I/O Multiplexing

Giảng viên: TS. Trần Hải Anh
Khoa KTMT
Trường CNTT & TT

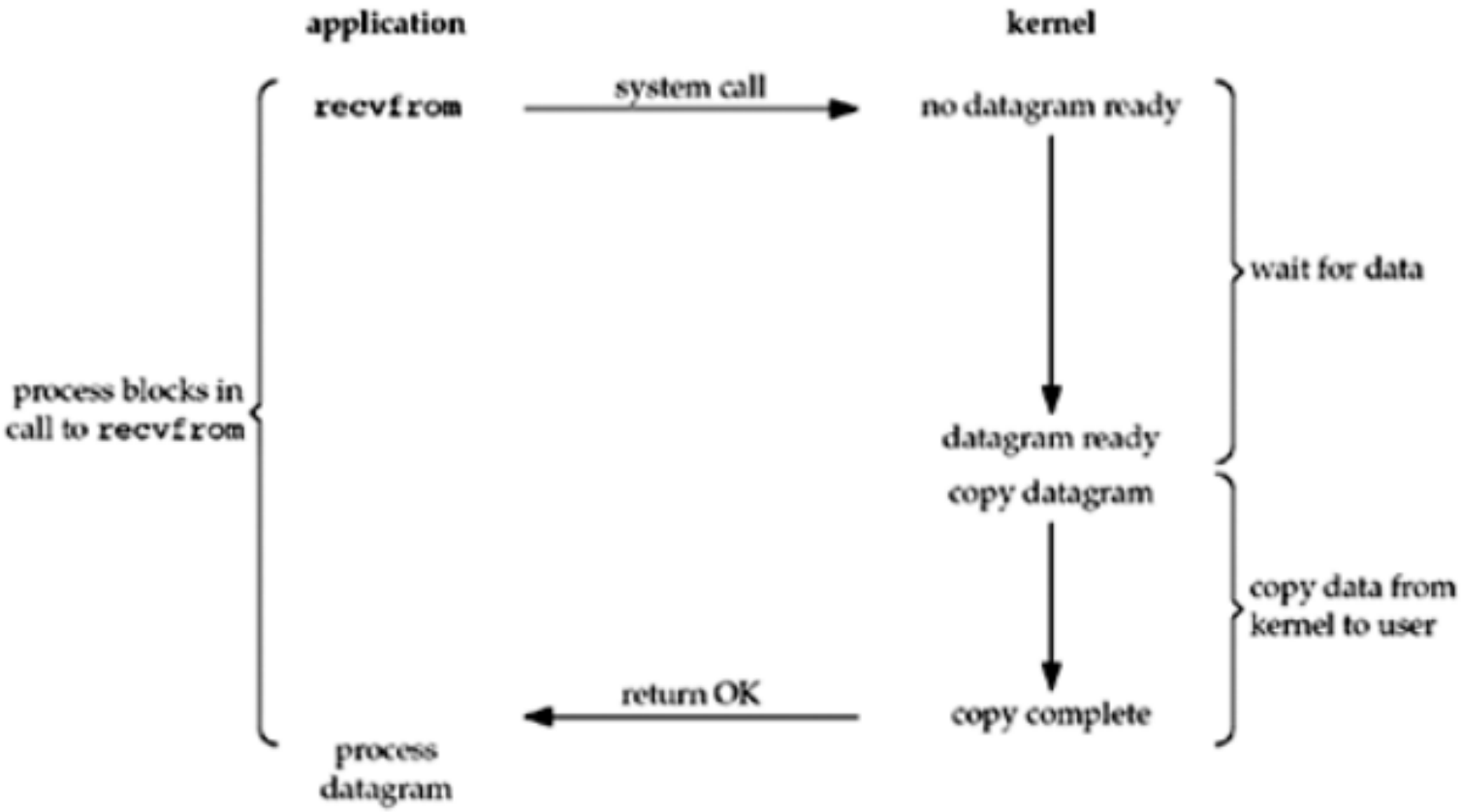
Intro

- The client has to handle two inputs at the same time: **standard input** and a **TCP socket.**
- Problem: when the client was blocked in a call to fgets (on standard input) and the server process was killed. The server TCP correctly sent a FIN to the client TCP, but since the client process was blocked reading from standard input => it never saw that.
- Need the capability to tell the kernel that we want to be notified if one or more I/O conditions are ready
- **I/O Multiplexing** (**select** & **poll** functions)

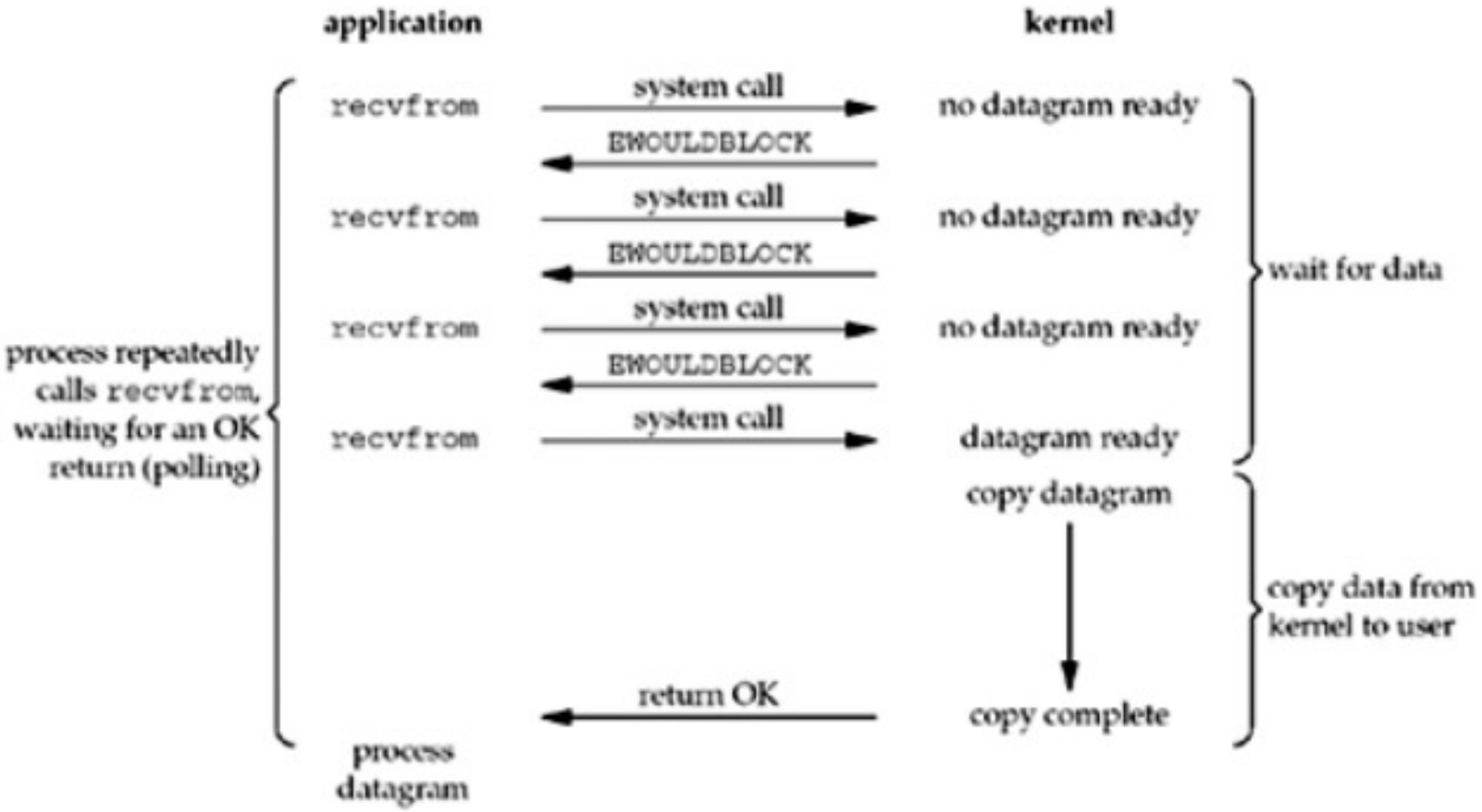
I/O Models

- blocking I/O
- nonblocking I/O
- I/O multiplexing (*select* and *poll*)
- signal driven I/O (*SIGIO*)
- asynchronous I/O (the POSIX *aio*_functions)

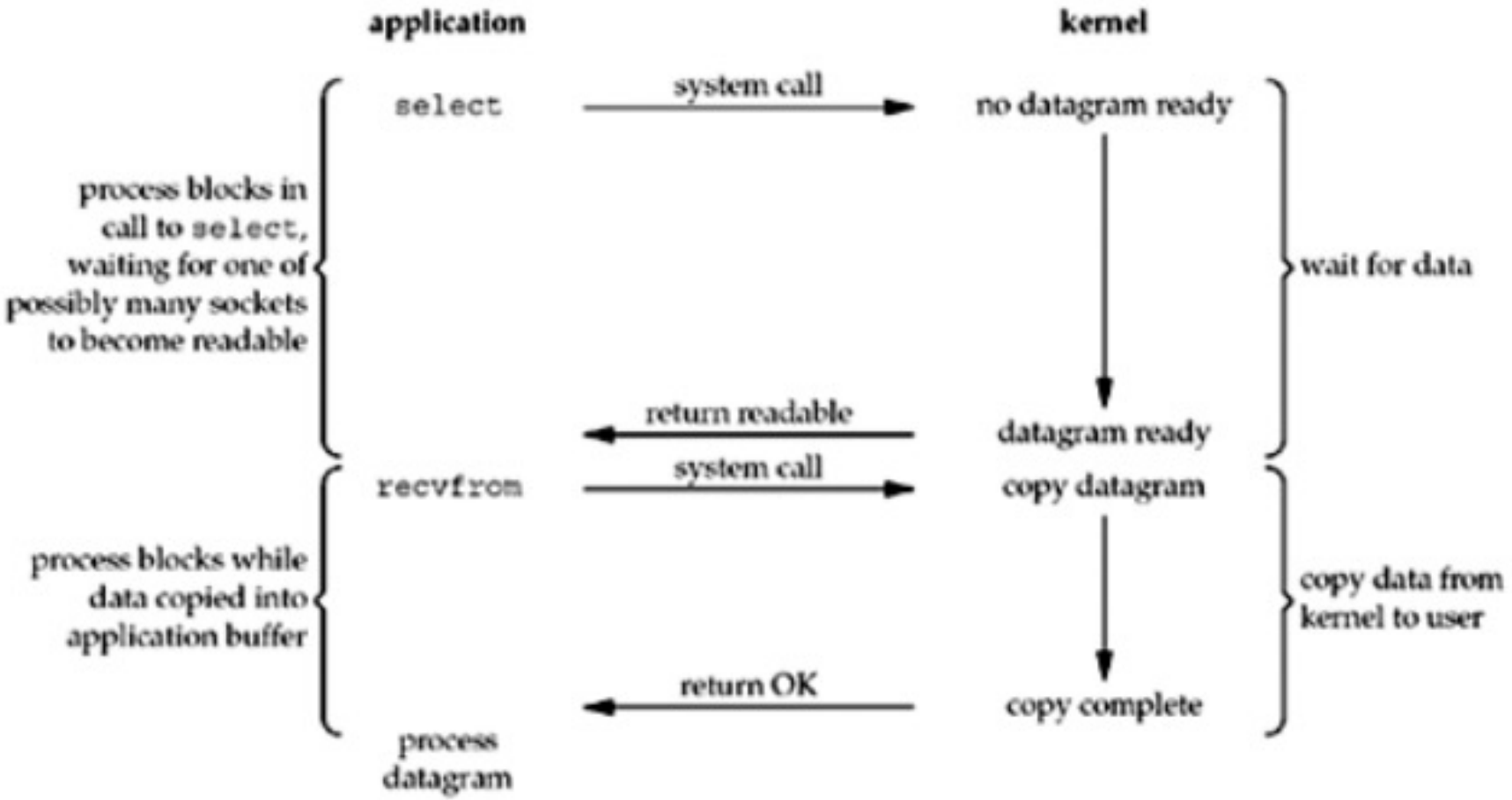
Blocking I/O Model



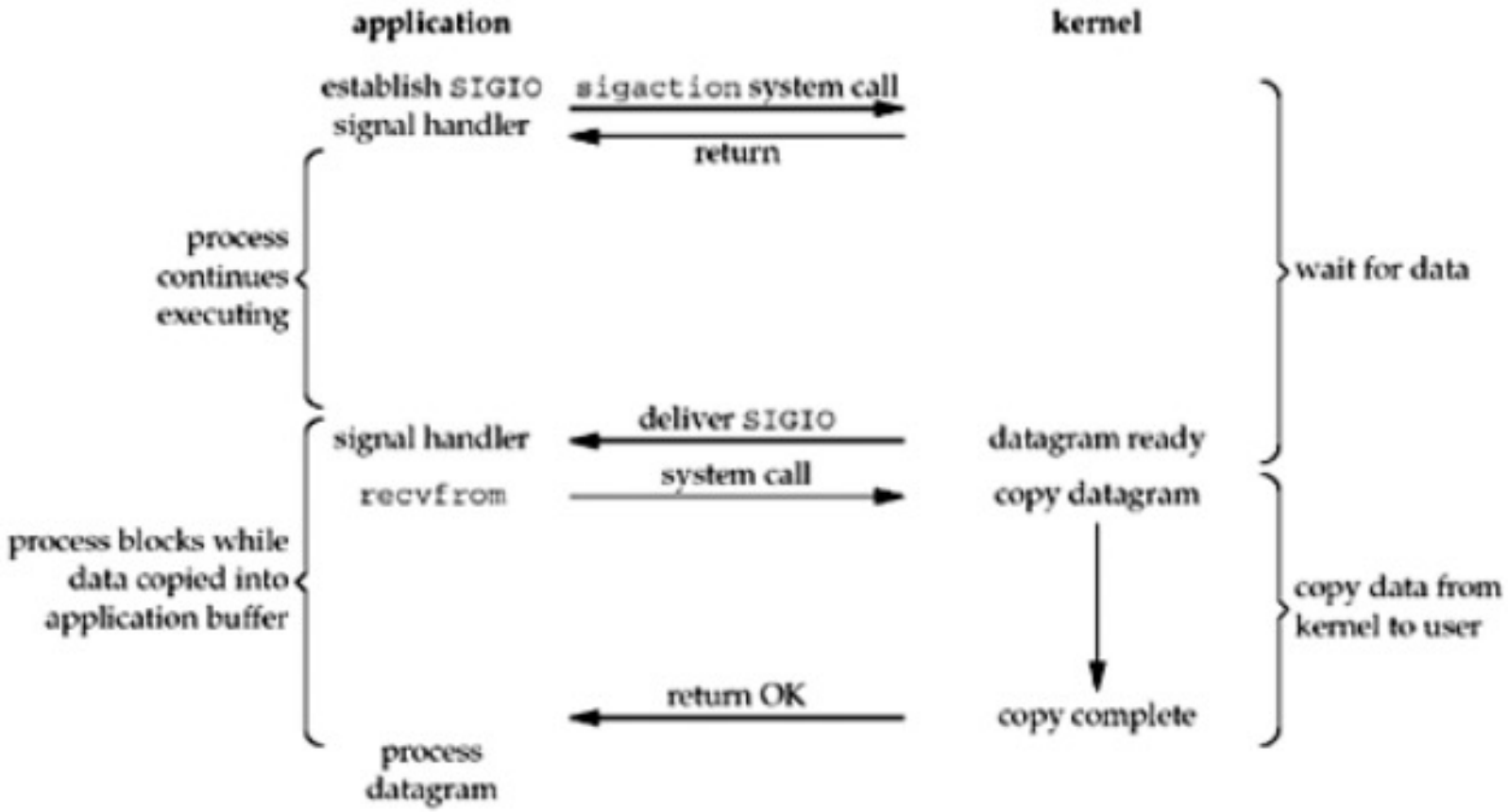
Nonblocking I/O Model



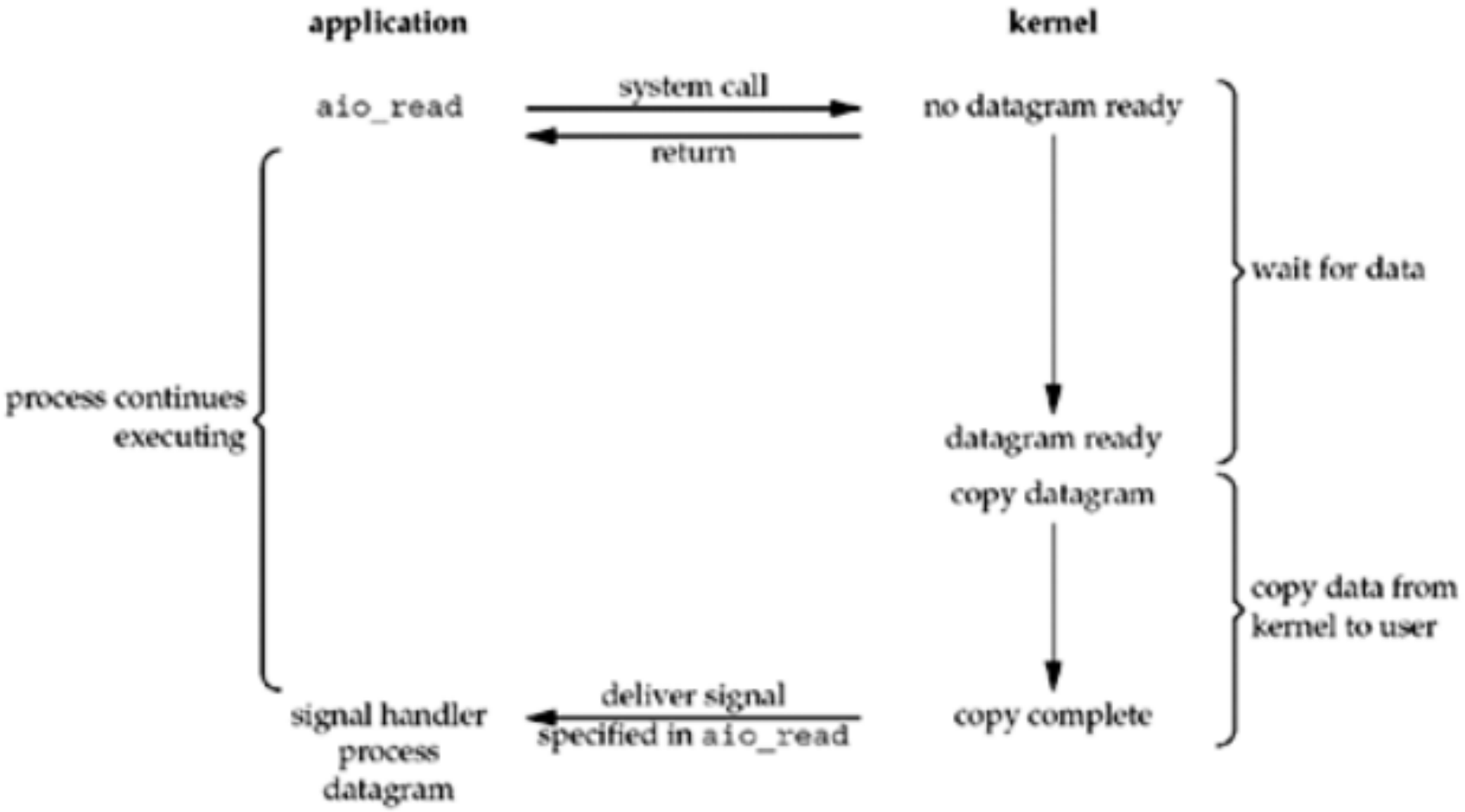
I/O Multiplexing Model



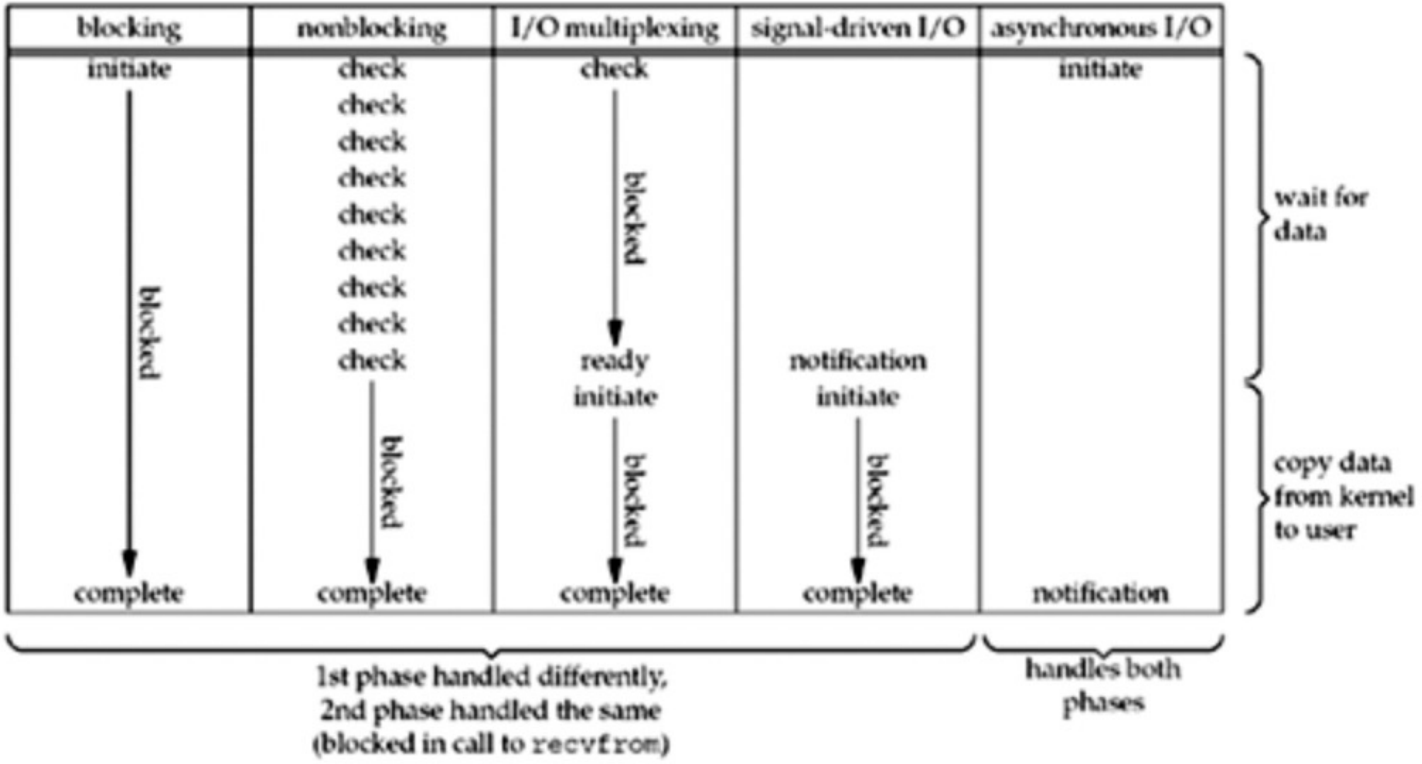
Signal-Driven I/O Model



Asynchronous I/O Model



Comparison of the five I/O models



select function

- This function allows the process to instruct the kernel to wait for any one of multiple events to occur and to wake up the process only when one or more of these events occurs or when a specified amount of time has passed.
- Example:
  - Any of the descriptors in the set {1, 4, 5} are ready for reading
  - Any of the descriptors in the set {2, 7} are ready for writing
  - Any of the descriptors in the set {1, 4} have an exception condition pending
  - 10.2 seconds have elapsed

select function

```
#include <sys/select.h>
#include <sys/time.h>
int select(int maxfdp1, fd_set *readset, fd_set
  *writeset, fd_set *exceptset, const struct
  timeval *timeout);
```

- Returns: positive count of ready descriptors, 0 on timeout, –1 on error

Arguments of *select* function

- *const struct timeval *timeout* argument
- Structure:

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

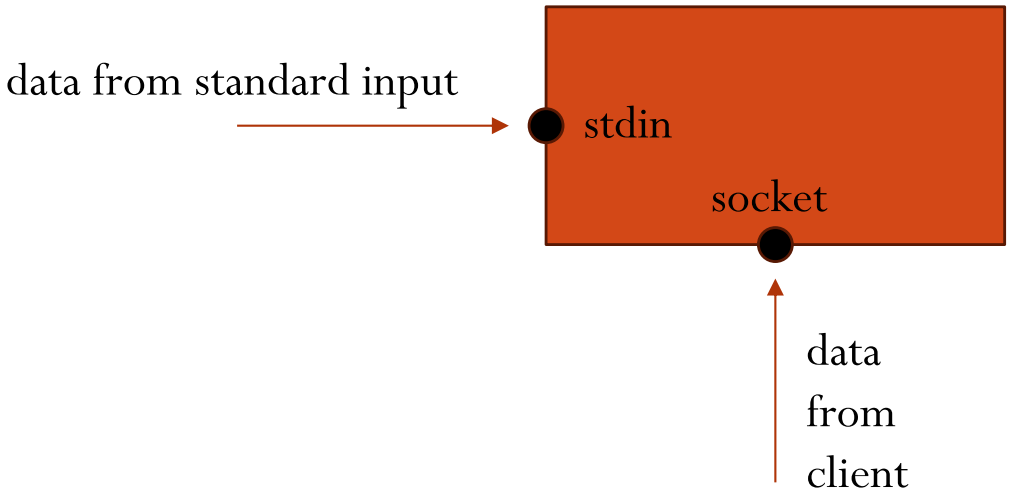
- 3 possibilities:
  - Wait forever => null pointer.
  - Wait up to a fixed amount of time
  - Do not wait at all: This is called polling. => assign to 0

Arguments of *select* function

- The three middle arguments, *readset*, *writeset*, and *exceptset*, specify the descriptors that we want the kernel to test for reading, writing, and exception conditions.

```
void FD_ZERO(fd_set *fdset);
void FD_SET(int fd, fd_set *fdset);
void FD_CLR(int fd, fd_set *fdset);
int FD_ISSET(int fd, fd_set *fdset);
```

Server with two inputs at the same time



Chương trình ví dụ:

select-client.c

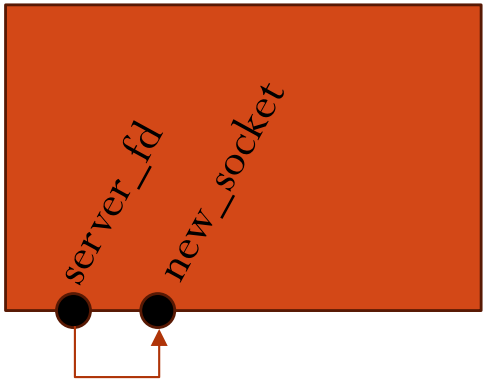
select-server.c

Single-process server using *select()*

client_sockets[]

0
0
0
0
0
0
0
0

5

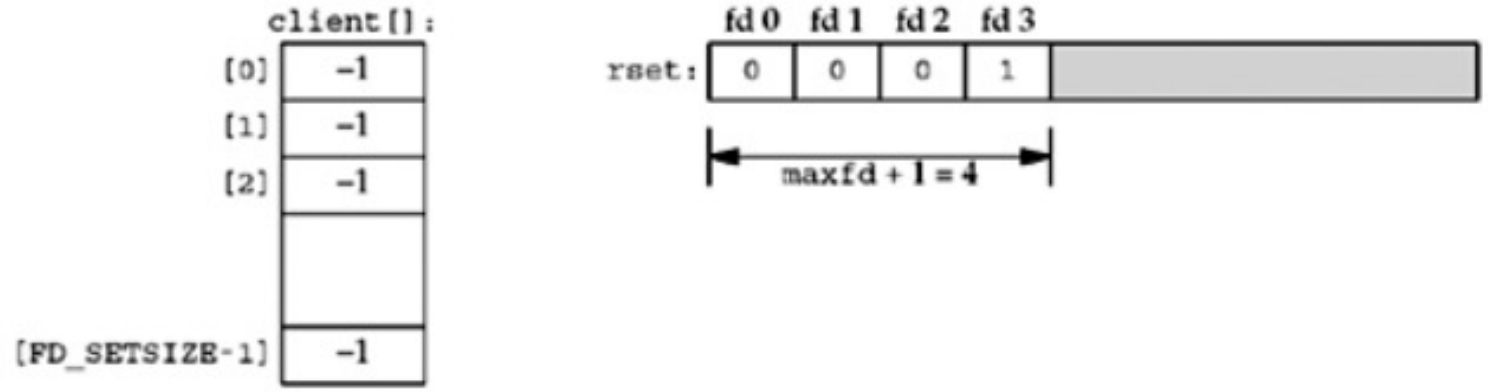


Chương trình ví dụ:

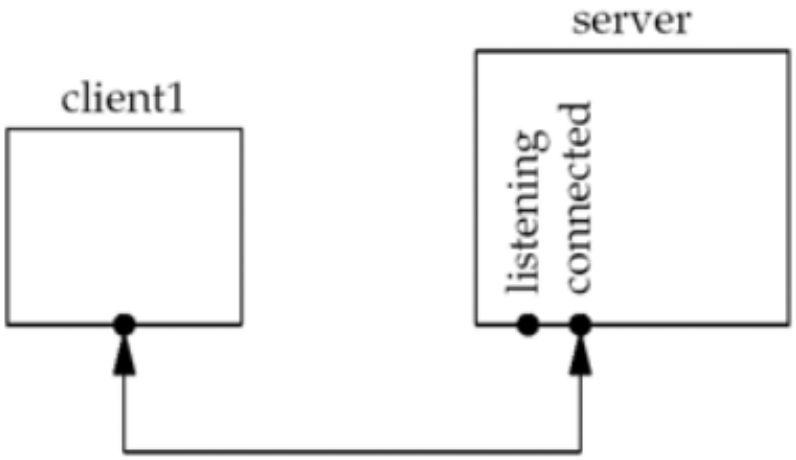
single-process-select-server.c

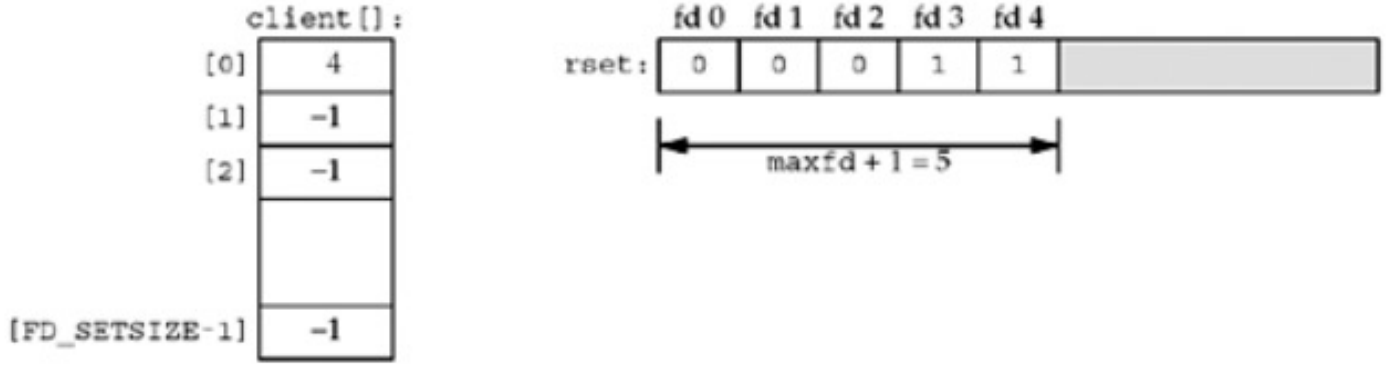
select-client.c

Single-process server using *select()*

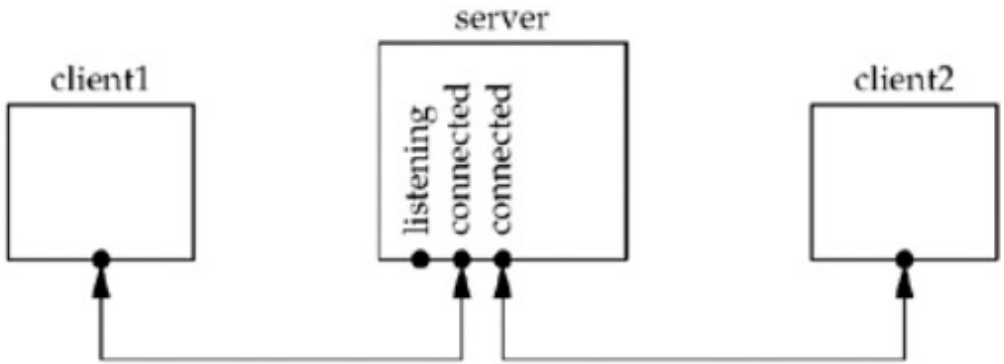


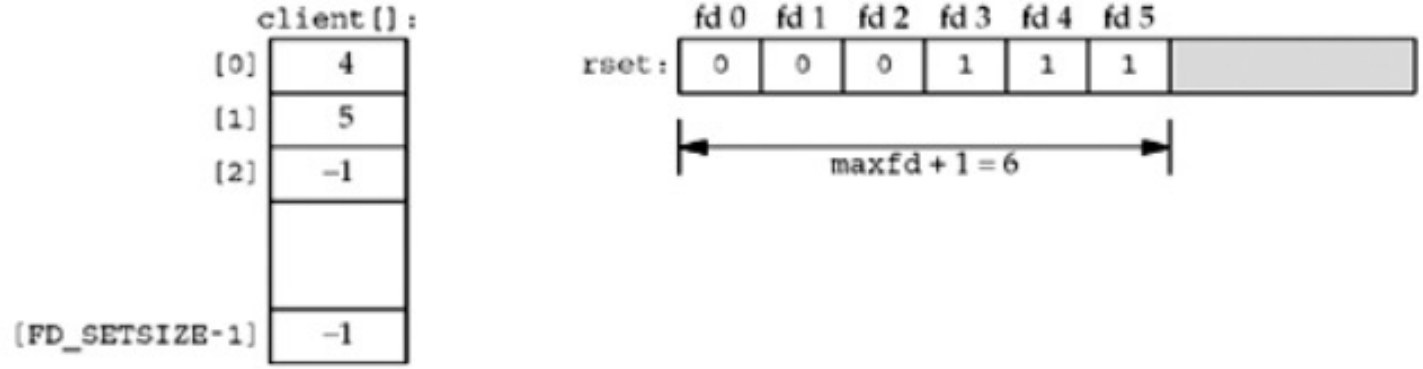
Single-process server using *select()*



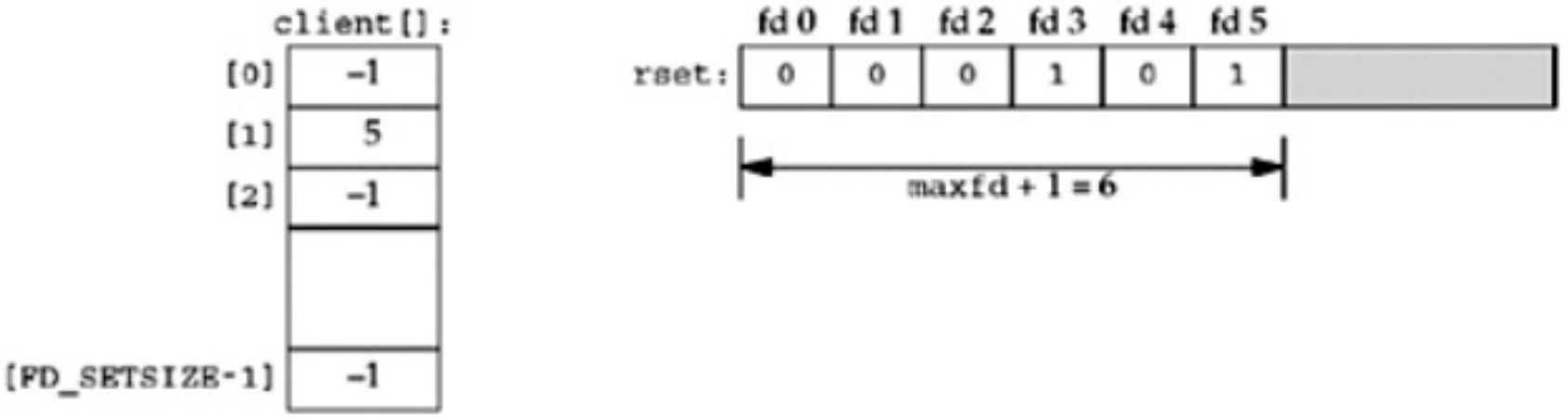


Single-process server using *select()*





Single-process server using *select()*



Data structures after first client terminates its connection.

pselect()

- The select() and pselect() functions are both used in POSIX systems to monitor multiple file descriptors, waiting until one or more of them become "ready" for some class of I/O operation (e.g., input ready to read, output ready to write, etc.).

```
#include <sys/select.h>
#include <signal.h>
#include <time.h>
int pselect (int maxfdp1, fd_set *readset, fd_set *writeset, fd_set *exceptset, const struct timespec *timeout, const sigset_t *sigmask);
```

- The key differences between these two functions are:
  - Signal Masking
  - Time Resolution

pselect()

- **Signal Masking:**
  - select(): This function does **not** provide any mechanism for atomically modifying signal masks while waiting for file descriptors to be ready.
  - pselect(): This function allows you to atomically set a signal mask before waiting and restore the previous signal mask afterward. It ensures that signals can be blocked during the wait and thus prevents race conditions where a signal might arrive between the setup of the signal handler and the start of select().
- **Time Resolution:**
  - select(): Uses a struct timeval to specify the timeout, which has microsecond resolution.
  - pselect(): Uses a struct timespec, which offers better resolution (nanoseconds) for the timeout.

Chương trình ví dụ: **ex_pselect.c**

poll()

- The functions select() and poll() are both used for monitoring multiple file descriptors to see if they are ready for I/O operations

```
#include <poll.h>
int poll (struct pollfd *fdarray, unsigned long nfds, int timeout);
```

poll() vs. select()

- **File Descriptor Limits:**
  - select() has a limitation on the number of file descriptors it can monitor, which is typically restricted by FD_SETSIZE (often 1024).
  - In contrast, poll() does not have this limitation and can handle more descriptors.
- **Data Structure:**
  - select() uses fixed-size bitmaps (file descriptor sets) to monitor the file descriptors. These sets are modified by the kernel, meaning the caller has to reset them before each call.
  - poll() uses a variable-sized array of structures (struct pollfd) where each element describes a file descriptor and the events of interest.
- **Efficiency:**
  - poll() scales better with large numbers of file descriptors because it does not need to copy and modify large bitmaps.
- **Timeout Precision:**
  - Both functions allow specifying a timeout, but poll() allows specifying time in milliseconds, whereas select()specifies it in seconds and microseconds.

Chương trình ví dụ: **ex_poll-select.c**