# `select()`

Network Programming

# Motivation

- What happens when you call **`read()`** or **`fgets()`**?
- What if no data is available to satisfy those requests?
- We need a mechanism to inform the kernel we're interested in a changes to specific file descriptors (FDs)

# Compare I/O Models

- Blocking I/O
- Non-blocking I/O
- I/O multiplexing
- Signal-driven I/O
- Asynchronous I/O

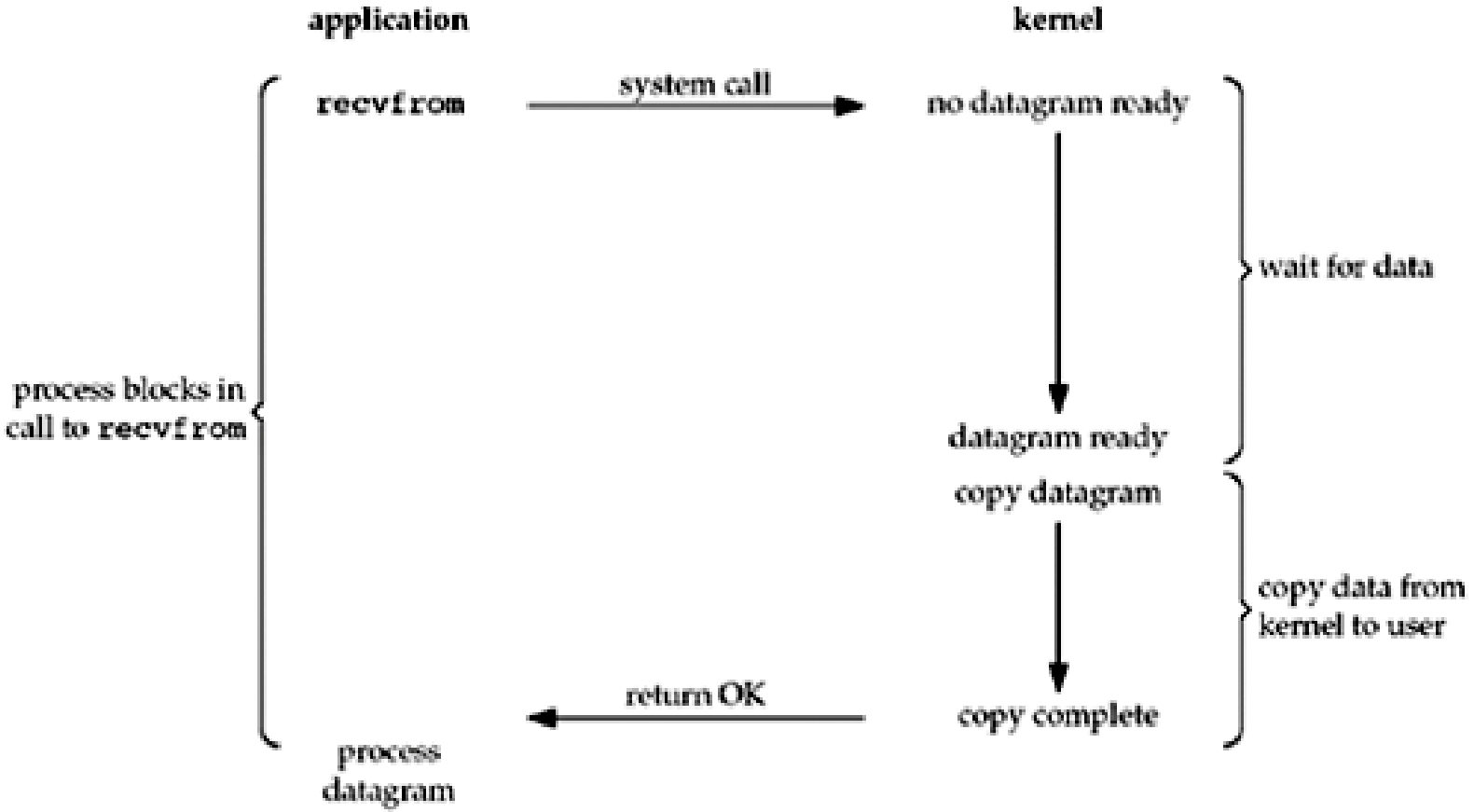
# Blocking I/O Model

- By default, all sockets are blocking
- Request data, userspace->kernel-> userspace
- Common error while waiting? Being interrupted by kernel (EINTR)
- “Slow system calls” such as **`accept()`**

# Slow accept()

```
for ( ; ; ) {
    clilen = addrlen;
    if ( (connfd = accept(listenfd, cliaddr, &clilen)) < 0) {
        if (errno == EINTR)
            continue;       /* back to for() */
        else
            err_sys("accept error");
    }
```

# Blocking I/O Model

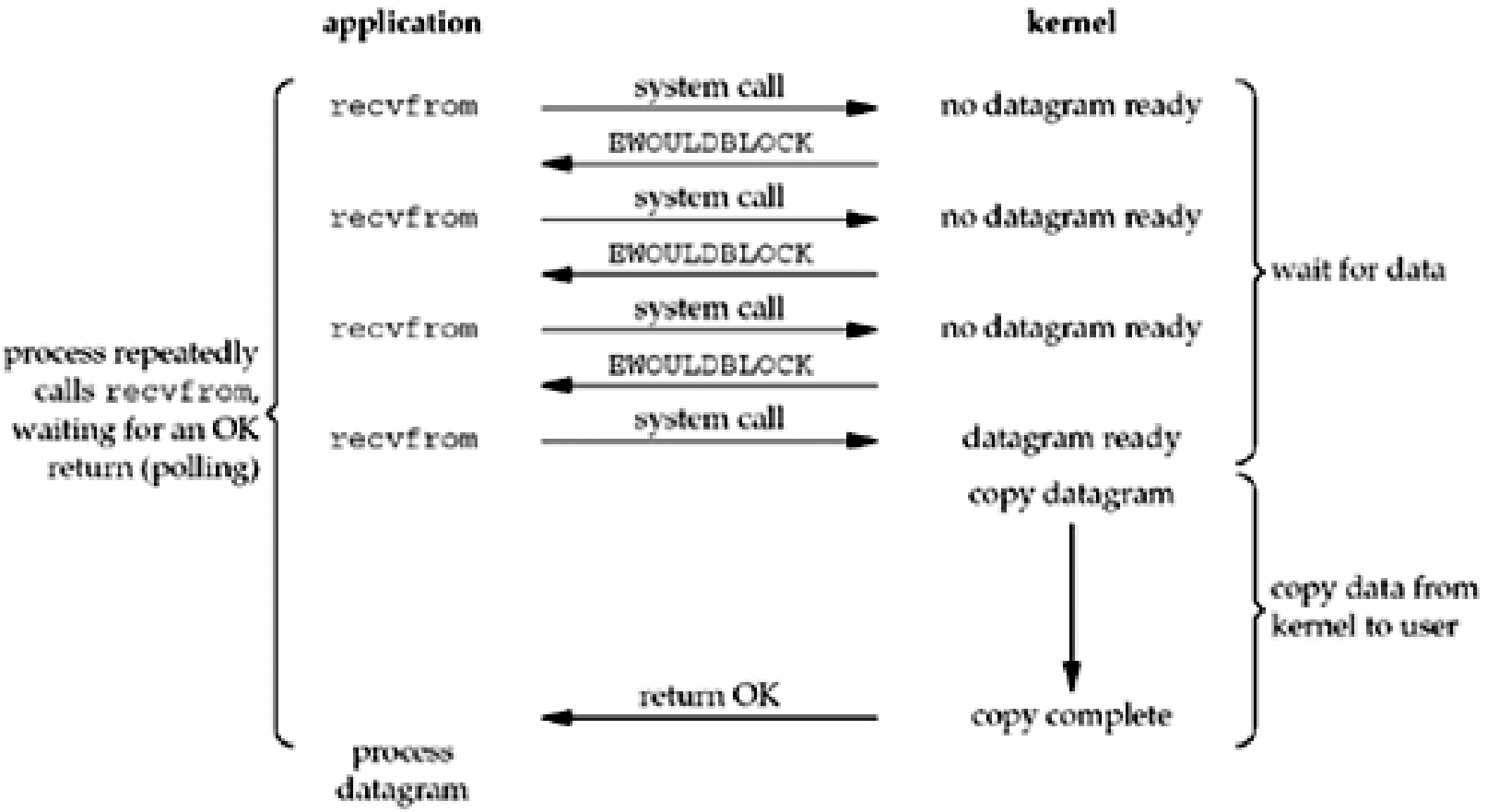



Section 6.2 of UNIX Network Programming, Vol 1, 3rd ed.

# Non-blocking I/O Model

- Set socket to non-blocking
- Return -1, **`errno`** = **`EWOULDBLOCK`** instead of blocking!
- Often referred to as *polling* and is fairly wasteful of resources

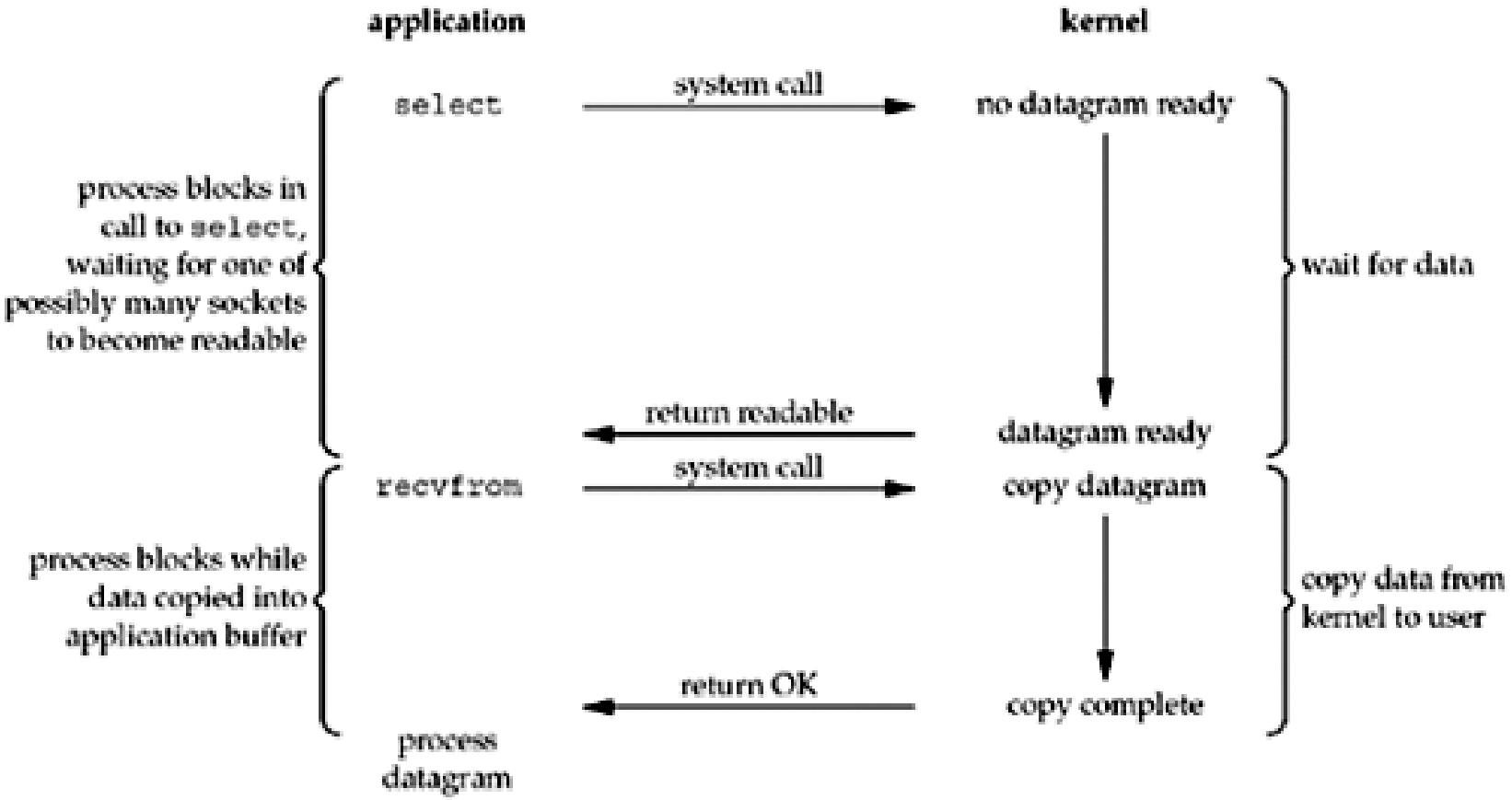
# Non-blocking I/O Model



# I/O Multiplexing Model

- Instead of **`read()`** blocking, **`select()`** will block
- Upon return, socket may be readable (or not, may return errors)
- For a single item, not much benefit vs. blocking I/O model
- With multiple items, we can wait on all of them!

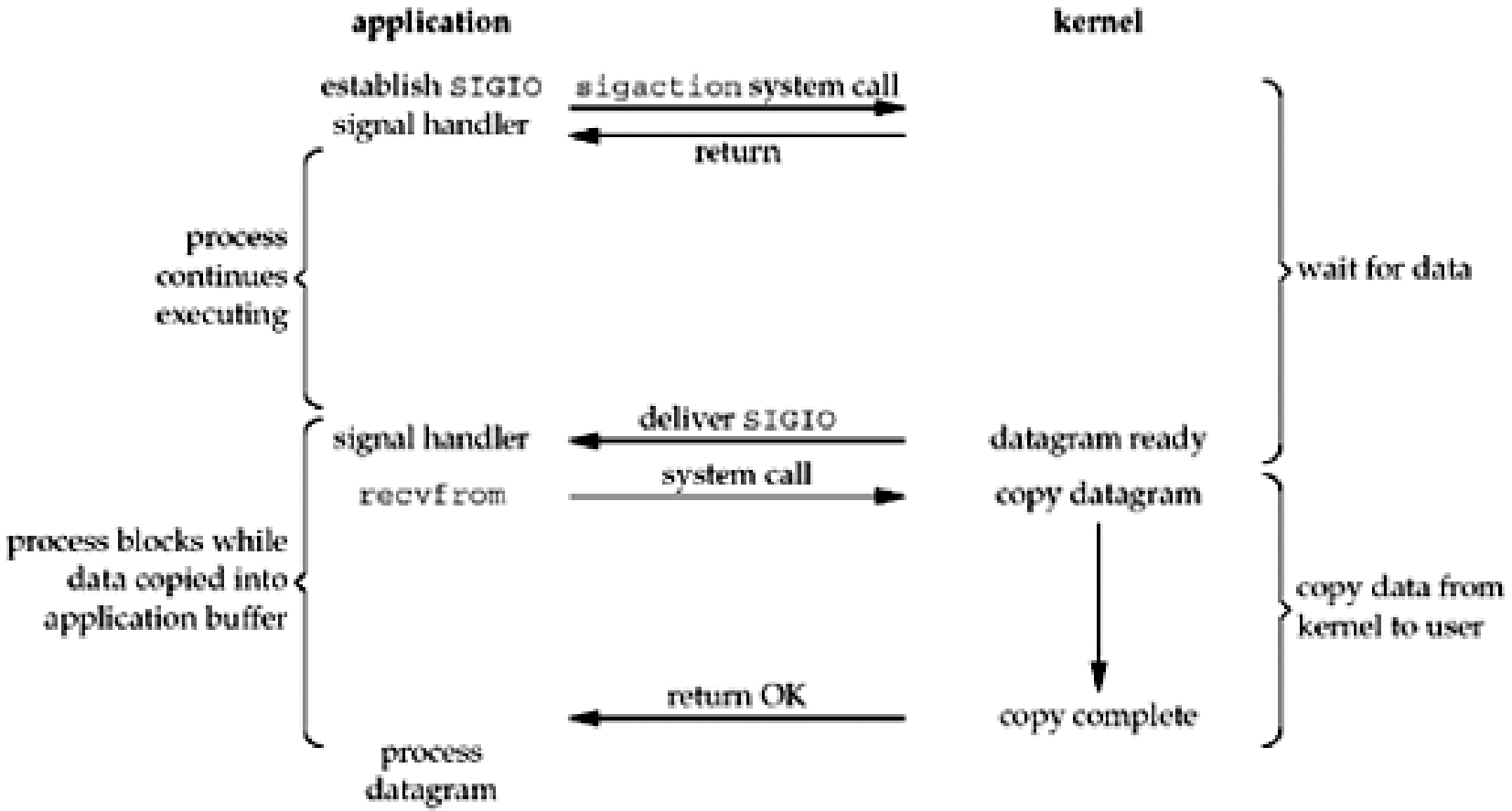
# I/O Multiplexing Model



# Signal-Driven I/O Model

- Requires signal handler for SIGIO
- Set socket owner
- Toggle signal-driven I/O on socket
- Probably won't be discussed again in this class

# Signal-Driven I/O Model



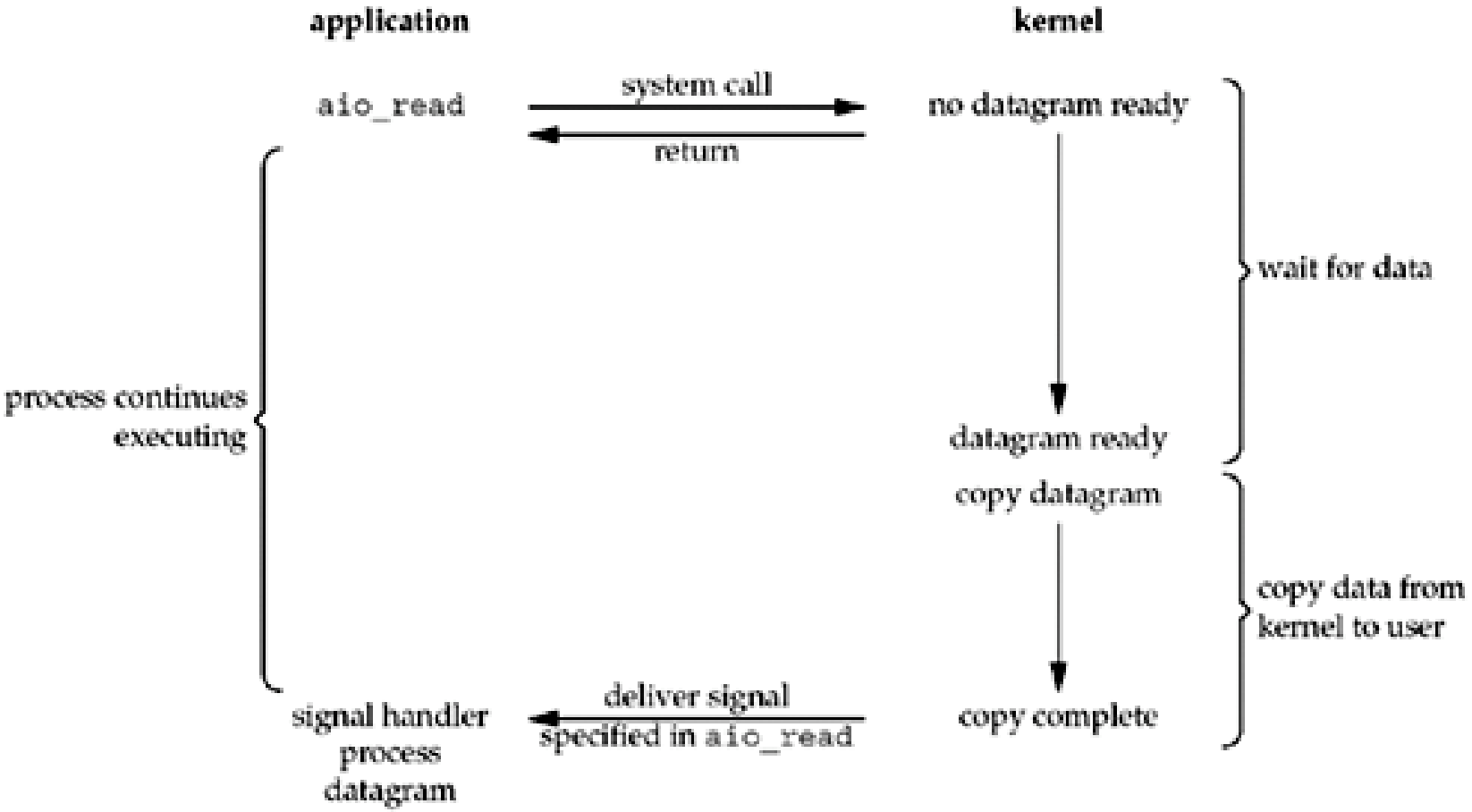
# Asynchronous I/O Model

- Similar to signal-driven I/O model
- Signal-driven informs us when I/O operation can be initiated
- Asynchronous informs us when I/O operation completes
- “Fuzzy” support on many platforms

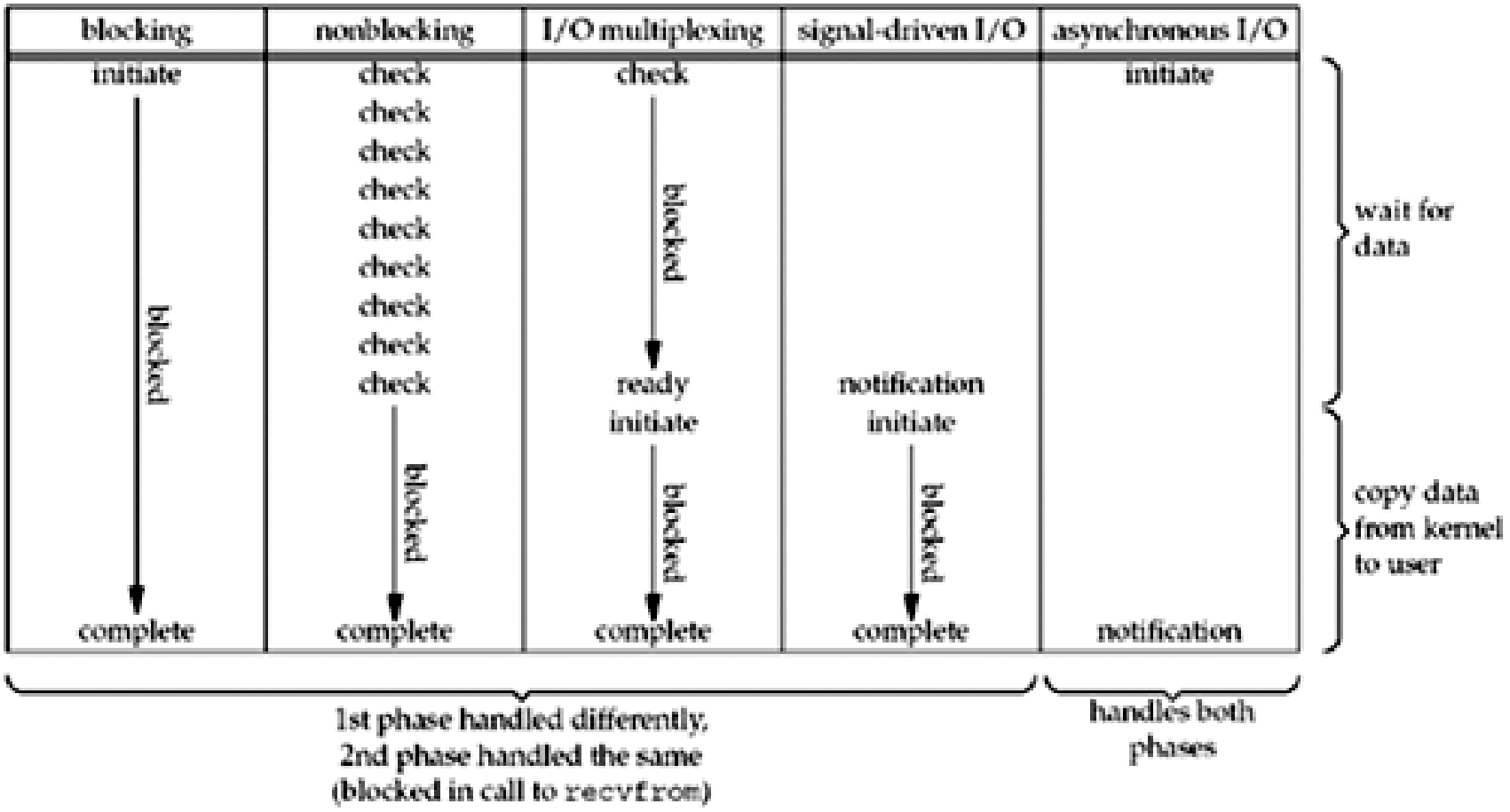
# Asynchronous I/O Model

- Optional reading:
  - http://lse.sourceforge.net/io/aio.html (old!)
  - https://lwn.net/Articles/724198/
  - https://blog.cloudflare.com/io_submit-the-epoll-alternative-youve-never-heard-about/

# Asynchronous I/O Model



# Comparison


blocking
nonblocking
I/O multiplexing
signal-driven I/O
asynchronous I/O
initiate
check
check
check
check
check
check
check
check
check
check
blocked
blocked
blocked
ready
initiate
blocked
notification
initiate
blocked
initiate
complete
complete
complete
complete
notification
wait for data
copy data from kernel to user
1st phase handled differently, 2nd phase handled the same (blocked in call to recvfrom)
handles both phases

# select() system call

- Enables handling reading, writing, exceptional cases for file descriptors
- Informs kernel of interest in those specific file descriptors

```
int select(int nfds, fd_set
   *readfds, fd_set *writefds,
   fd_set *errorfds, struct
   timeval *timeout);
```

# Select Is Fundamentally Broken?

- https://idea.popcount.org/2017-01-06-select-is-fundamentally-broken/

- Mentions the “Thundering Herd Problem”: https://en.wikipedia.org/wiki/Thundering_herd_problem

# Clearing / Toggling FDS

- `void FD_SET(int fd, fd_set *s)`
- `void FD_CLR(int fd, fd_set *s)`
- `int FD_ISSET(int fd, fd_set *s)`
- `void FD_ZERO(fd_set *s)`

# Checking `select()` results

- Make sure the return value is positive!
  - If not, check **`errno`** variable
- Zero if **`select()`** timed out
- Otherwise, return number of file descriptor bits that are set
- Modifies the readfdset, writefdset, and errorfdset!

# When Can We Read?

- Read buffer has data in it
- Read half of connection is closed
  - read() will return 0
- Listening socket can **`accept()`**
  - Timing conditions exist!!!
- Socket error pending, **`read()`** will return -1, check **`errno`**

# When Can We Write?

- Send buffer has available space
- Write half of connection closed
  - **`write()`** will return -1, **`errno: SIGPIPE`**
- Non-blocking **`connect()`** has completed (or failed…)
- Socket error pending, **`write()`** will return -1, check **`errno`**

# tcpservselect01.c

- See file (in tcpcliserv/)

# strcliselect01.c

- See file (in select/)

# Buffered Data

- If we ran this as a batch (background) task: ./cli.out < ten_lines.txt &
- Might be a lot of data still travelling between the client and server, even once we got EOF from our input.
- select() only checks if we can read(), doesn't look at stdio **buffers**, lots of data available at once
  - fgets() gets one line and then returns
  - Dangerous to mix select() and stdio (see 6.5)

# `shutdown()` system call

- **`close()`** decrements FD reference count, close when reaches zero
- **`shutdown()`** starts the 4-way termination process
- **`close()`** closes both directions, **`shutdown()`** allows one at a time

# strcliselect02.c

- See file (in select/)

# Bonjour / Zeroconf

Network Programming

# Simplicity

- Easily browse for available services
  - Avoid situations where you can't use a printer in the same room as you
- Service Discovery
  - Browse for services, not hardware
- Names and Addresses
  - Grab IP address, name via [m]DNS

# IP Address First

- Manual, DHCP, whatever
  - Sometimes use private addresses
- RFC 3927 defines the *link-local address* range 169.254.0.0 to 169.254.255.255
- Pick a target IP, ARP for owner
  - Hopefully get no response
  - Claim that IP

# .local namespace

- Sent to 224.0.0.251, IP reserved for mDNS
- mDNS queries utilize "Known Answer Lists" and exponential back-off to reduce network load
- Zeroconf goes to great lengths to not overburden the network

# Hostname Uniqueness

- Repeated queries to establish uniqueness
- Identical to DNS A records
  - A: hostname -> IPv4 adddress
- Announce ownership via response; updates all neighbors of new owner

# Browsing

- Services must indicate “how” as well as “what”
  - Ex. _ipp: what (printing) how (Internet Printing Protocol)
  - http://www.dns-sd.org/ServiceTypes.html
- Does http://www.example.com point to a host or a service???
- RFC 2782 (DNS SRV) (created by Microsoft

# DNS Resource Records

- A: hostname -> IPv4 address
- PTR: IP address -> hostname
- SRV: Service Records
  - Include hostname/port number for available services
  - This can answer the question from the previous slide!

# Service Discovery on DNS

- DNS already provides much of the infrastructure necessary for Zeroconf:
  - Central aggregation server
  - Service registration
  - Query protocol

- By piggy-backing off DNS, deployment issues solved

# DNS TXT Records

- Useful to store additional data
  - Key/value pairs
  - Format: 1 length byte followed by 0-255 bytes of text
- RFC 1035 *requires* at least one string
  - 0 length byte followed by empty string not uncommon
- Good idea to support txtvers=*xxx* and ignore unknown keys

# Non-Local Service Discovery

- Zeroconf works well on small, local networks
- Same ideas would *not scale well* on global internet
  - Far too much traffic would be sent around!
- Unicast DNS instead! Solved!
  - Not covered in this course, see Ch. 5 Zeroconf book

# dns-sd

- -B: browse for available services
  - Ex: dns-sd -B _daap._tcp
- -R: register services
  - dns-sd -R ”J Music" _daap._tcp "" 9904
- -L: resolve services
  - dns-sd -L ”J Music" _daap._tcp

# See Also

- http://www.dns-sd.org/servicetypes.html
- http://opac.lib.rpi.edu/record=b3909317
  (RPI Library link to Zerconf book)