The C10K Problem and Solutions

References:

- http://www.kegel.com/c10k.html
- http://www.monkey.org/~provos/libevent
- http://byteliu.com/2019/05/08/LINUX-%E2%80%93-IO-MULTIPLEXING-%E2%80%93-SELECT-VS-POLL-VS-EPOLL/

- A Scalable and Explicit Event Delivery Mechanism for UNIX, http://static.usenix.org/event/usenix99/full_papers/banga/banga.pdf

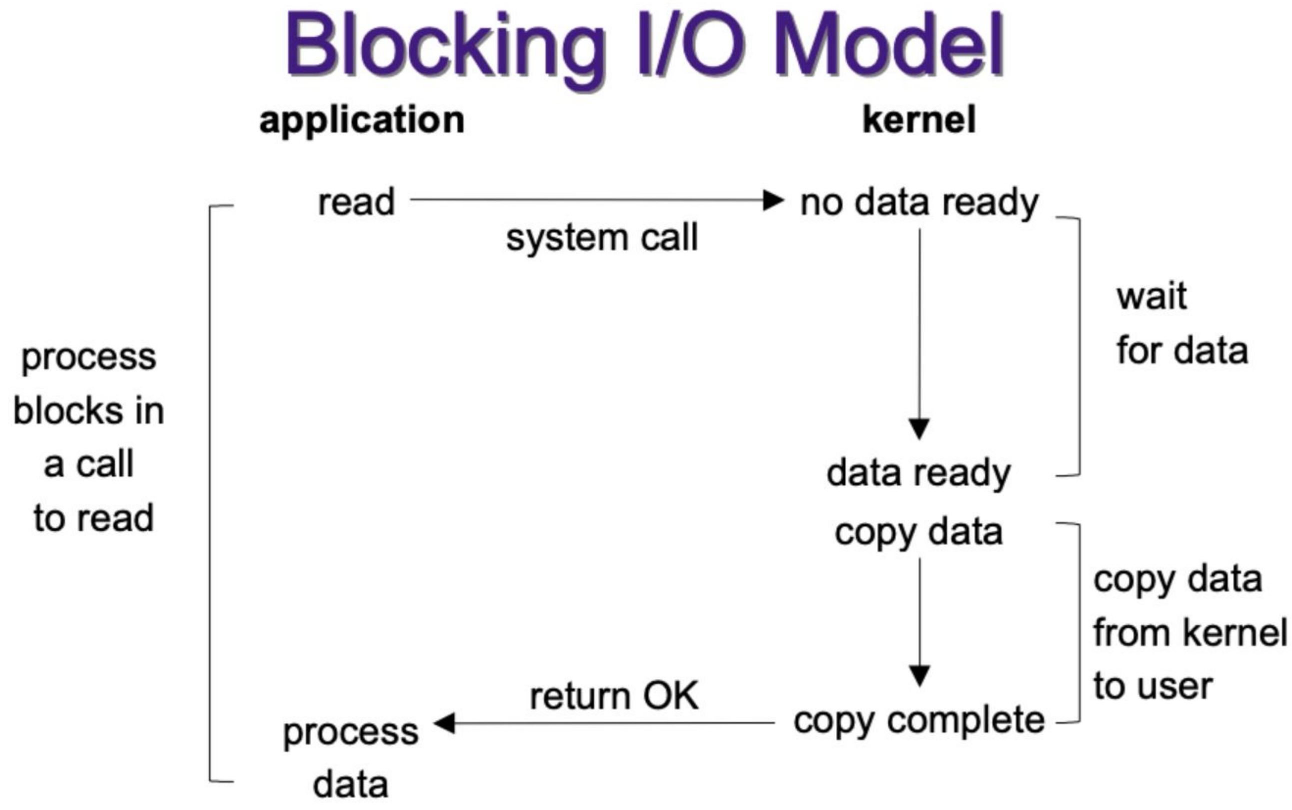
- Acknowledgement: Modified from the slides of Che-Yi Lin and Hao-Yun Liu.

Outline

- Background
  - Types of I/O
  - Problem of select()
- The C10K problem
- Asynchronous I/O (AIO)
- Design of networking software
- Using libevent
- Benchmarks

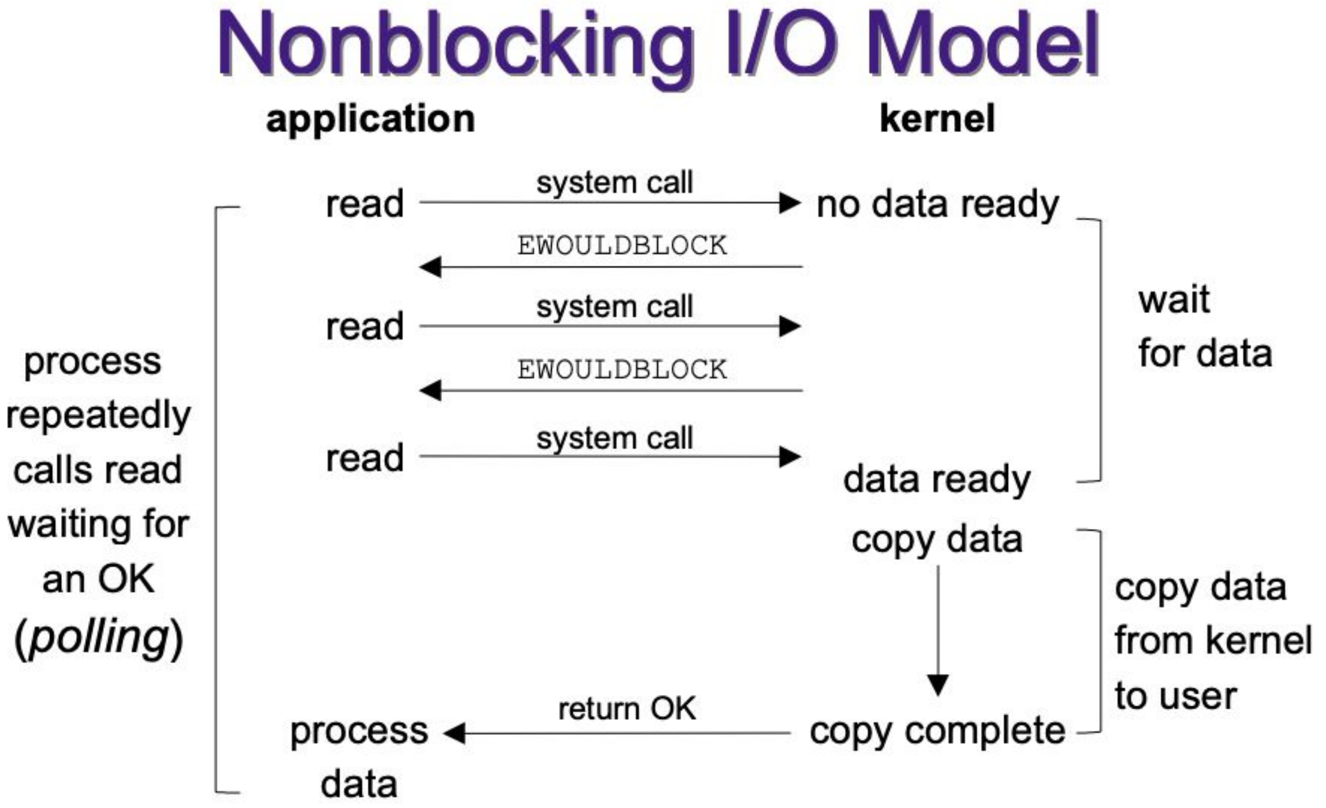
Background: Types of I/O

- Wait until data is ready



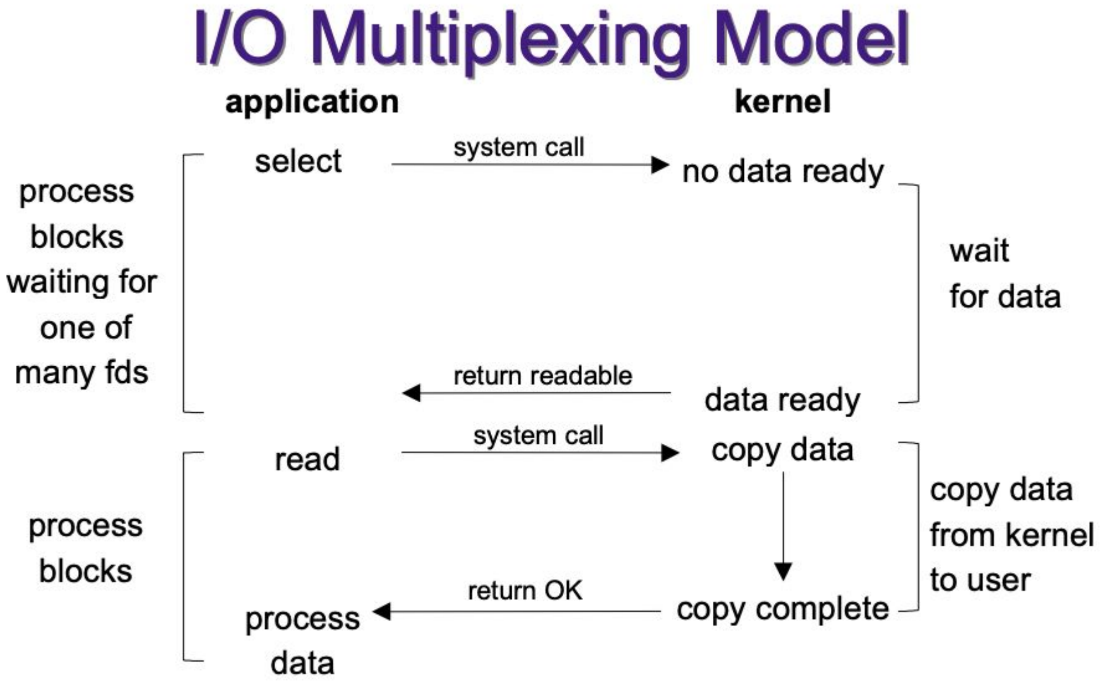
Background: Types of I/O (cont.)

- Return an error if data was not ready

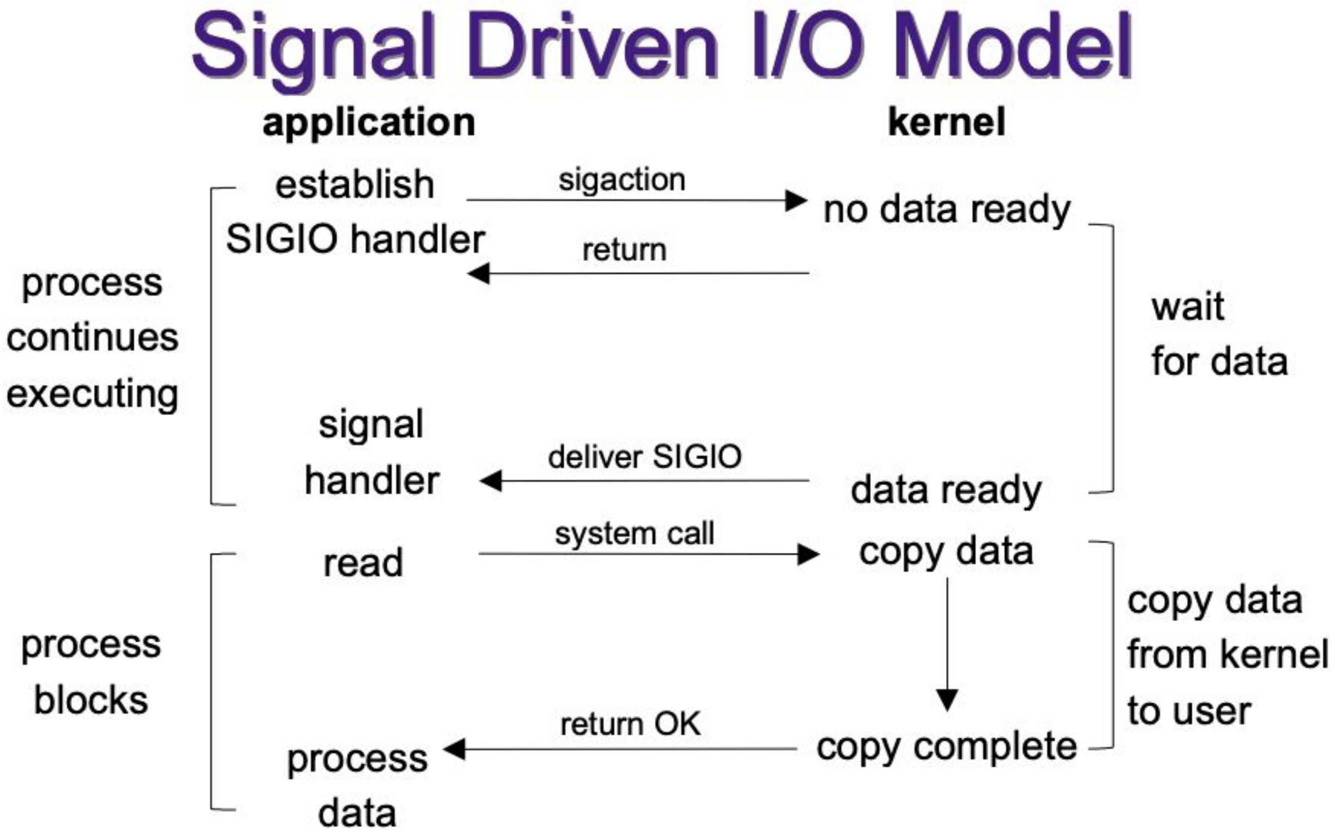


Background: Types of I/O (cont.)

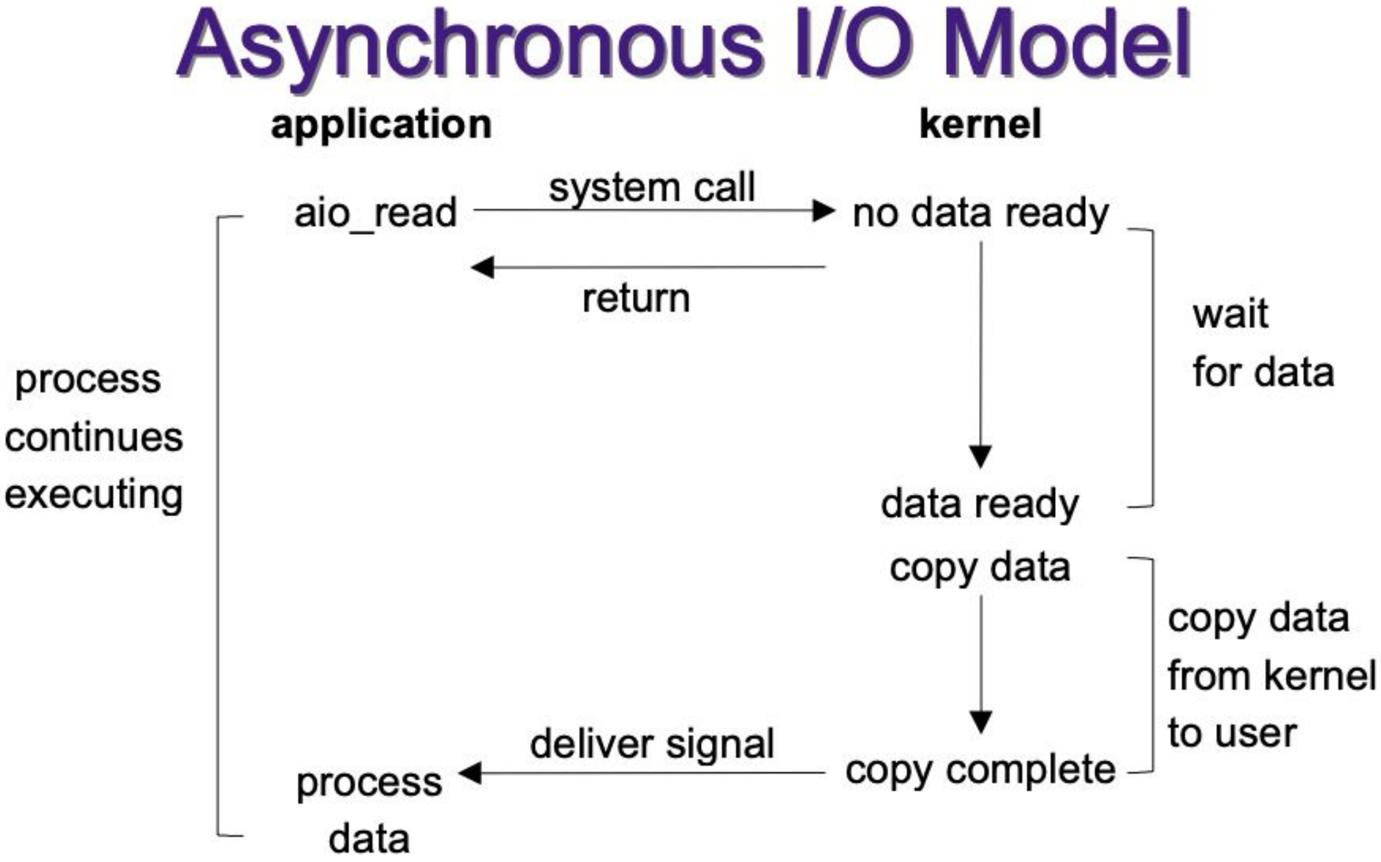
- Block until the given file descriptors are ready to perform I/O
- Block select() or poll(), instead of blocking I/O
  - select()
  - poll()
  - epoll()



Background: Types of I/O (cont.)


Signal Driven I/O Model

Background: Types of I/O (cont.)



Asynchronous I/O Model

application

kernel

aio_read

system call

no data ready

return

wait
for data

process
continues
executing

data ready

copy data

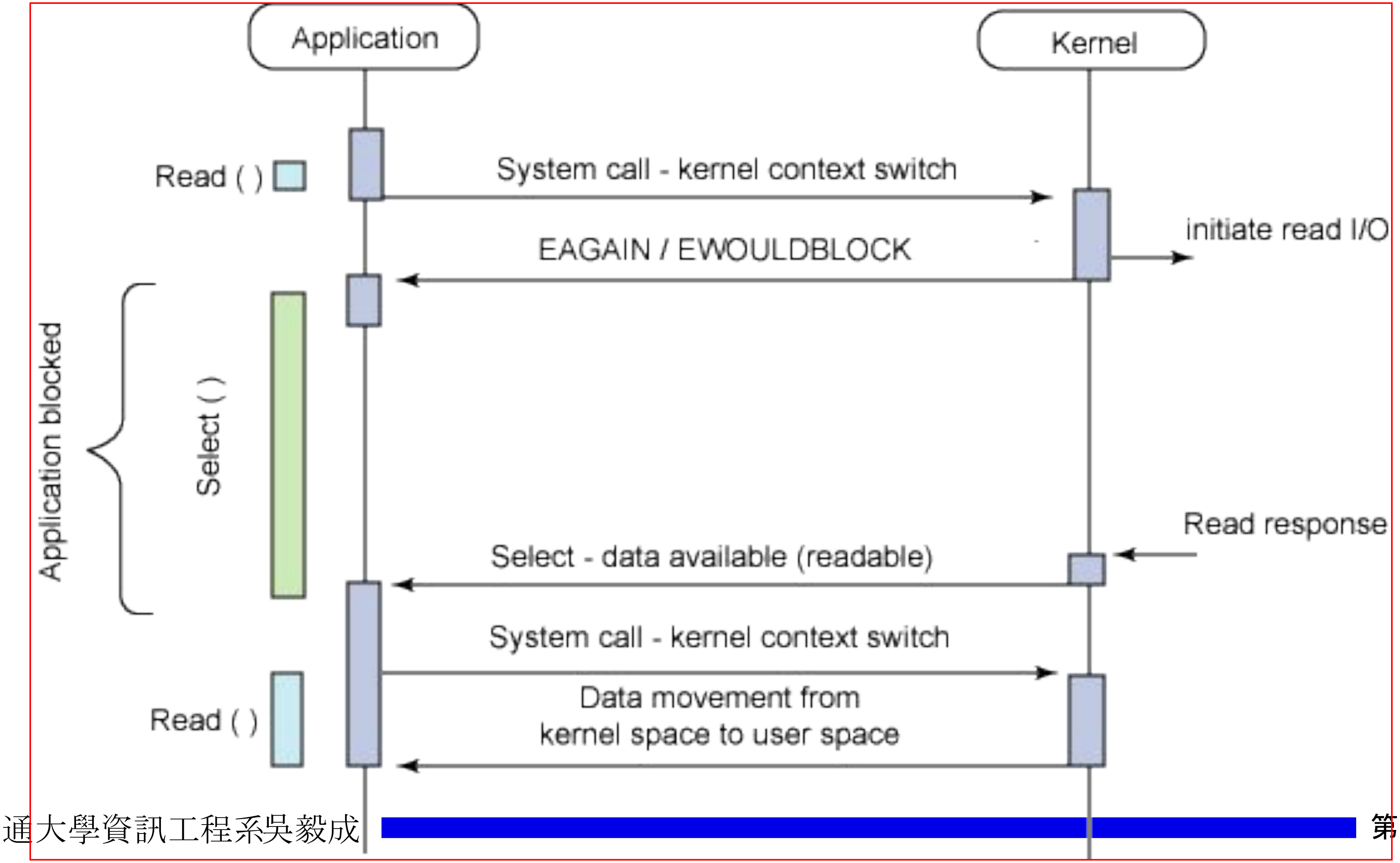
copy data
from kernel
to user

deliver signal

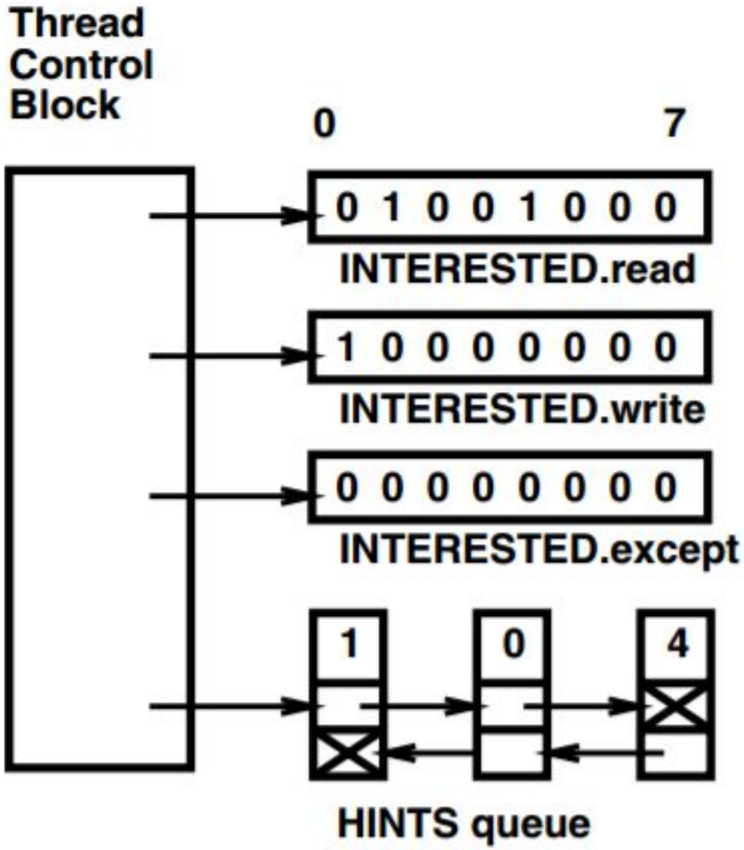
copy complete

process
data

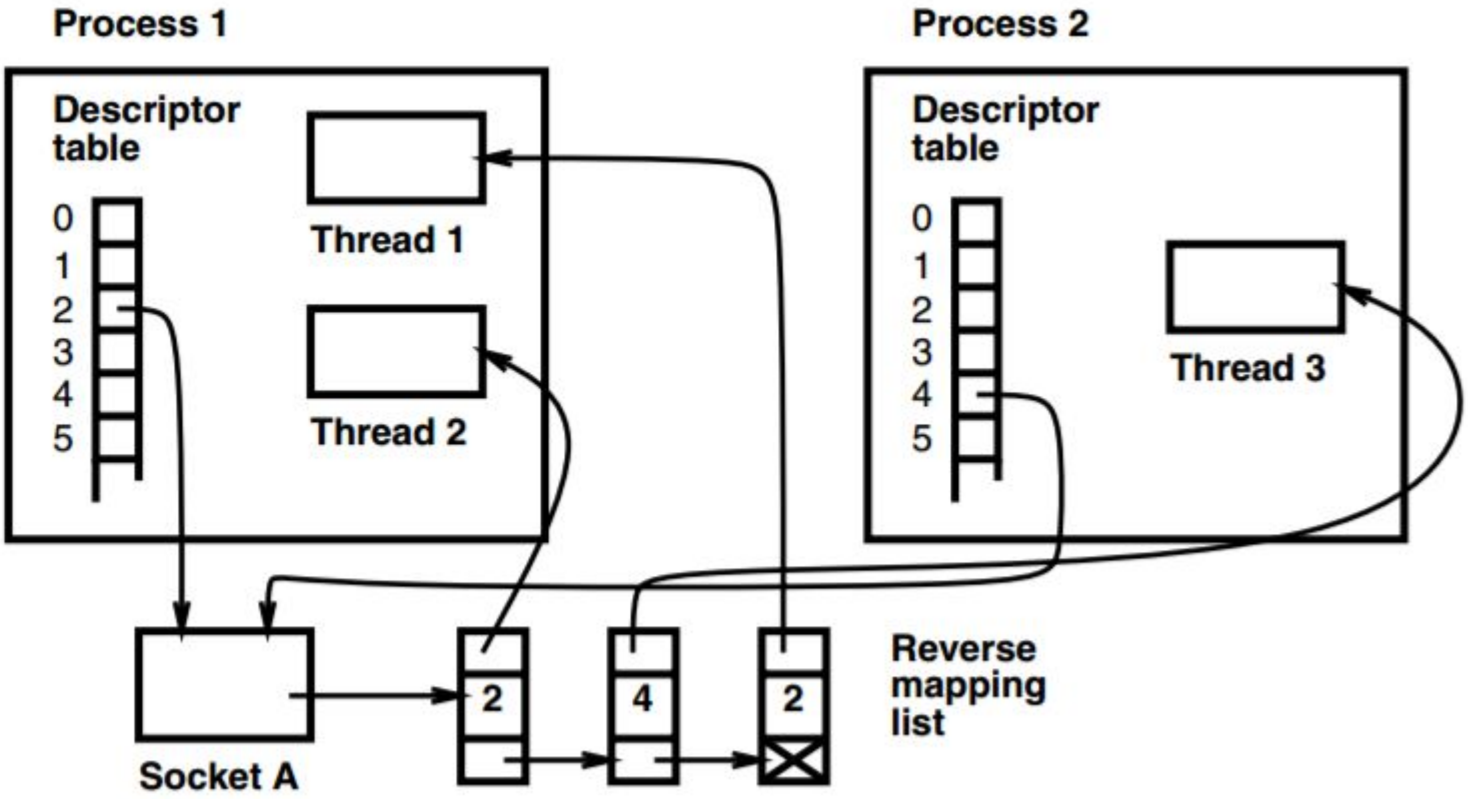
Background: select()


Application
Kernel
Read ( )
System call - kernel context switch
initiate read I/O
EAGAIN / EWOULDBLOCK
Application blocked
Select ( )
Read response
Select - data available (readable)
System call - kernel context switch
Data movement from
kernel space to user space
Read ( )

Per-Thread Data Structure



Per-Socket Data Structure

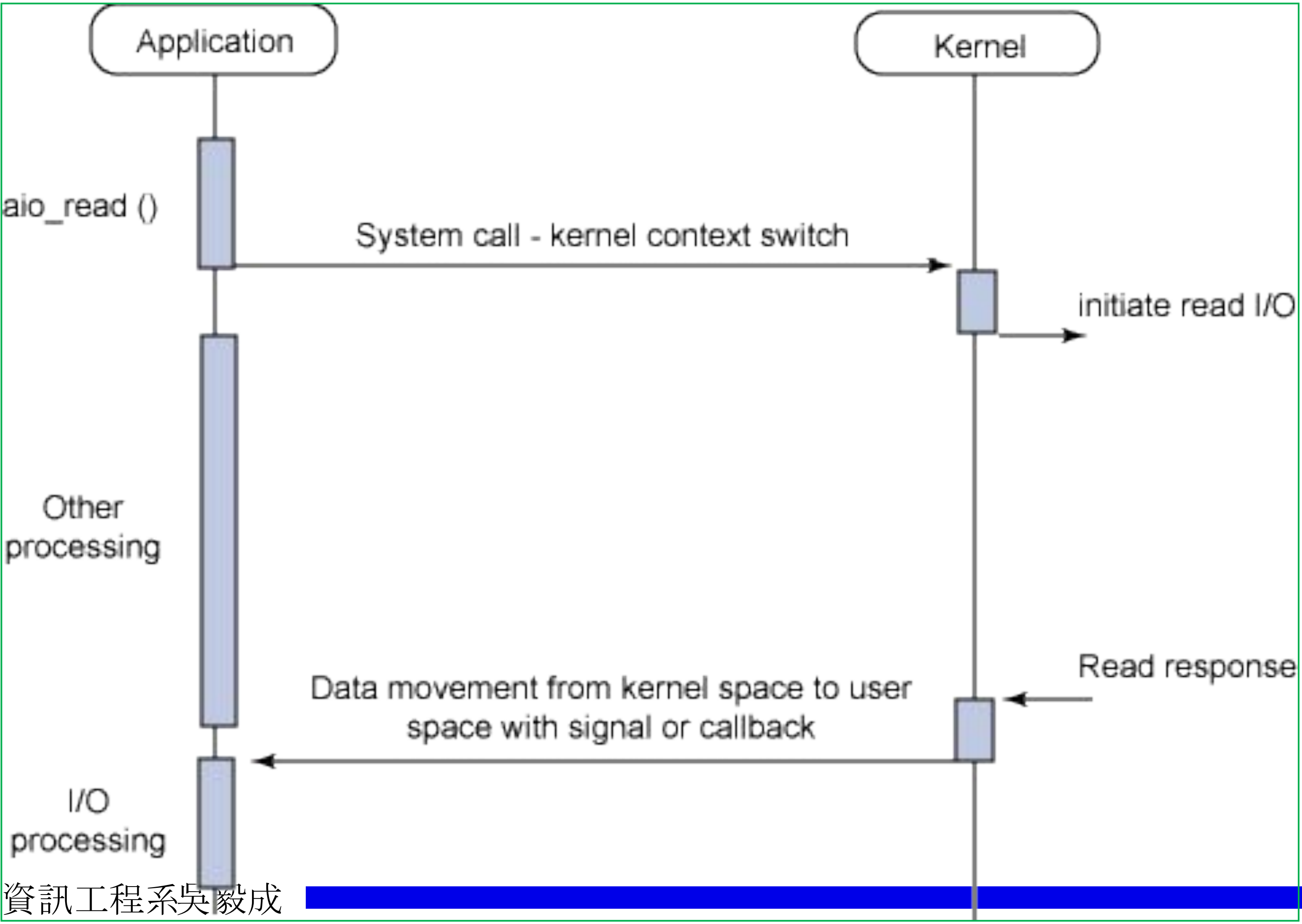


The C10K problem

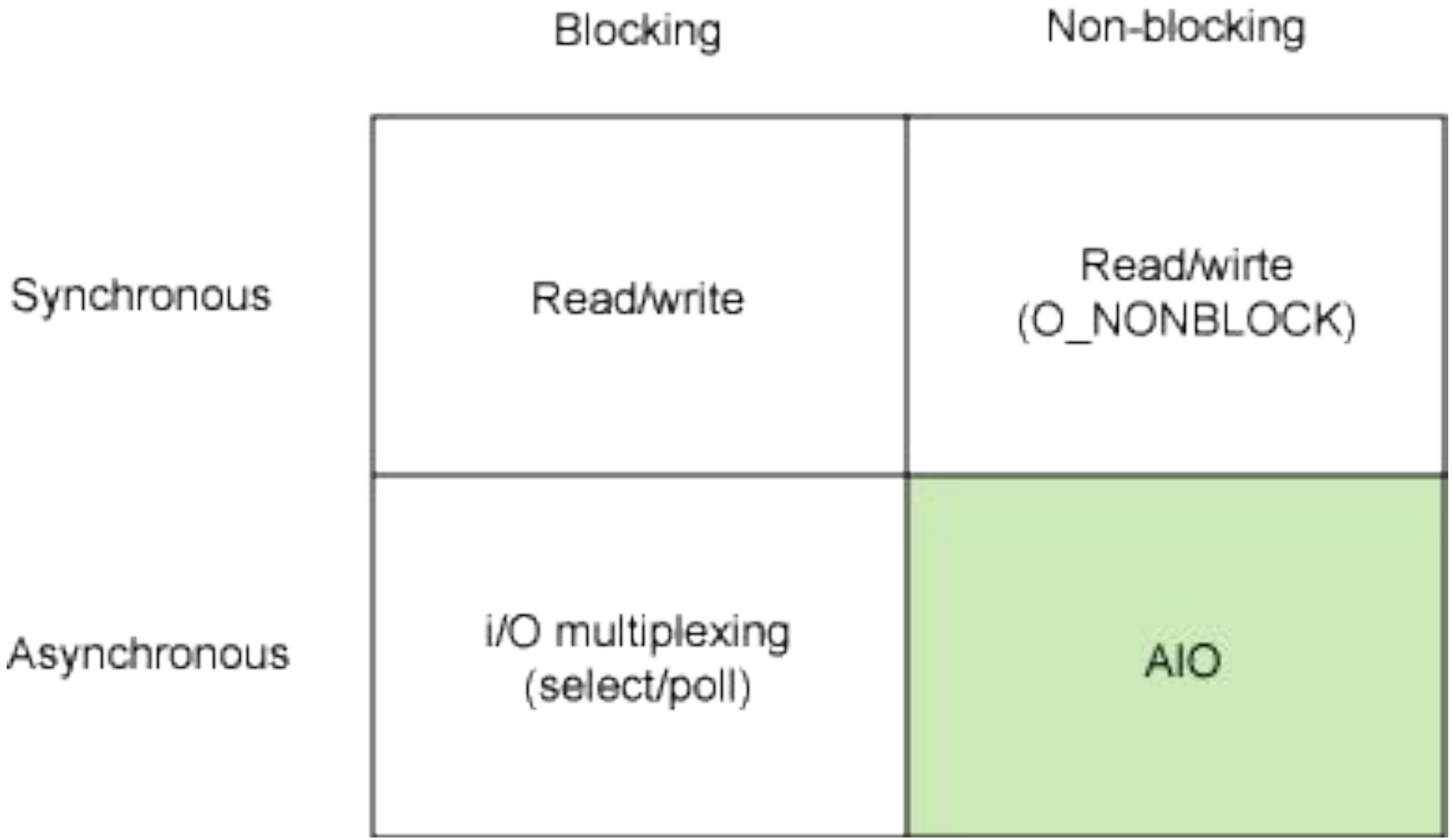
- Web servers have to handle ten thousand clients simultaneously
- Web is a big place now
- Hardware is no longer the bottleneck

http://www.kegel.com/c10k.html

Asynchronous I/O (AIO)


Application
Kernel
aio_read ()
System call - kernel context switch
initiate read I/O
Other processing
Read response
Data movement from kernel space to user space with signal or callback
I/O processing

AIO in Linux



	Blocking	Non-blocking
Synchronous	Read/write	Read/wirte (O_NONBLOCK)
Asynchronous	i/O multiplexing (select/poll)	AIO

AIO in Linux (cont.)

- Introduced in Linux kernel 2.6 (released at 2008) and also available in 2.4 if patched.
- The completion of I/O can be notified by two method.
  - Signal.
  - Register a completion handler function to create a new thread.
- API:
  - aio_read
  - aio_error
  - aio_return
  - aio_write
  - aio_suspend
  - aio_cancel
  - lio_listio

Note: someone says the AIO in Linux is not as efficient as W

Design of application

- Use **`fork()`**

  defect：High overhead for each connection
  solution：Return to accept() and child process die automatically
  example：**Apache 1.3**

- Use **`pthread_create()`**

  defect：Thread-safe and Memory-leak problems
  solution：Use Thread-safe library and Garbage collection library
  example：**Apache 2.0 Thread MPM**

Design of application (cont.)

- Event-based process

  advantage：Without overhead of create process or thread, no need to use Share Memory or Mutex for process / thread

  hard to implement：

  - BSD □ kqueue(), Linux □ epoll(), Solaris □ /dev/poll
    None of these are Standard !
  - Buffering of nonblocking I/O

  Solution：libevent library

Using libevent

- libevent by Niels Provos
  http://www.monkey.org/~provos/libevent/
- Is a lightweight C I/O framework
- Support **`kqueue()`**, **`epoll()`**, **`/dev/poll`**, and the traditional **`select()`**, **`poll()`**
- Under 3-clause BSD license !!

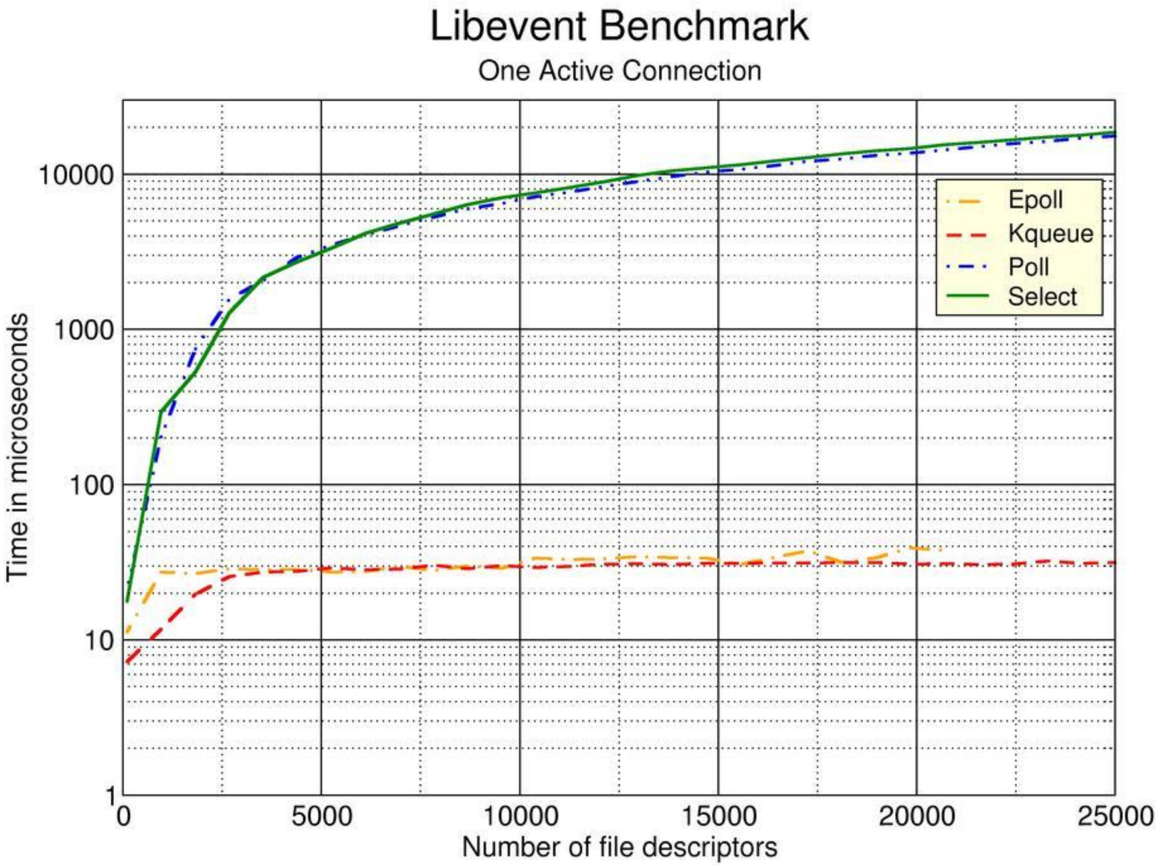
Using libevent (cont.)

- Execute a callback function when a specific event occurs on a file descriptor or after a timeout has been reached
- Replace the event loop found in event driven network servers
- An application just needs to call event_dispatch() and then add or remove events dynamically without having to change the event loop
- Can also be used for multi-threaded applications

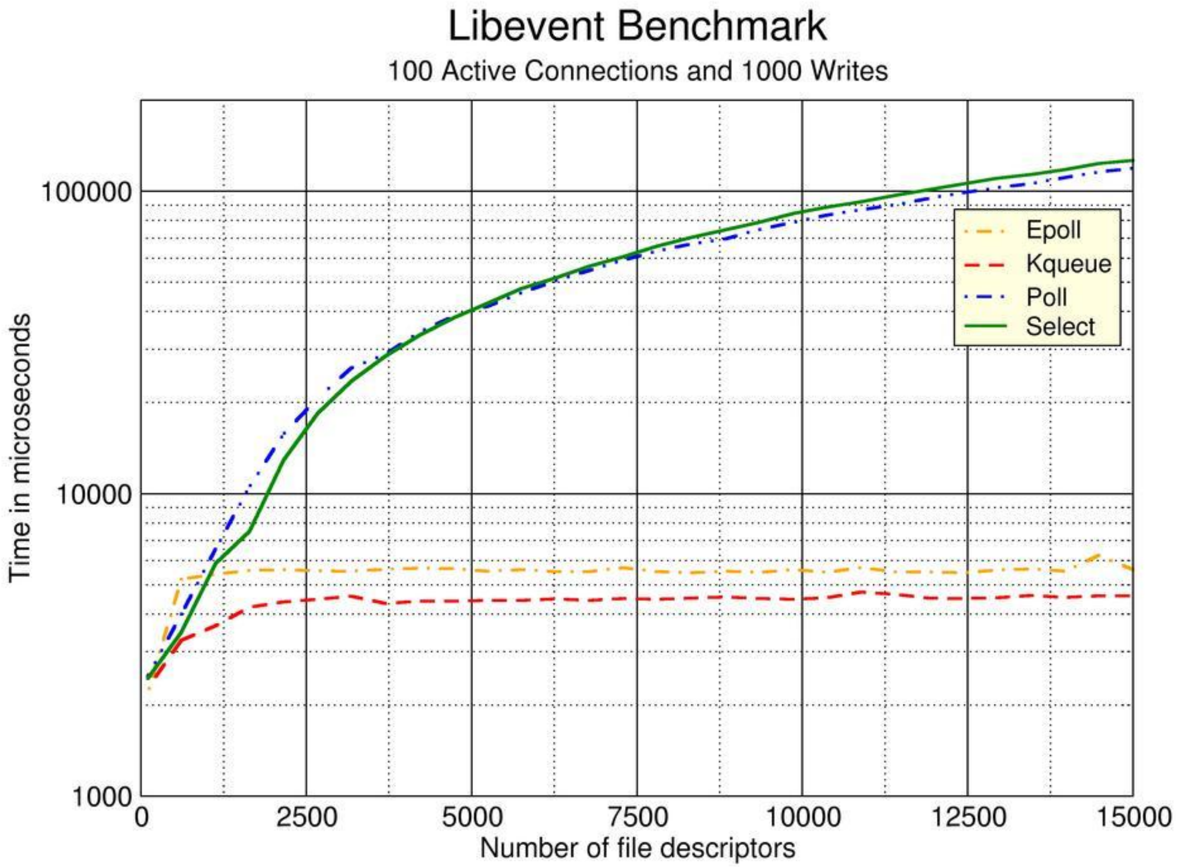
Using libevent (cont.)

```
/* Initial libevent. */
event_init();

/* Create event. */
struct event ev;
event_set(&ev, sfd, EV_READ | EV_PERSIST,
  connection_accept, &ev);

/* Add event. */
event_add(&ev, NULL);

event_dispatch();
```

Benchmarks of libevent



Benchmarks (cont.)


Libevent Benchmark
100 Active Connections and 1000 Writes
Time in microseconds
Number of file descriptors
Epoll
Kqueue
Poll
Select