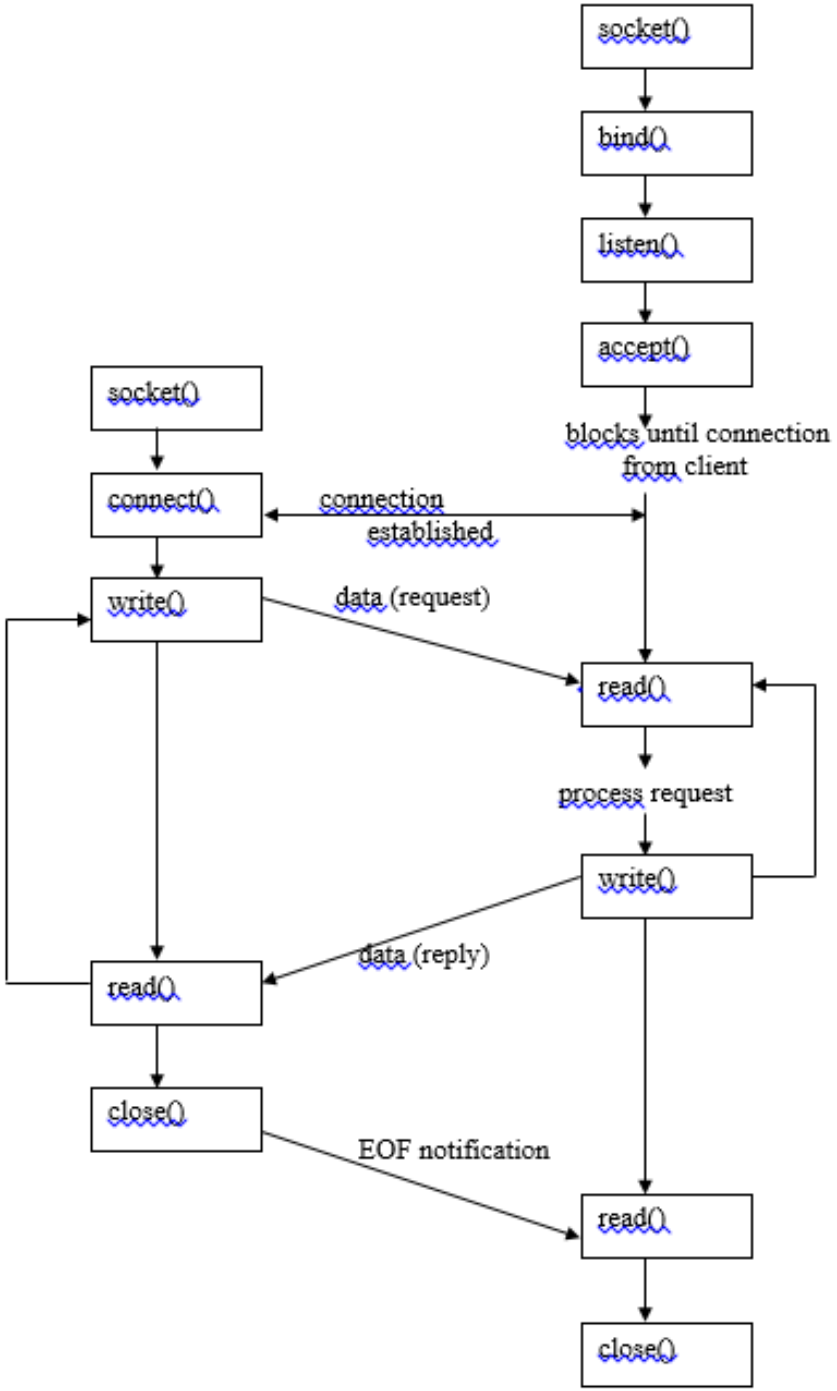
Network Programming

Server Algorithms and Issues

Note: **This class lecture will be recorded!**
If you do not consent to this recording, please do not ask questions via your video, audio or public chat; send your question to the instructor using the private chat.

Lisa Frye, Instructor
frye@Kutztown.edu
Kutztown University

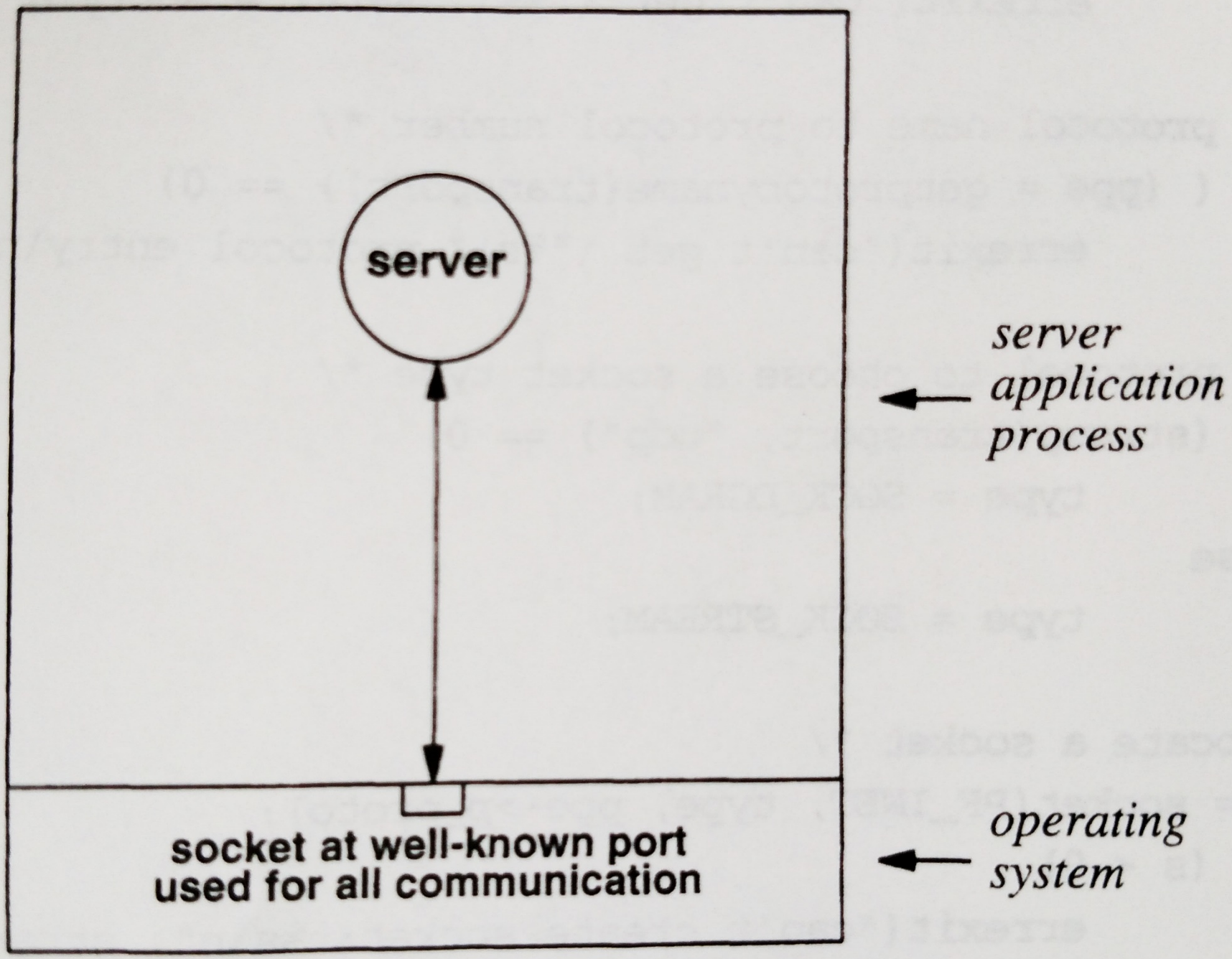

socket()
bind()
listen()
accept()
blocks until connection from client
socket()
connect()
connection established
write()
data (request)
read()
process request
write()
data (reply)
read()
close()
EOF notification
read()
close()

Server Types

- Iterative server
  - What are some problems with this type of server?
- Concurrent server
- Connection-oriented vs. Connectionless
- Stateful vs. Stateless

Basic Types of Servers

Iterative Connectionless	Iterative Connection-oriented
Concurrent Connectionless	Concurrent Connection-Oriented


server
server application process
socket at well-known port used for all communication
operating system

Iterative, Connectionless Algorithm

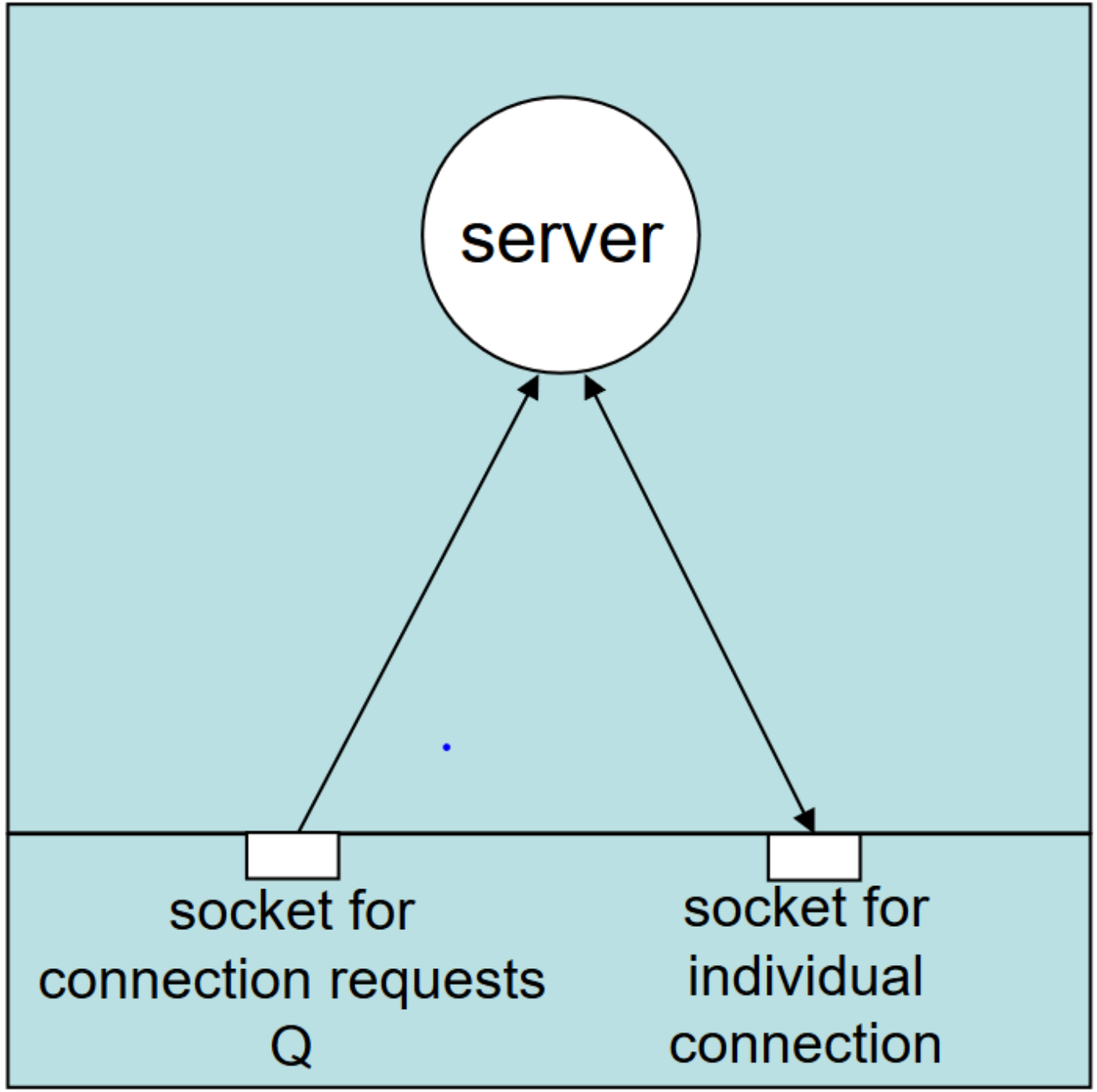
- Create a socket and bind to the well-known address for the service being offered
- Repeatedly read the next request from a client, formulate a response, and send a reply back to the client according to the application protocol

Discussion

- Conduct an experiment to determine what happens if N clients all send requests to *UDPtimed* simultaneously. Vary both N, the number of senders, and S, the size of the datagrams they send. Explain your experiment, including the values of the inputs used and why you chose those values. Explain why the server fails to respond to all requests.

server application thread

operating system



Iterative, Connection-oriented Algorithm

- Create a socket and bind to the well-known address for the service being offered
- Place the socket in passive mode, making it ready for use by a server
- Accept the next connection request from the socket, and obtain a new socket for the connection
- Repeatedly read a request from the client, formulate a response, and send a reply back to the client according to the application protocol
- When finished with a particular client, close the connection and return to step 3 to accept a new connection

Discussion

- Calculate how long an iterative server takes to transfer a 200 MB file if the internet has a throughput of 2.3 KBps

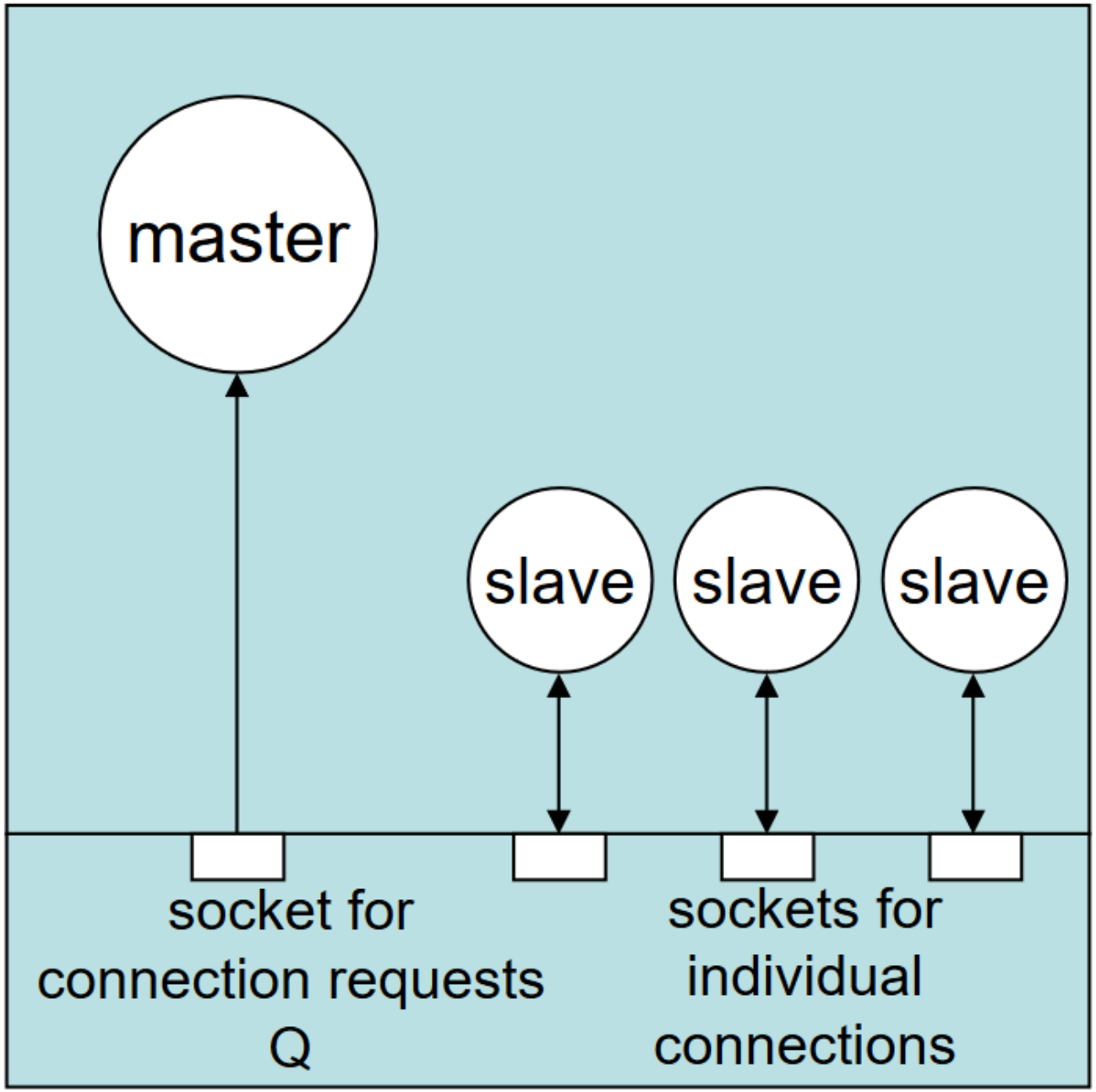
- If 20 clients each send 2 requests per second to an iterative server, what is the maximum time that the server can spend on each request?

Concurrent Servers

- Multiple threads or multiple processes
- ***Main thread*** *or* ***Parent process***
  - Begins execution
  - Listens on socket (passive socket)
  - New connection from client
    - Create ***sub-thread*** *or* ***child process*** *to handle connected client communication*

server application thread

operating system



Concurrent, Connectionless Algorithm

- Main Thread or Parent process
  - Create a socket and bind to the well-known address for the service being offered. Leave the socket unconnected.
  - Repeatedly call *recvfrom* to receive the next request from a client, and create a new sub-thread (possibly in a new process) to handle the response
- Sub-Thread or Child process
  - Begin with a specific request from the main thread as well as access to the socket
  - Form a reply according to the application protocol and send it back to the client using *sendto*
  - Exit after handling the one request

Concurrent, Connection-oriented Algorithm

- Main Thread or Parent process
  - Create a socket and bind to the well-known address for the service being offered. Leave the socket unconnected.
  - Place the socket in passive mode, making it ready for use by a server.
  - Repeatedly call *accept* to receive the next request from a client and create a new sub-thread or process to handle the request.
- Sub-thread or Child process
  - Begin with a connection passed from the main thread or parent process
  - Interact with the client using the connection: read request(s) and send back response(s)
  - Close the connection and exit. The sub-thread or process exits after handling all requests from one 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 ( ; ; )  {       // while (1)
      connfd = accept(listenfd, …);
      if  ((pid = fork()) == 0)  {
            close(listenfd);            // child closed listening socket (not if using threads)
            doit(connfd);               // process the request
            close(connfd);              // done with this client (close for processes and threads)
            exit(0);                    // child terminates
      }     // end if
      close(connfd);                    // parent closes connected socket (not if using threads)
}   // end for
```

Concurrent, Connection-oriented Server Example

- sockets/TCPecho

Example

- Sockets/TCPsockets/sum/client_bad.c

Client or Server Ends

- How does the client or server know when the client disconnects?

- Possible solutions
  - Add a keepalive message to the application protocol
  - Explicit timer
  - Manipulate TCP/IP keepalive packet settings