



# ADVANCED IO

---

Ngoc Nguyen Tran, SoICT, HUST

# Content

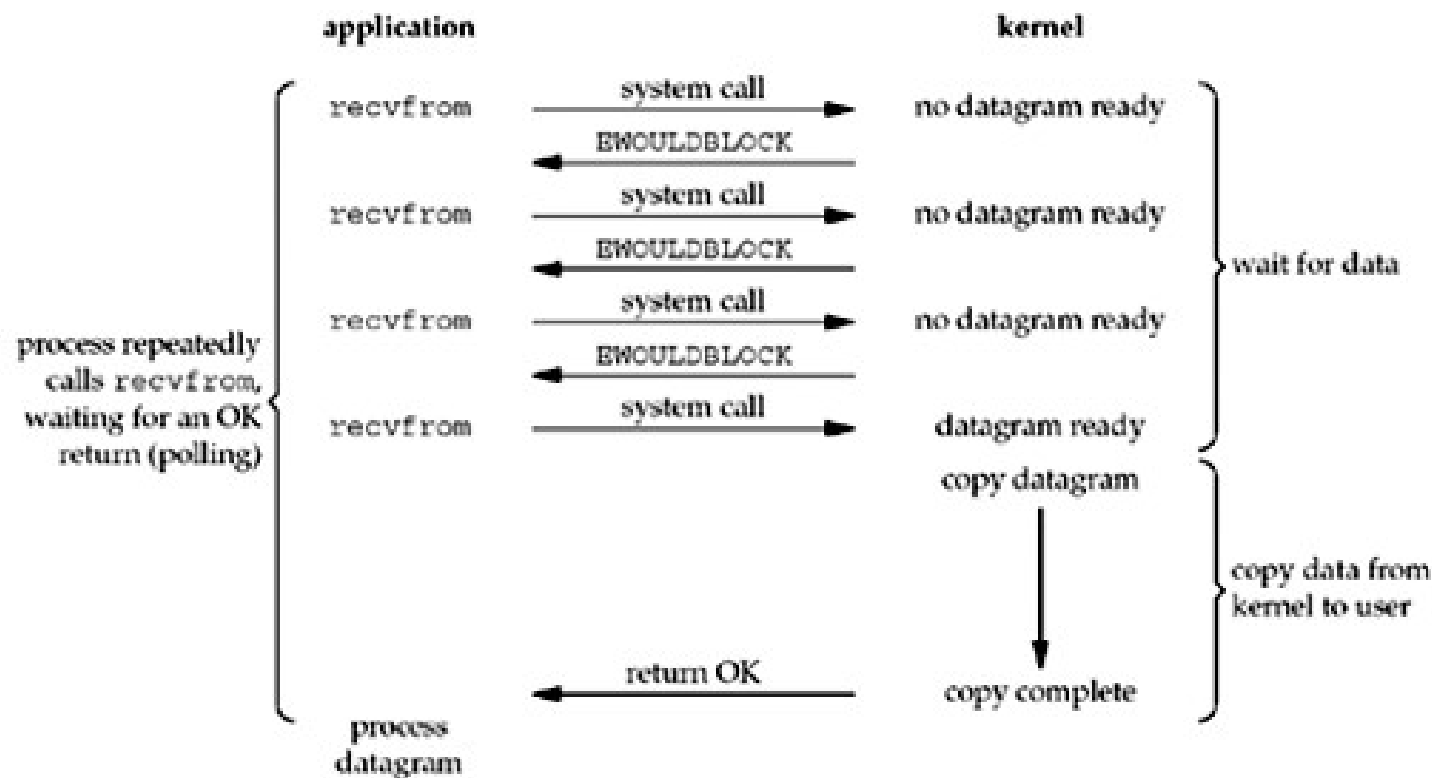
- Non-blocking IO
- Signal-driven I/O
- Some advanced I/O functions

# Blocking I/O

- By default, sockets are blocking: when a socket call cannot be completed immediately, the process is put to sleep, waiting for the condition to be true
- Input functions: `recv()`, `recvfrom()`, etc
  - Blocks until some data arrives
- Output function: `send()`, `sendto()`, etc
  - TCP: blocks until there is free space in sending buffer
  - UDP: block on some systems due to the buffering and flow control
- Accepting incoming connections: `accept()`
  - Blocks until a new connection is available
- Initiating outgoing connections: `connect()`
  - Blocks until the client receives the ACK of its SYN

# Non-blocking I/O Model

- Non-blocking I/O model: I/O function returns immediately
- If there is no data to return, so the kernel immediately returns an error of EWOULDBLOCK instead



# Non-blocking I/O: use `fcntl()`

```
#include <fcntl.h>
int fcntl(int fd, int cmd, ... /* int arg */ );
```

- Perform the file control operations described below on open files
- Parameter:
  - [IN] `fd`: the file descriptor
  - [IN] `cmd`: the control operation
  - The 3<sup>rd</sup> argument according to `cmd`
- Return:
  - Return -1 on error
  - Otherwise, return others depending on `cmd`

# Non-blocking I/O: use `fcntl()`

- Set non-blocking mode

```
int flags;
/* Get the file status flags and file access modes */
if ((flags = fcntl(fd, F_GETFL, 0)) < 0)
    perror("F_GETFL error");
/* Set a socket as nonblocking */
if (fcntl(fd, F_SETFL, flags | O_NONBLOCK) < 0)
    perror("F_SETFL error");
```

- Turn off non-blocking mode

```
int flags;
/* Get the file status flags and file access modes */
if ((flags = fcntl(fd, F_GETFL, 0)) < 0)
    perror("F_GETFL error");
/* Turn off non-blocking mode on socket */
if (fcntl(fd, F_SETFL, flags & ~O_NONBLOCK) < 0)
    perror("F_SETFL error");
```

# Non-blocking I/O: use ioctl()

```
#include <sys/ioctl.h>
int ioctl(int fd, int request, ... /* void arg */ );
```

- Manipulates the underlying device parameters of special files and control operating characteristics of files
- Parameters
  - [IN]fd: the file descriptor
  - [IN]request: device-dependent request code
  - The 3rd argument according to request
- Return:
  - 0 if succeed
  - -1 if error

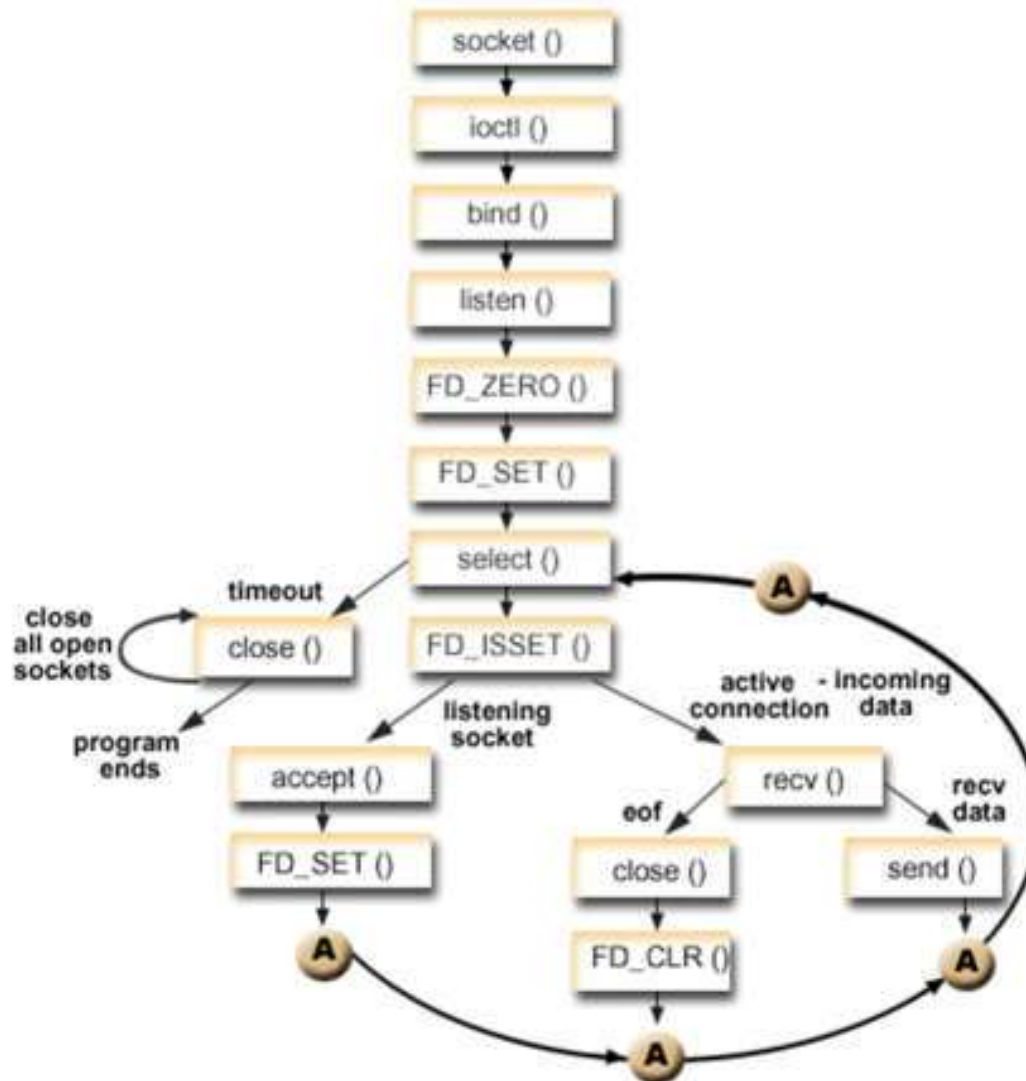
```
int on = 1;
/* Set a socket as nonblocking */
ioctl(fd, FIONBIO, (char *)&on);
on = 0;
/* Turn off non-blocking mode on socket */
ioctl(fd, FIONBIO, (char *)&on);
```

# Non-blocking I/O: process return value

```
//call I/O functions
if(ret < 0){
    //Error on I/O operation
    if (errno != EWOULDBLOCK)
    {
    }
    else if(ret == 0){
        //Connection terminated normally
    }
    else if{
        //I/O operation is successful
    }
}
```



# Non-blocking I/O: Example





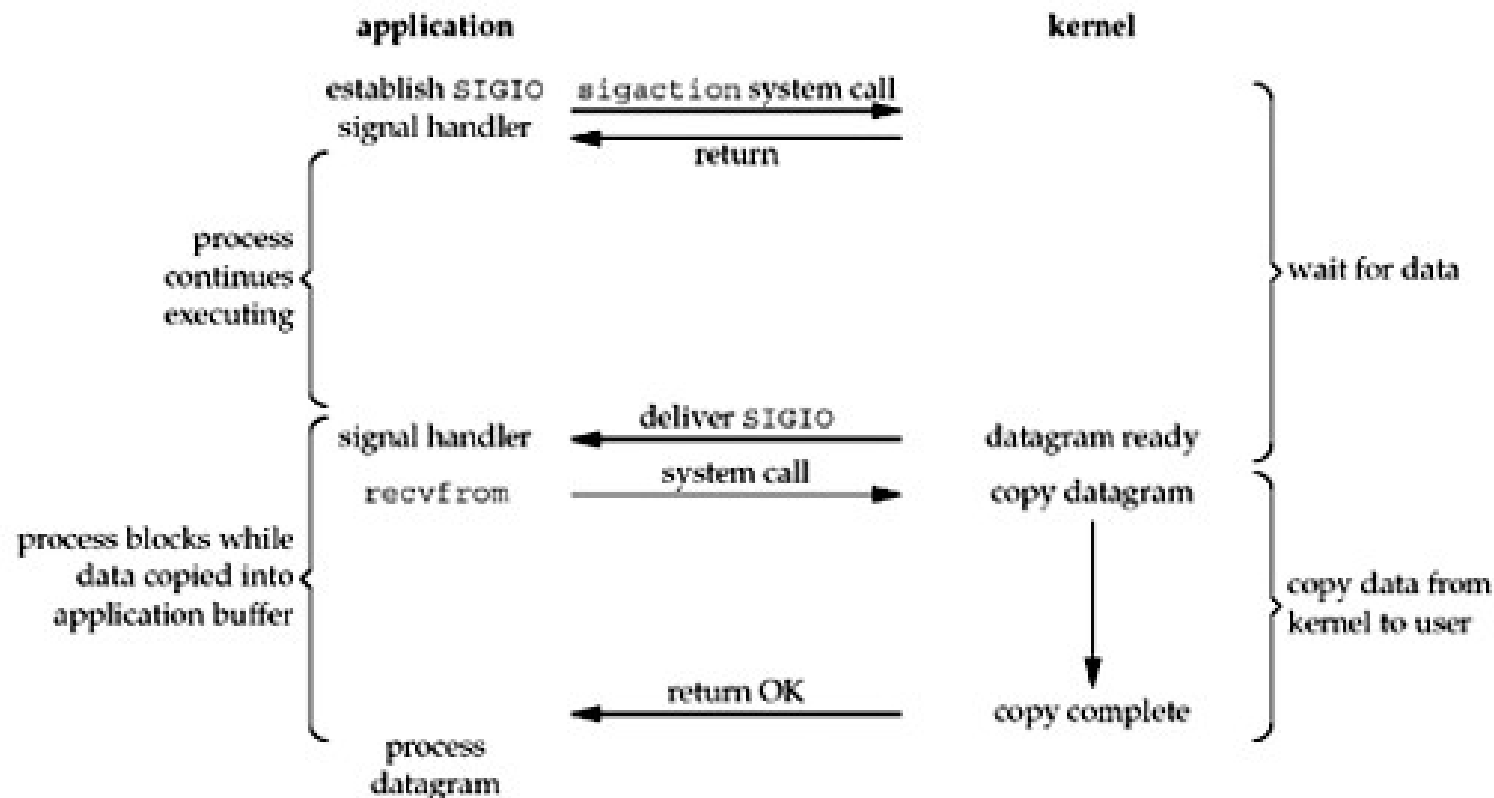
# SIGNAL-DRIVEN I/O

---

Ngoc Nguyen Tran, SoICT, HUST

# Signal-driven I/O

- Use signals, telling the kernel to notify app with the SIGIO signal when the descriptor is ready



# Signal-driven I/O: 3 steps

1. A signal handler must be established for the SIGIO signal.
2. Assign a process to receive the SIGIO signal  
`fcntl(fd, F_SETOWN, process_id)`
3. Enable signal-driven I/O on socket
  - Turn on asynchronous mode
  - Turn on non-blocking mode
- The importance is determining what conditions cause SIGIO to be generated for the socket owner

# Signal-driven I/O: use `fcntl()`

- Enable signal-driven I/O on socket

```
int flags;
/* Get the file status flags and file access modes */
if ( (flags = fcntl (fd, F_GETFL, 0)) < 0)
    err_sys("F_GETFL error");
/* Set a socket as nonblocking */
if (fcntl(fd, F_SETFL, flags | O_ASYNC | O_NONBLOCK) < 0)
    err_sys("F_SETFL error");
```

- Turn off asynchronous I/O mode

```
int flags;
/* Get the file status flags and file access modes */
if ( (flags = fcntl (fd, F_GETFL, 0)) < 0)
    err_sys("F_GETFL error");
/* Turn off non-blocking mode on socket */
if (fcntl(fd, F_SETFL, flags & ~O_ASYNC & ~O_NONBLOCK) < 0)
    err_sys("F_SETFL error");
```

# Signal-driven I/O: use `ioctl()`

```
int on = 1;
/* Set a socket as nonblocking */
ioctl(fd, FIOASYNC, (char *)&on);
ioctl(fd, FIONBIO, (char *)&on)
on = 0;
/* Turn off non-blocking mode on socket */
ioctl(fd, FIOASYNC, (char *)&on);
ioctl(fd, FIONBIO, (char *)&on)
```

# SIGIO on sockets

- UDP socket: The signal SIGIO is generated whenever
  - A datagram arrives for the socket
  - An asynchronous error occurs on the socket
- TCP socket: the following conditions all cause SIGIO to be generated(very complex)
  - A connection request has completed on a listening socket
  - A disconnect request has been initiated
  - A disconnect request has completed
  - Half of a connection has been shut down
  - Data has arrived on a socket
  - Data has been sent from a socket (i.e., the output buffer has free space)
  - An asynchronous error occurred

# Example: signal-driven I/O on UDP socket

- See source code