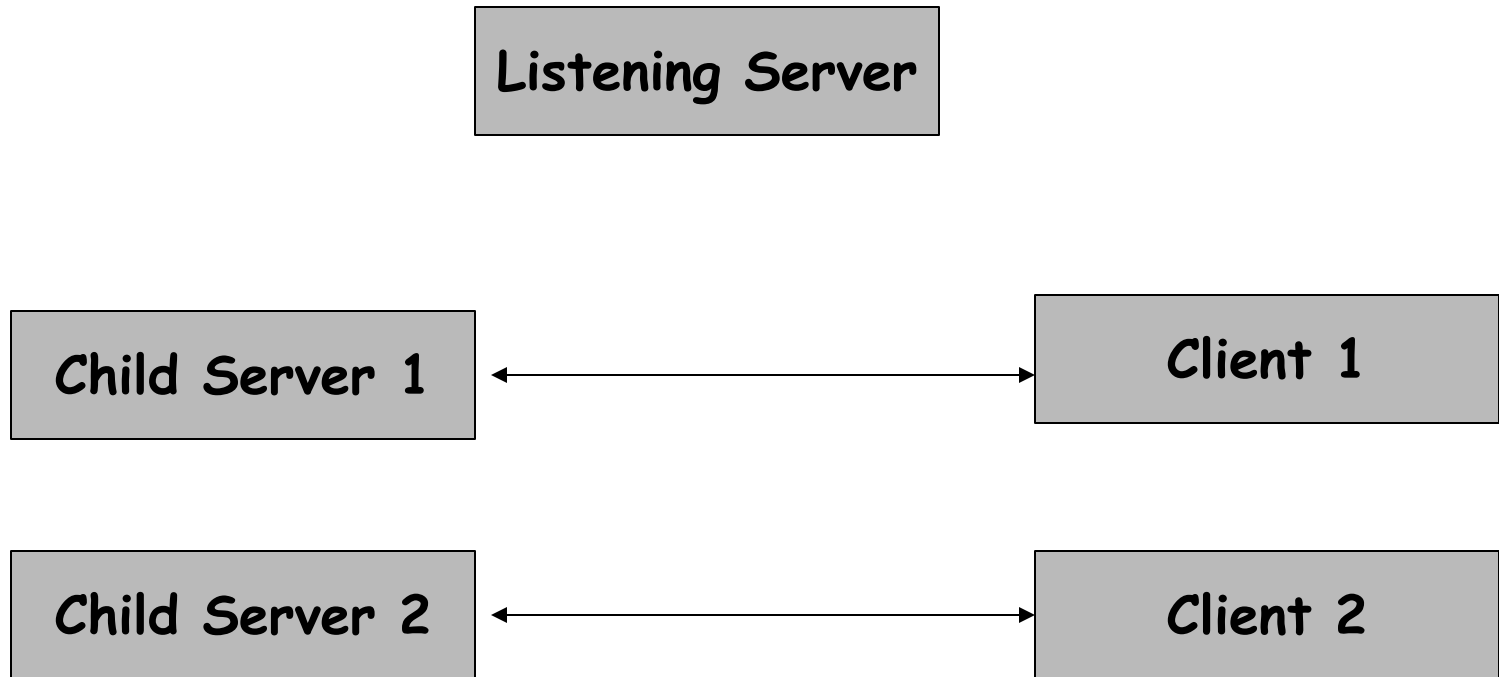# I/O Models

Kartik Gopalan

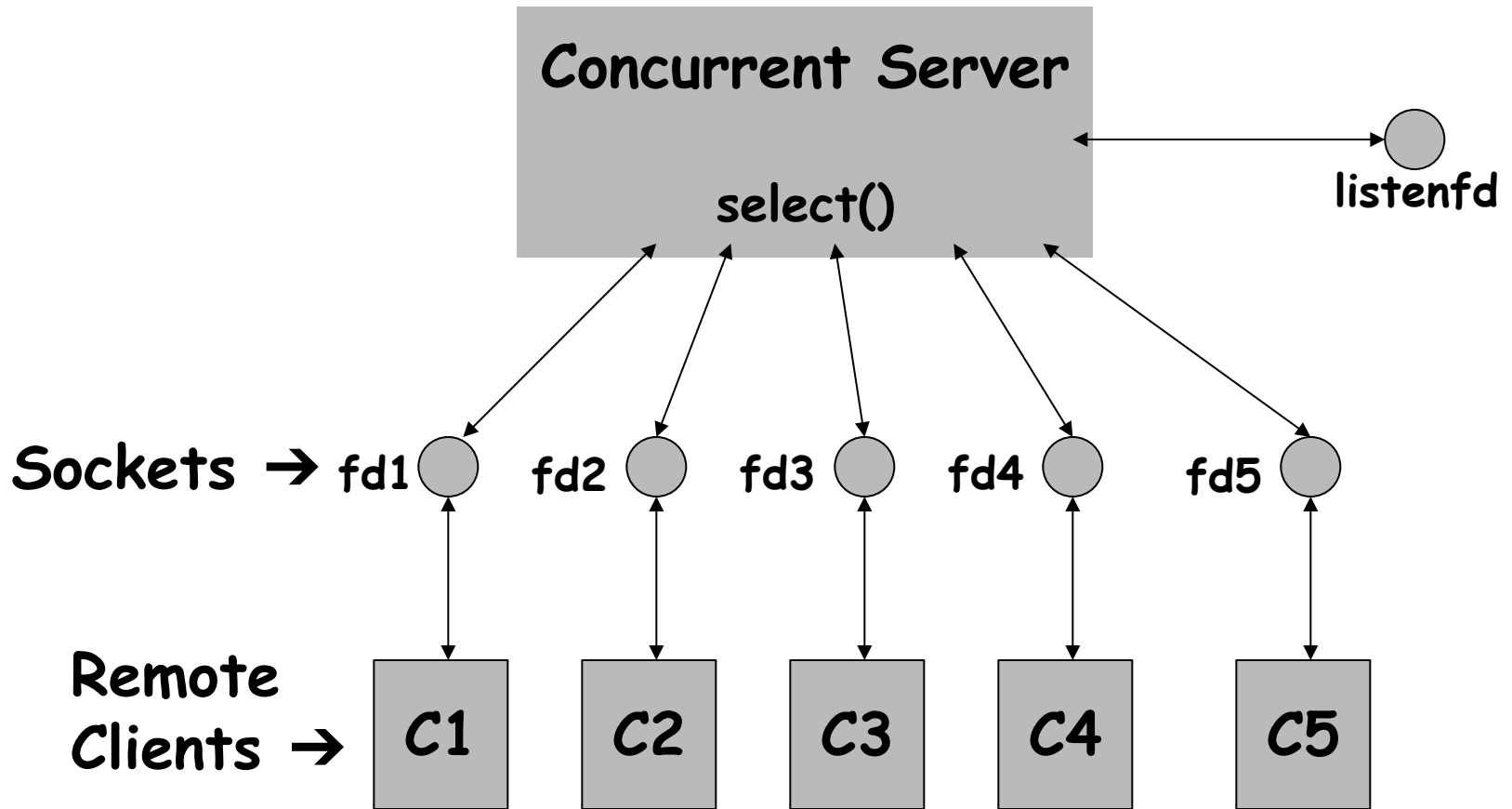# Types of Concurrency

- True Concurrency (multiple processes or threads)
  - Multi-processor machines
    - Child processes/threads execute in parallel.

  - Multi-process (forking) servers
    - If one child server blocks, another executes

  - Multi-threaded servers
    - If one thread blocks, another executes.
    - Only if threads supported by kernel

- Apparent Concurrency
  - Single process does multiplexing among multiple clients.
    - E.g. I/O multiplexing with select().
  - Multi-threaded Server (again!)
    - If threads implemented at user level.

True Concurrency:
Forking Concurrent Servers

Listening Server

Child Server 1 ⟷ Client 1

Child Server 2 ⟷ Client 2

Apparent Concurrency
Non-Forking Concurrent Server

Concurrent Server

select()

listenfd

Sockets → fd1 fd2 fd3 fd4 fd5

Remote
Clients → C1 C2 C3 C4 C5
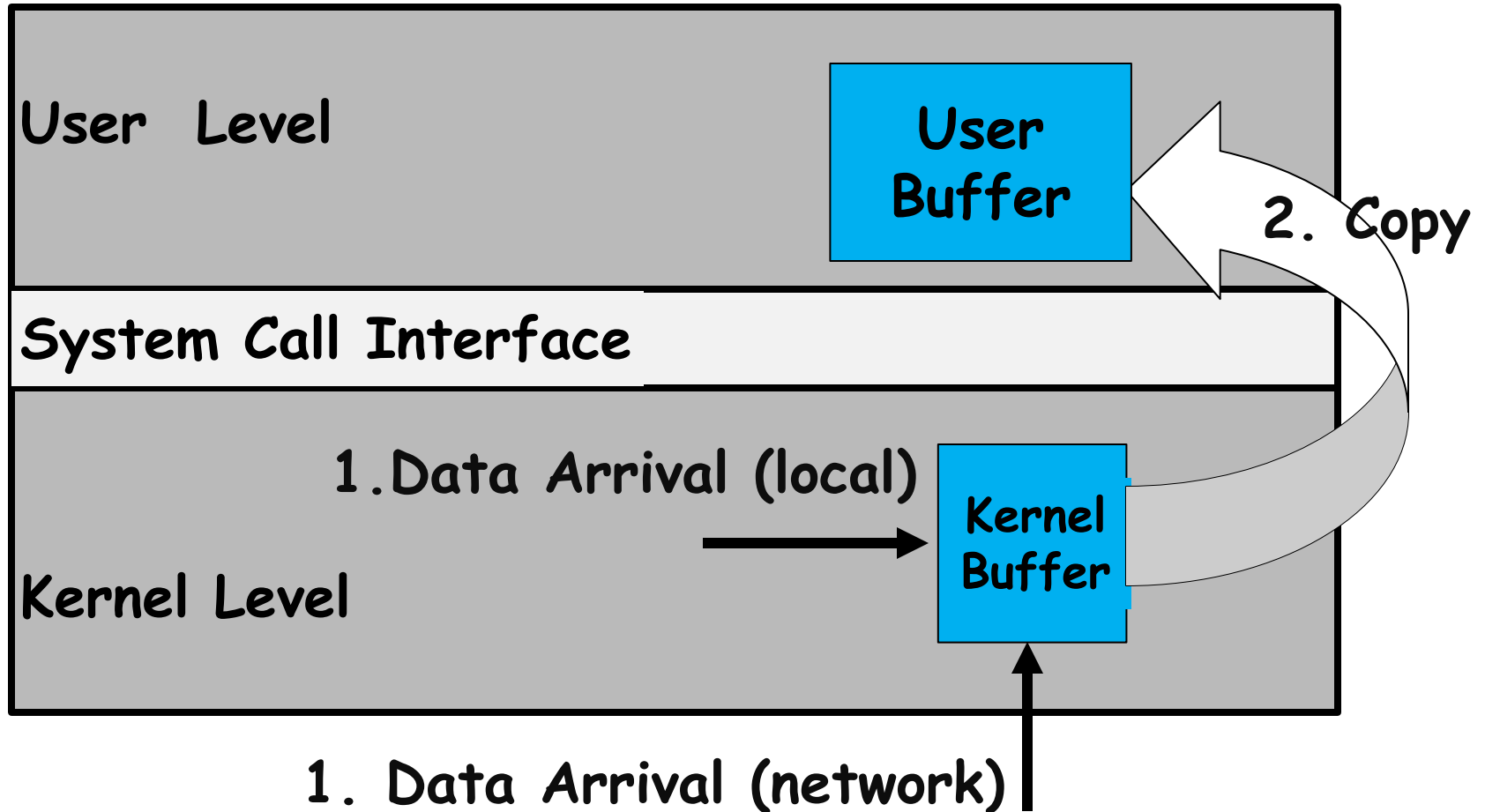
I/O Models
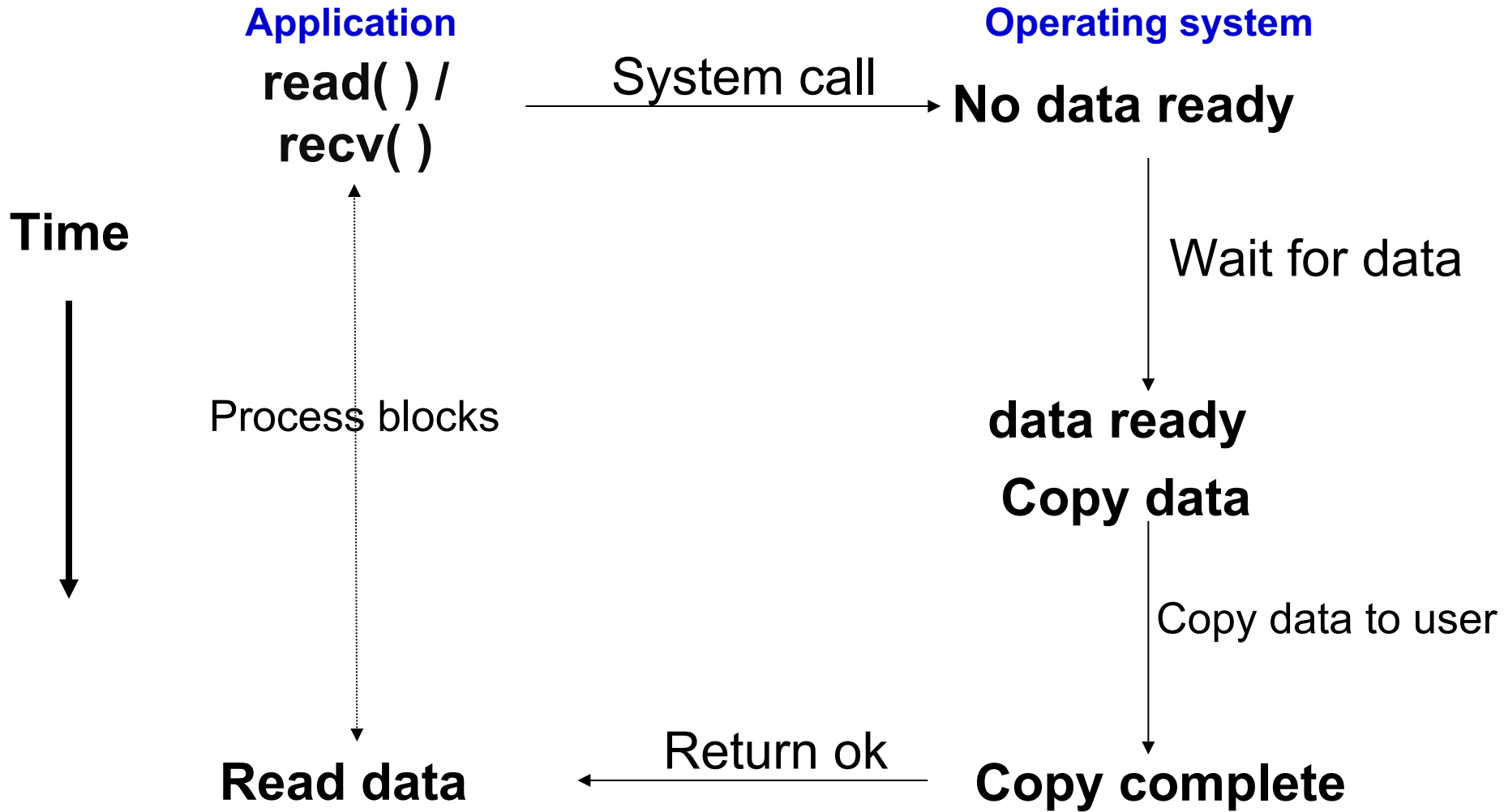
- Blocking I/O
- Non-blocking I/O
- I/O multiplexing – select()
- Signal driven I/O
- Asynchronous I/O
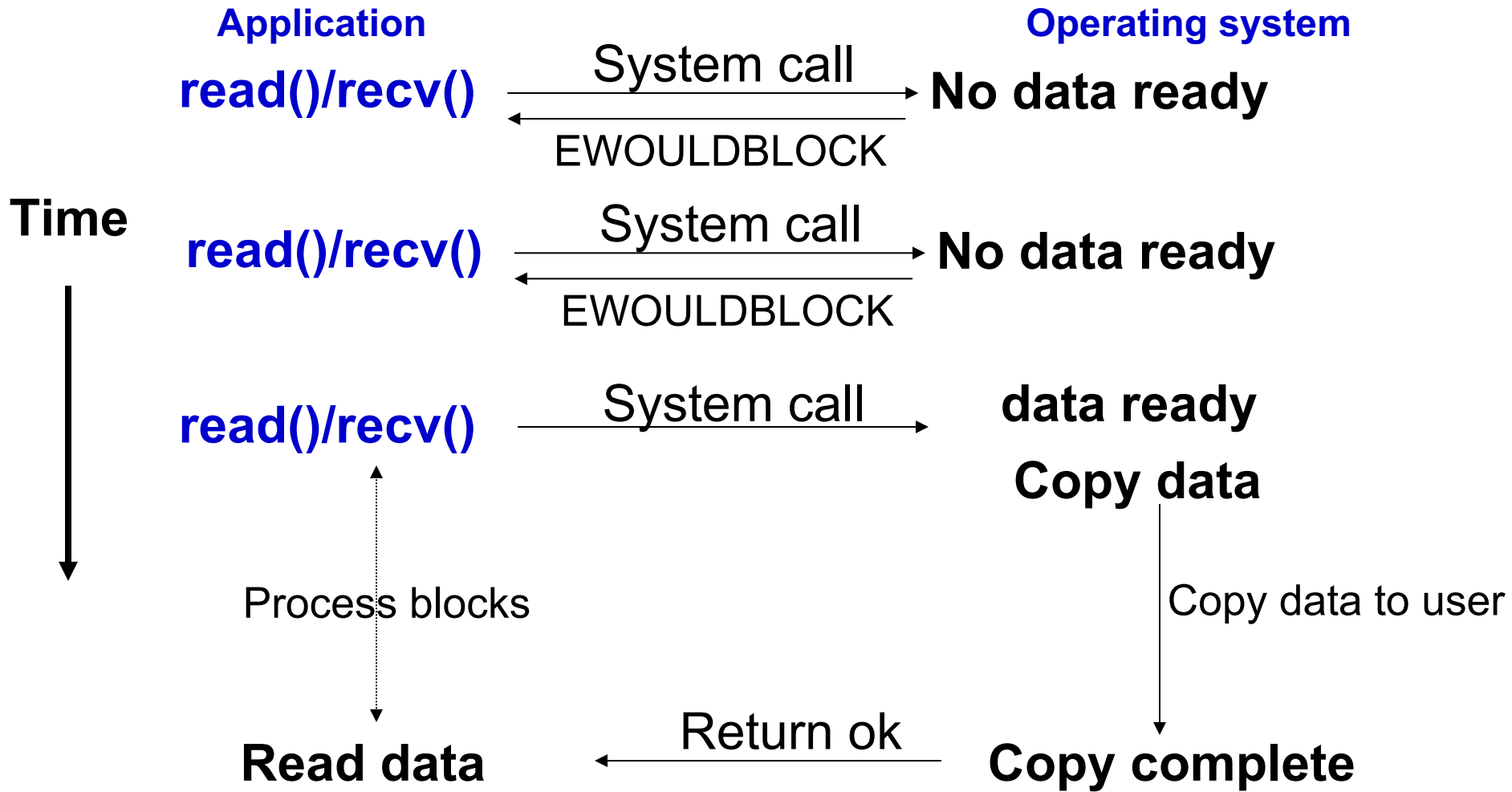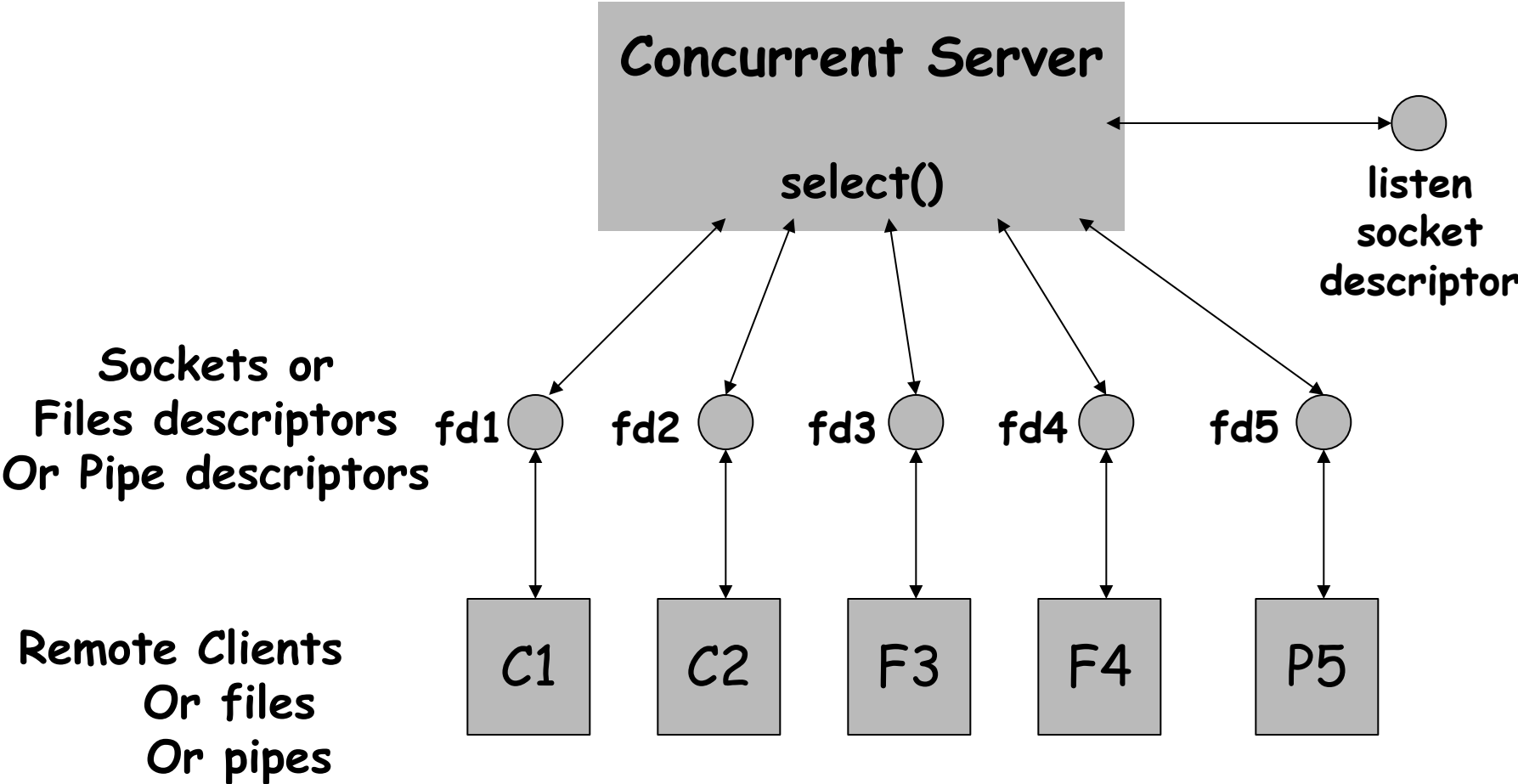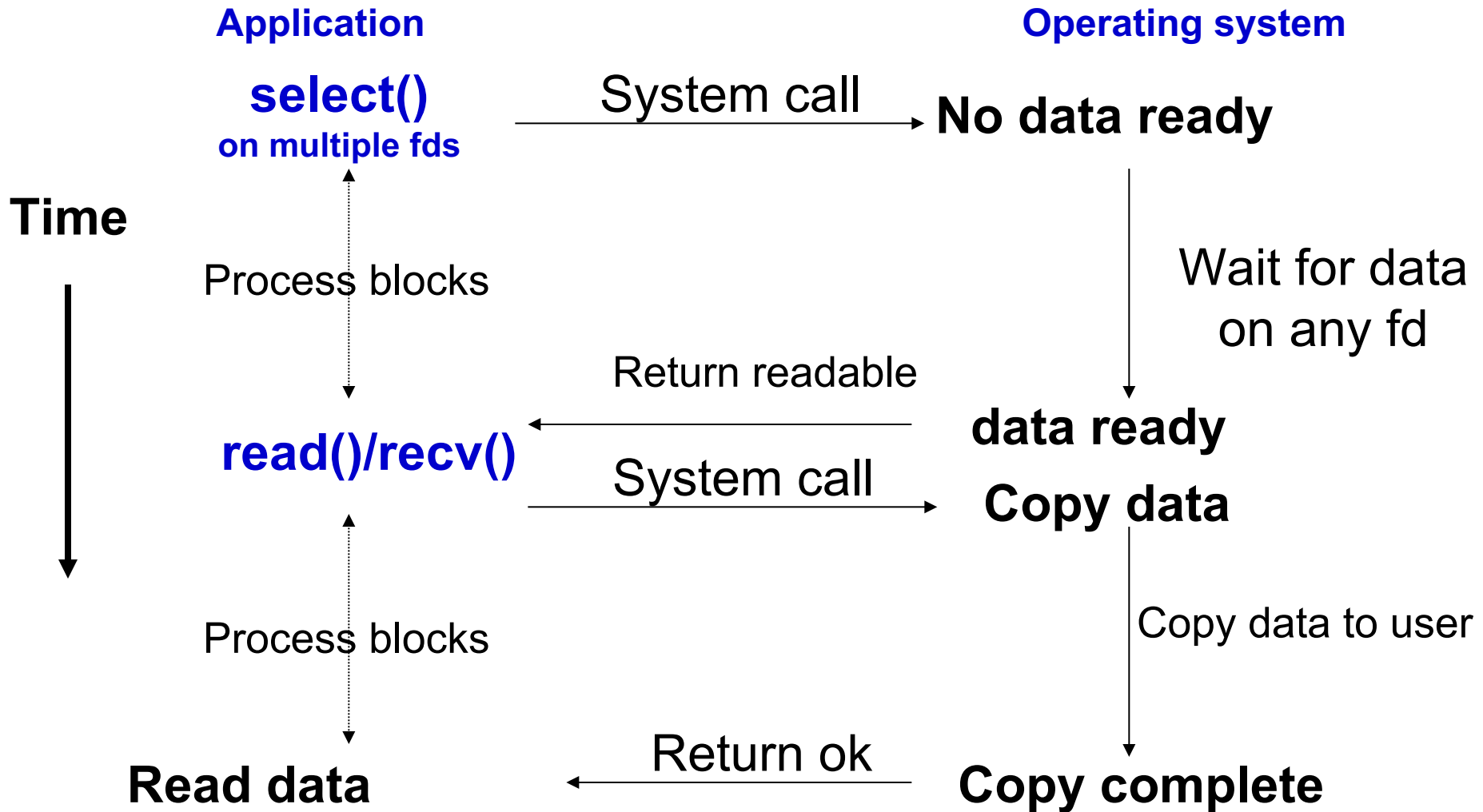
Two steps in data reception

User Level

User Buffer

2. Copy

System Call Interface

1.Data Arrival (local)

Kernel Buffer

Kernel Level

1. Data Arrival (network)

Overheads: Context switching, Data copying

Blocking I/O

Application

read( ) /
recv( )

System call →

Operating system

No data ready

Time

Process blocks

Wait for data

data ready

Copy data

Copy data to user

Read data   ← Return ok   **Copy complete**

Non-Blocking I/O (polling)

Application   **Operating system**

read()/recv() — System call → **No data ready**

EWOULDBLOCK

Time

read()/recv() — System call → **No data ready**

EWOULDBLOCK

read()/recv() — System call → **data ready**

Copy data

Process blocks   Copy data to user

Read data ← Return ok — **Copy complete**

8

I/O Multiplexing

Concurrent Server

select()

listen socket descriptor

Sockets or Files descriptors Or Pipe descriptors

fd1 **fd2** **fd3** **fd4** **fd5**

Remote Clients Or files Or pipes

C1 C2 F3 F4 P5

I/O Multiplexing

Application

Operating system

select()
on multiple fds

System call →

No data ready

Time

Process blocks

Wait for data
on any fd

Return readable ←

data ready

read()/recv()

System call →

Copy data

Copy data to user

Process blocks

Return ok ←

Read data

Copy complete

Signal driven I/O

Application　　　　　　　　　　　　**Operating system**

Establish SIGIO　　System call　　→ **No data ready**
Signal handler　　←　return

Time

Process continues

　　　　　　　　　　Deliver SIGIO
Signal Handler　←　　　　　　　**data ready**
read()/recv()　　System call　　→ **Copy data**

　　　　　　　　　　　　　　Wait for data

Process blocks　　　　　　　　　Copy data to user

Read data　←　Return ok　　　**Copy complete**

Asynchronous I/O

Application
aio_read()

Operating system

System call $\longrightarrow$ **No data ready**

$\longleftarrow$ return

Time

Wait for data

Process continues

data ready

Copy data

Copy data to user

Signal handler
Read data

Deliver signal
specified in
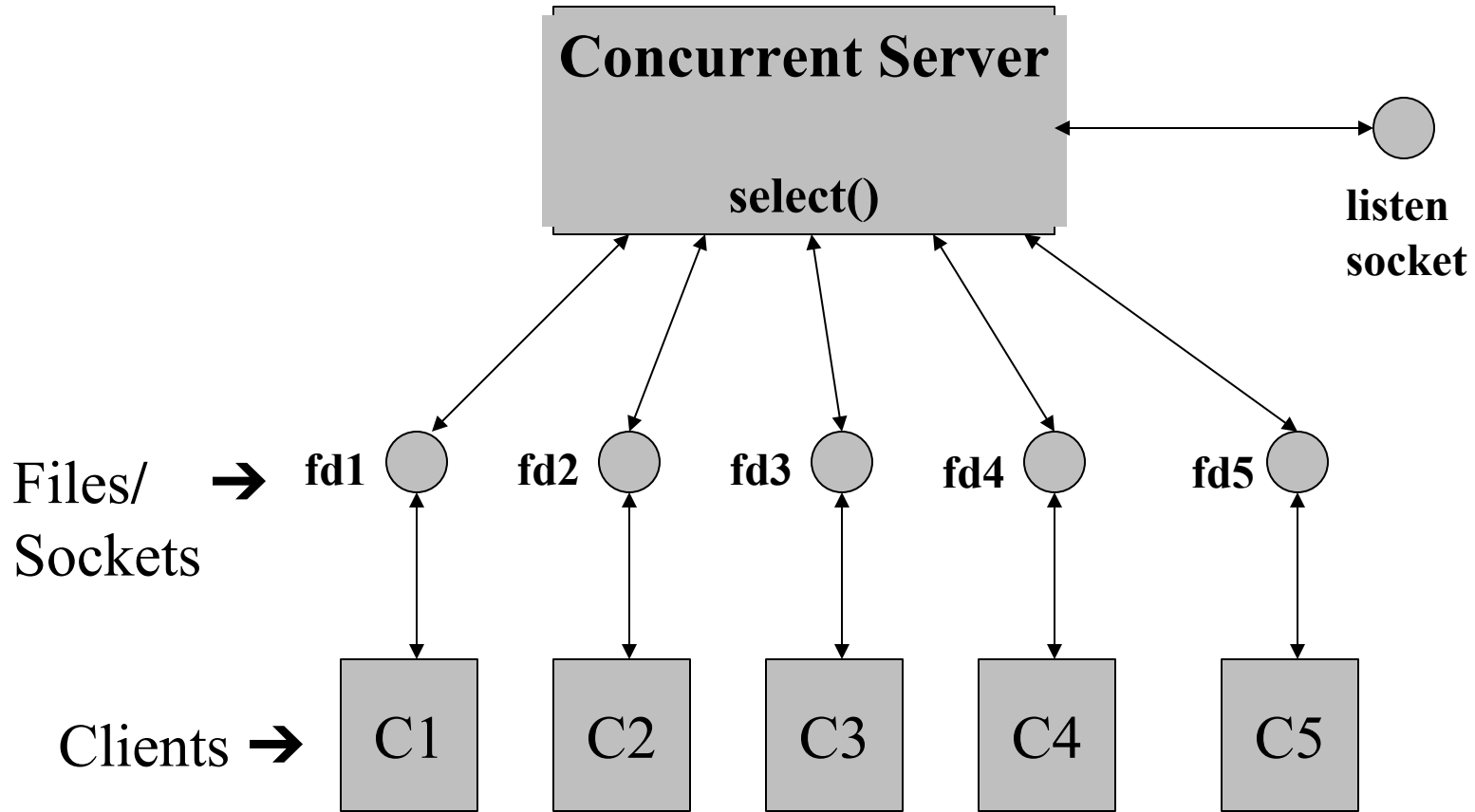**aio_read()** $\longleftarrow$

Copy complete

I/O Multiplexing

Example of Event-oriented programming

What is I/O multiplexing?

- When an application needs to handle multiple I/O descriptors at the same time
  - E.g. file and socket descriptors, multiple socket descriptors

- When I/O on any one descriptor can result in blocking

Non-forking concurrent server

Concurrent Server

select()

listen socket

Files/ Sockets →

fd1 **fd2** **fd3** **fd4** **fd5**

Clients → C1 C2 C3 C4 C5

select() call

- Allows a process to wait for an event to occur on any one of its descriptors.

- Types of event
  - ready for read
  - ready for write
  - Exception condition

select() call

```
int  select(
    int  maxfdp1,                  /* max. fd + 1 */
    fd_set  *readfds,  /* read ready? */
    fd_set  *writefds,      /* write ready? */
    fd_set *exceptfds,      /* exceptions? */
    struct timeval *timeout);

struct  timeval {
    long tv_sec;  /* seconds */
    long tv_usec; /* microseconds */
}
```

struct fd_set

- Set of descriptors that we want to wait on for events.

- Typically holds 256 descriptor states.

- Manipulation macros
  - void FD_ZERO(fd_set *fds)
  - void FD_SET (int fd, fd_set *fds)
  - void FD_CLR (int fd, fd_set *fds)
  - int FD_ISSET(int fd, fd_set *fds)

Non-forking Concurrent Server

```
fdset rdset, wrset;
int listenfd, connfd1, connfd2;
int maxfdp1;
………
Connection establishment etc.
………

/* initialize */
FD_ZERO(&rdset);
FD_ZERO(&wrset);
```

```
for( ;; ) {
    FD_SET(connfd1, &rdset);
    FD_SET(connfd2, &wrset);
    FD_SET(listenfd, &rdset);

    maxfdp1 = max(connfd1, connfd2, listenfd) + 1;

    /* wait for some event */
    Select(maxfdp1, &rdset, &wrset, NULL, NULL);

    if( FD_ISSET(connfd1, &rdset) ) {
      Read data from connfd1…
    }
    if( FD_ISSET(connfd2, &wrset) ) {
      Write data to connfd2…
    }
    if( FD_ISSET(listenfd, &rdset) {
      Process a new connection…
    }
}
```