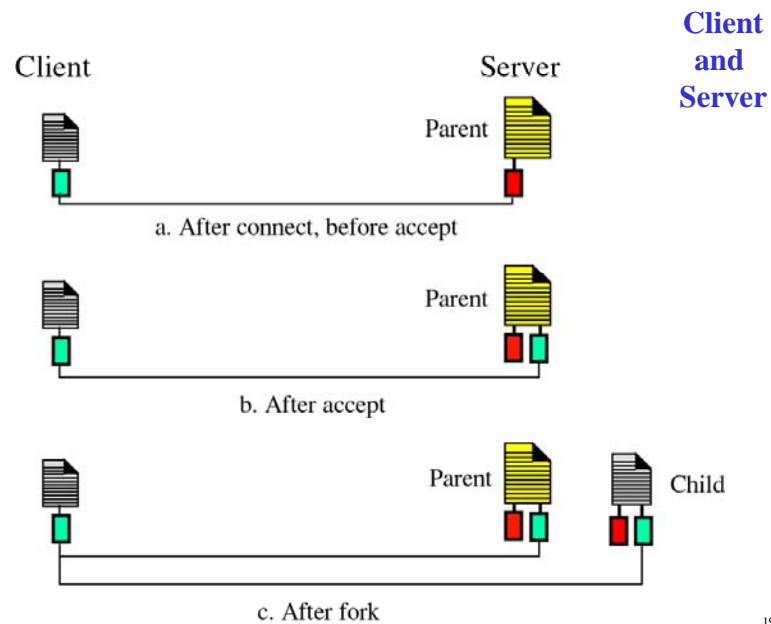
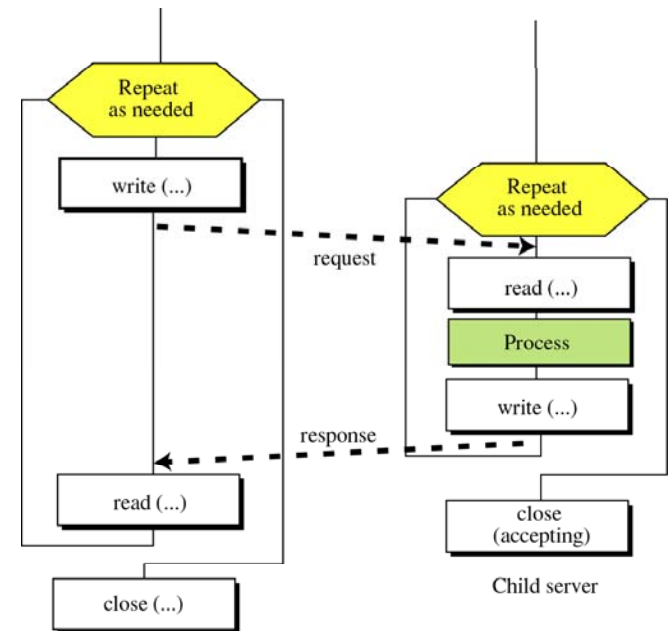
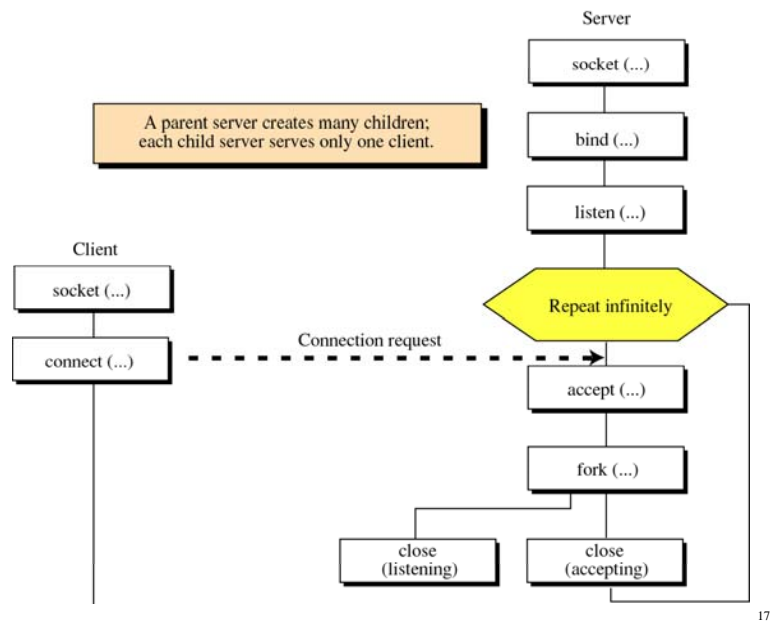
## Server Concurrency

- Servers use concurrency to achieve functionality and performance
- Concurrency is inherent in the server
  - must be explicitly considered in server design
- Exact design and mechanisms depend on support provided by the underlying operating system
- Achieved through
  - Concurrent processes
  - Concurrent threads (maybe cover later)

15



16





## fork Function

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

```
pid_t fork (void)
```

```
//returns 0 in child, process ID of child in parent, -1 on error
```

- A child has only 1 parent, can obtain parent ID by calling `getppid`
- Parent can not obtain IDs of its children unless keep track from return of `fork`
- All descriptors open in parent before call to `fork` are shared with child after `fork` returns (connected socket shared between parent and child)
- Use `fork` to
  - Process makes a copy of itself (typical for network servers)
  - Process wants to execute another program (call `fork` then `exec`)

21

## Concurrent servers 1/3

```
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 */
}
```

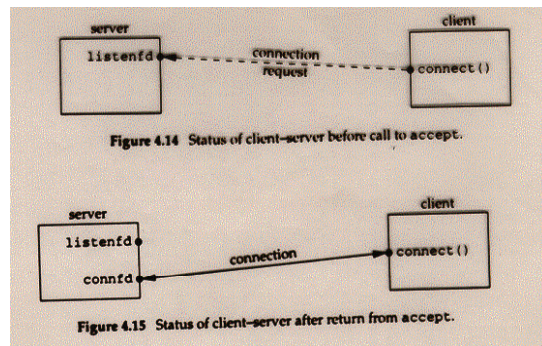
Figure 4.13 Outline for typical concurrent server.

22

## Concurrent Servers 2/3

Why close of `connfd` by parent does not terminate connection with the client?

- Every file or socket has a reference count
- Reference count: A count of the number of descriptors that are currently open that refer to this file or socket



23

## Concurrent Servers 3/3

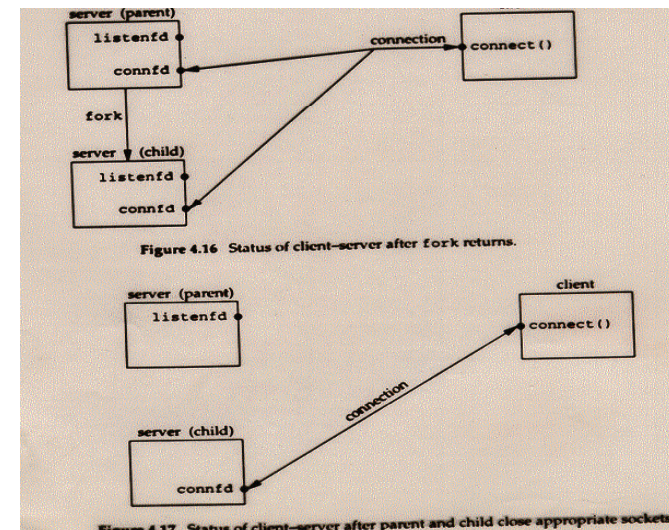


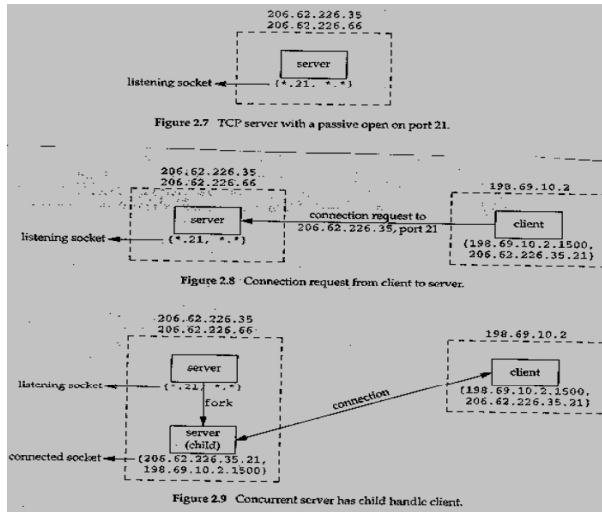
Figure 4.17 Status of client-server after parent and child close appropriate sockets.

24

## Port Numbers and Concurrent Servers 1/2

- Main server loop spawns a child to handle each new connection

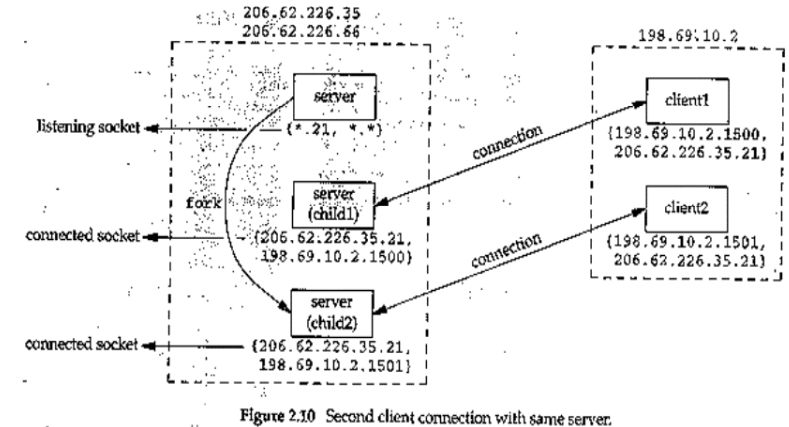
- What happens if child continues to use the well-known port number while serving a long request?



25

## Port Numbers and Concurrent Servers 2/2

- Another client process on client host requests a connection with the same server



26

## close Function

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

- Will try to send any data that is already queued to be sent to the other side, then normal TCP connection termination sequence takes place (send FIN)
- Can use an option to discard unsent data (later)

27

## 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)
```

- getsockname returns local protocol address associated with a socket
- getpeername returns the foreign protocol address associated with a socket
- getsockname will return local IP/Port if unknown (TCP client calling connect without a bind, calling a bind with port 0, after accept to know the connection local IP address, but use connected socket)

28

- Project Phase 3 is available at:

[http://www.eis.hu.edu.jo/ACUploads/10799/Project\\_P3\\_F2011.pdf](http://www.eis.hu.edu.jo/ACUploads/10799/Project_P3_F2011.pdf)