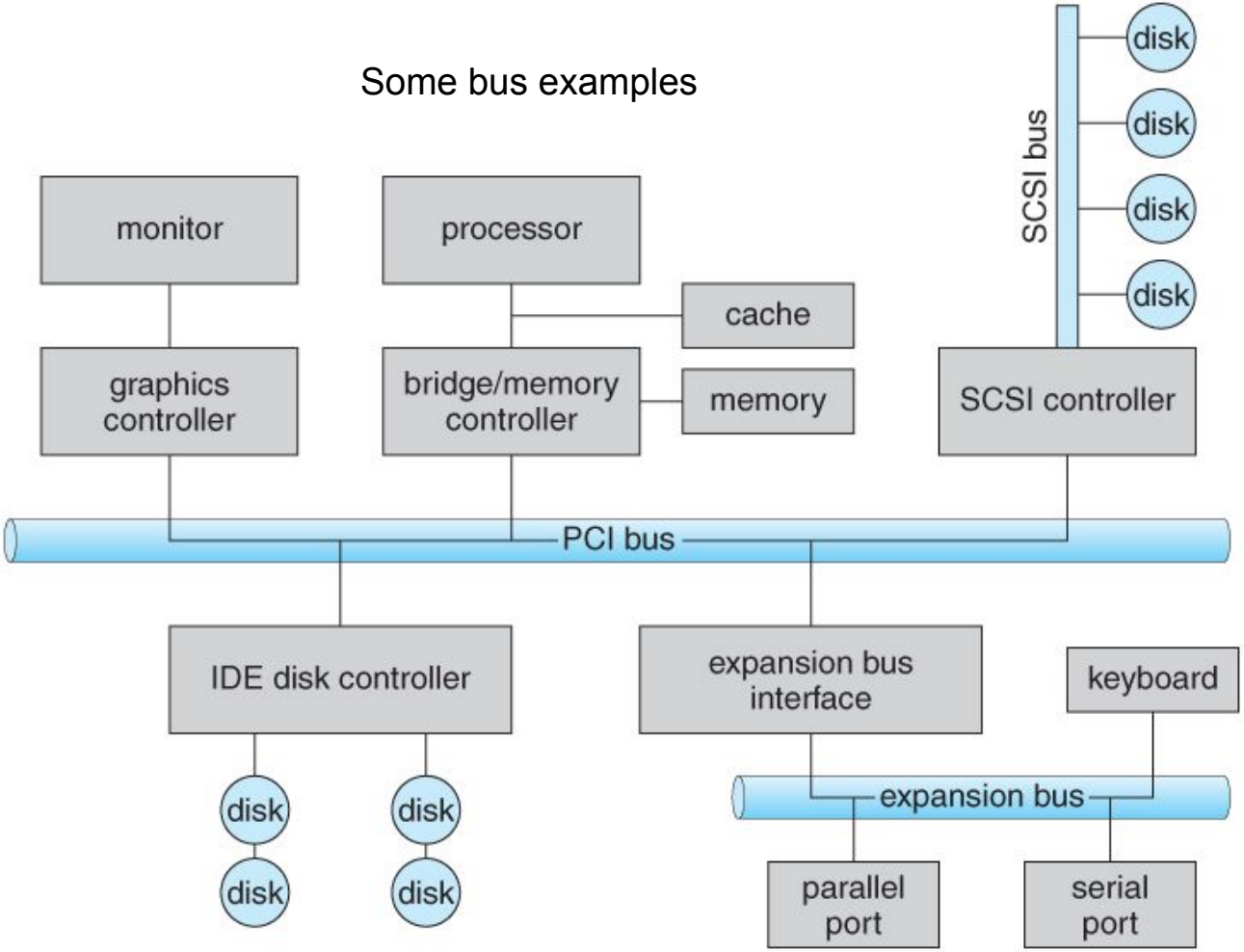
Storage,
I/O Systems

- Let us consider the range of devices on a modern computer,
  - mice,
  - keyboards,
  - disk drives (CD/DVD-ROM),
  - display adapters (VGA-HDMI),
  - USB devices (different USB protocols),
  - network connections (ethernet cable),
  - audio I/O (headphones and microphones),
  - printers,
  - many special-purpose peripherals.

1- I/O Hardware

- I/O devices can be roughly categorized as **storage**, **communications**, **user-interface**, and others.
- Devices communicate with the computer via signals sent over wires or through the air (wireless).
- Devices connect with the computer via ***ports***, e.g. a serial or parallel port.
- A common set of wires connecting multiple devices is termed a ***bus.***

Some bus examples



- **SCSI**: Small Computer System Interface
- **PCI**: Peripheral Component Interconnect

1) One way of communicating with devices is through ***registers*** associated with each port. Registers may be one to four bytes in size, and may typically include (a subset of) the following four:

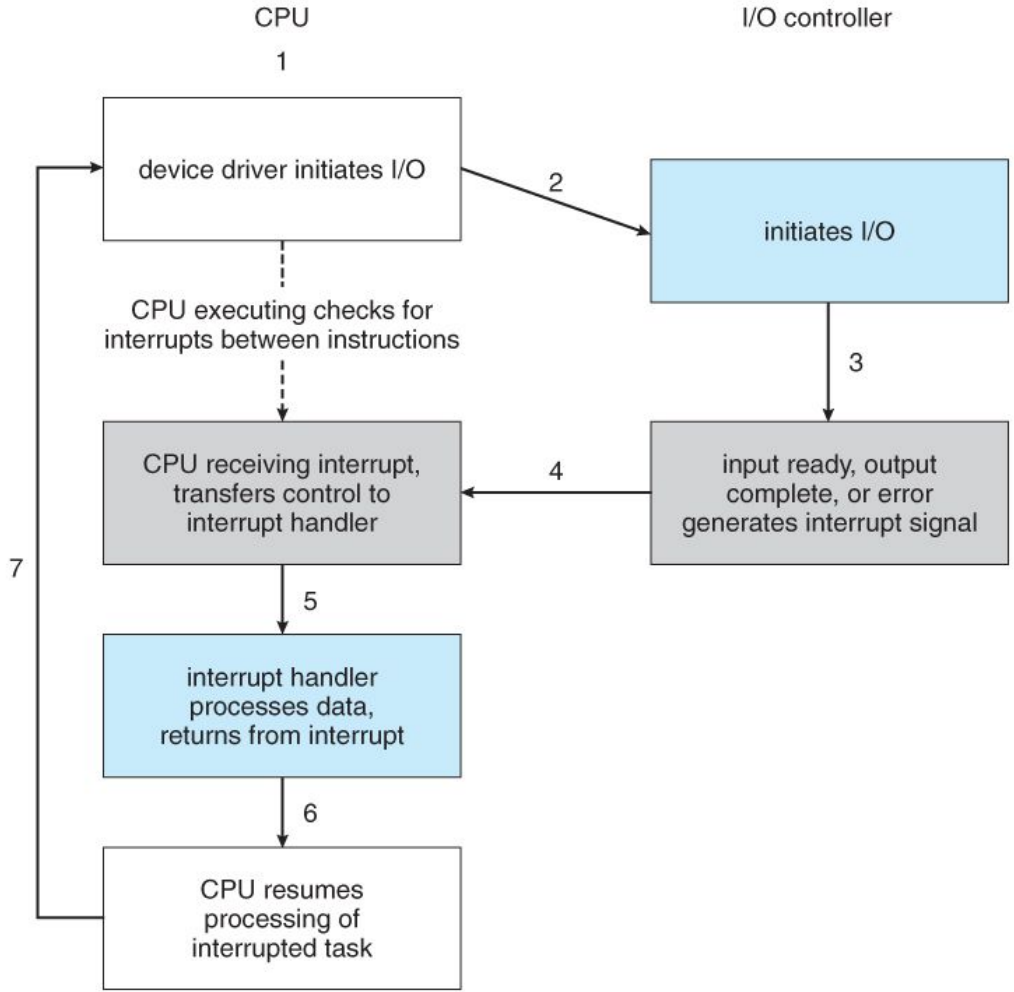
- **data-in register:** read by host to get input
- **data-out register:** written by host to send output
- **status register:** status informative register such as idle, ready for input, busy, error etc.
- **control register:** controlled by host to issue commands or to change settings such as parity checker configuration, word length etc.

2) Another technique for communicating with devices is ***memory-mapped I/O.***

- In this case a certain portion of the processor's address space is mapped to the device, and communications occur by reading and writing directly to/from those memory areas.
- Memory-mapped I/O is suitable for devices which must move large quantities of data quickly, such as graphics cards.
- **Polling** and **Interrupt**

Polling: Polling is a protocol in which CPU steadily checks whether the device needs attention. Wherever device tells process unit that it desires hardware processing, in polling process unit keeps asking the I/O device whether or not it desires CPU processing. The CPU ceaselessly check every and each device hooked up thereto for sleuthing whether or not any device desires hardware attention.

Interrupt: Interrupt is a hardware mechanism in which, the device notices the CPU that it requires its attention. Interrupt can take place at any time. So when CPU gets an interrupt signal trough the indication interrupt-request line, CPU stops the current process and respond to the interrupt by passing the control to interrupt handler which services device.



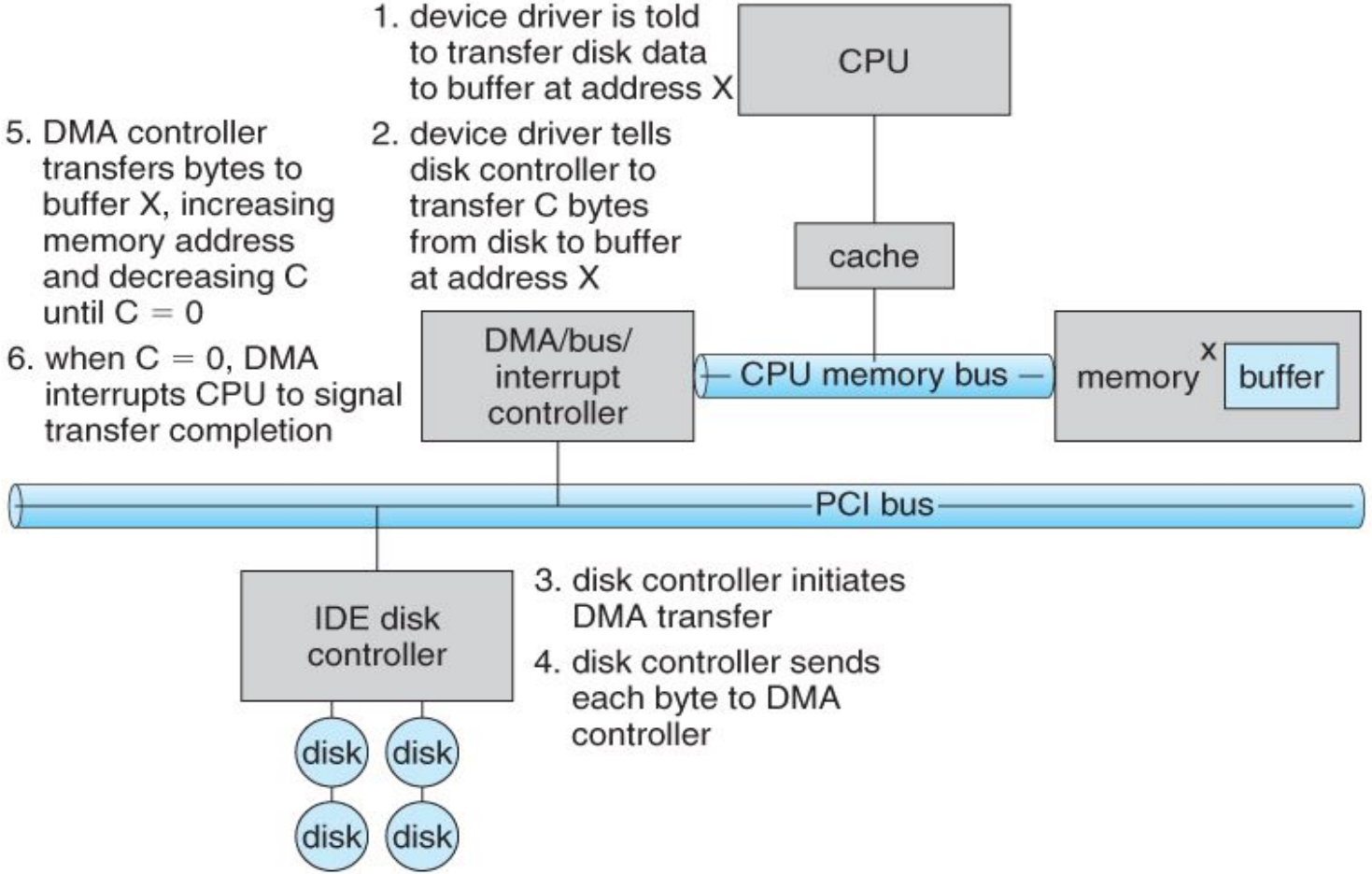
Interrupt-driven I/O cycle.

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Intel Pentium processor event-vector table.

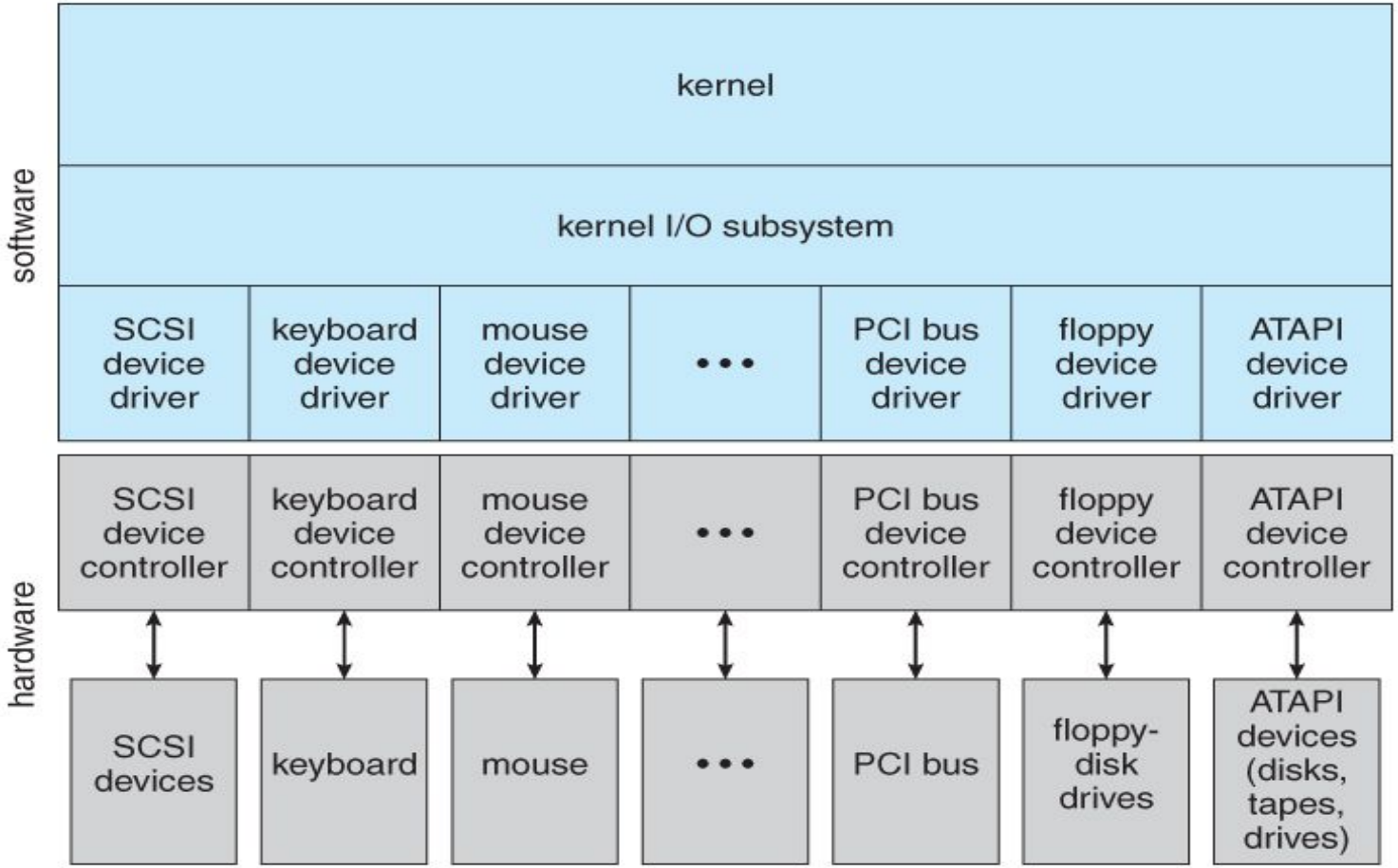
I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

Device I/O port locations on PCs.



Steps in a DMA transfer.

Application I/O Interface



A kernel I/O structure

path: /dev

IOCTL: The **ioctl**() system call manipulates the underlying device parameters of special files. In particular, many operating characteristics of character special files (e.g., terminals) may be controlled with **ioctl**() requests.

- Corresponding Library **-> #include <sys/ioctl.h>**
- function skeleton -> **int ioctl(int** *fd*, **unsigned long** *request*, **...)**
    - fd is a opened file.
    - The second argument is a device-dependent request code.
    - The third argument is an untyped pointer to memory.

EXAMPLE:

https://embetronicx.com/tutorials/linux/device-drivers/ioctl-tutorial-in-linux/#Example-3

fseek() -> is used to move file pointer associated with a given file to a specific position.

int fseek(FILE *pointer, long int offset, int position)

- **pointer:** pointer to a FILE object that identifies the stream.
- **offset:** number of bytes to offset from position
- **position:** position from where offset is added.
- **returns with** zero if successful, or else it returns a non-zero value
  - **SEEK_END:** It denotes end of the file.
  - **SEEK_SET:** It denotes starting of the file.
  - **SEEK_CUR:** It denotes file pointer’s current position.

```c
#include <stdio.h>

int main()
{
    FILE *fp;
    fp = fopen("test.txt", "r");

    // Moving pointer to end
    fseek(fp, 0, SEEK_END);

    // Printing position of pointer
    printf("%ld", ftell(fp));

    return 0;
}
```

The file test.txt contains the following text:

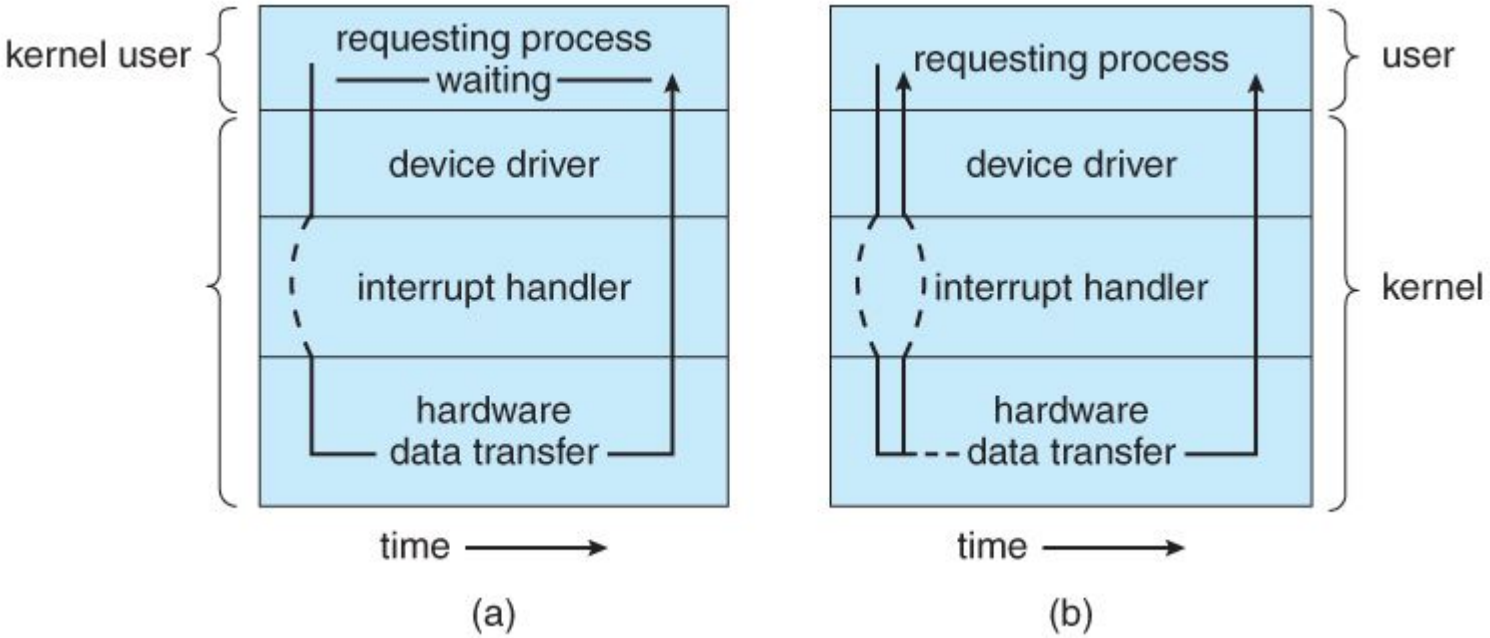
```
"Someone over there is calling you.
we are going for work.
take care of yourself."
```

Output:

```
81
```

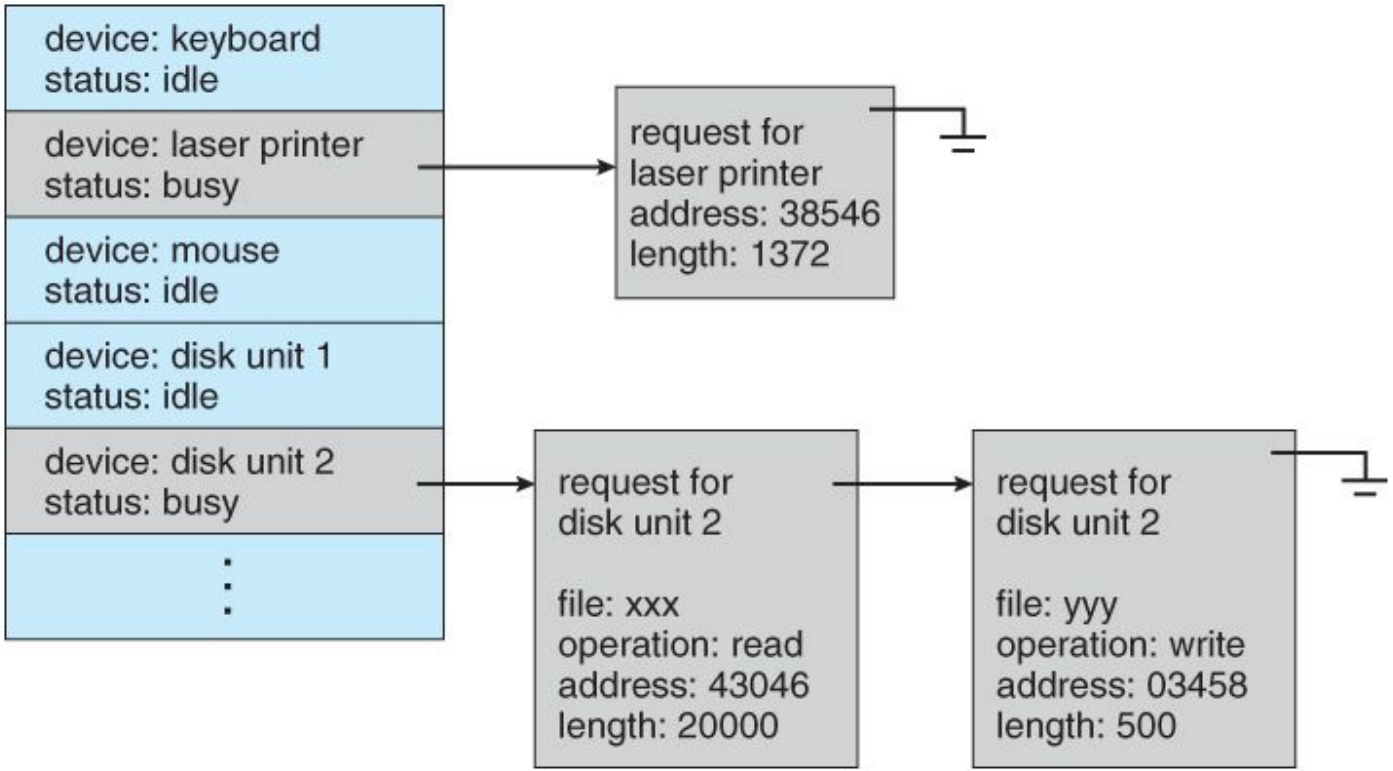
Network Devices

- Because network access is inherently different from local disk access, most systems provide a separate interface for network devices.
- One common and popular interface is the ***socket*** interface, which acts like a cable or pipeline connecting two networked entities. Data can be put into the socket at one end, and read out sequentially at the other end. Sockets are normally full-duplex, allowing for bi-directional data transfer.
- The select( ) system call allows servers (or other applications) to identify sockets which have data waiting, without having to poll all available sockets.
- For example, http and https domains use 80 port and corresponding socket.

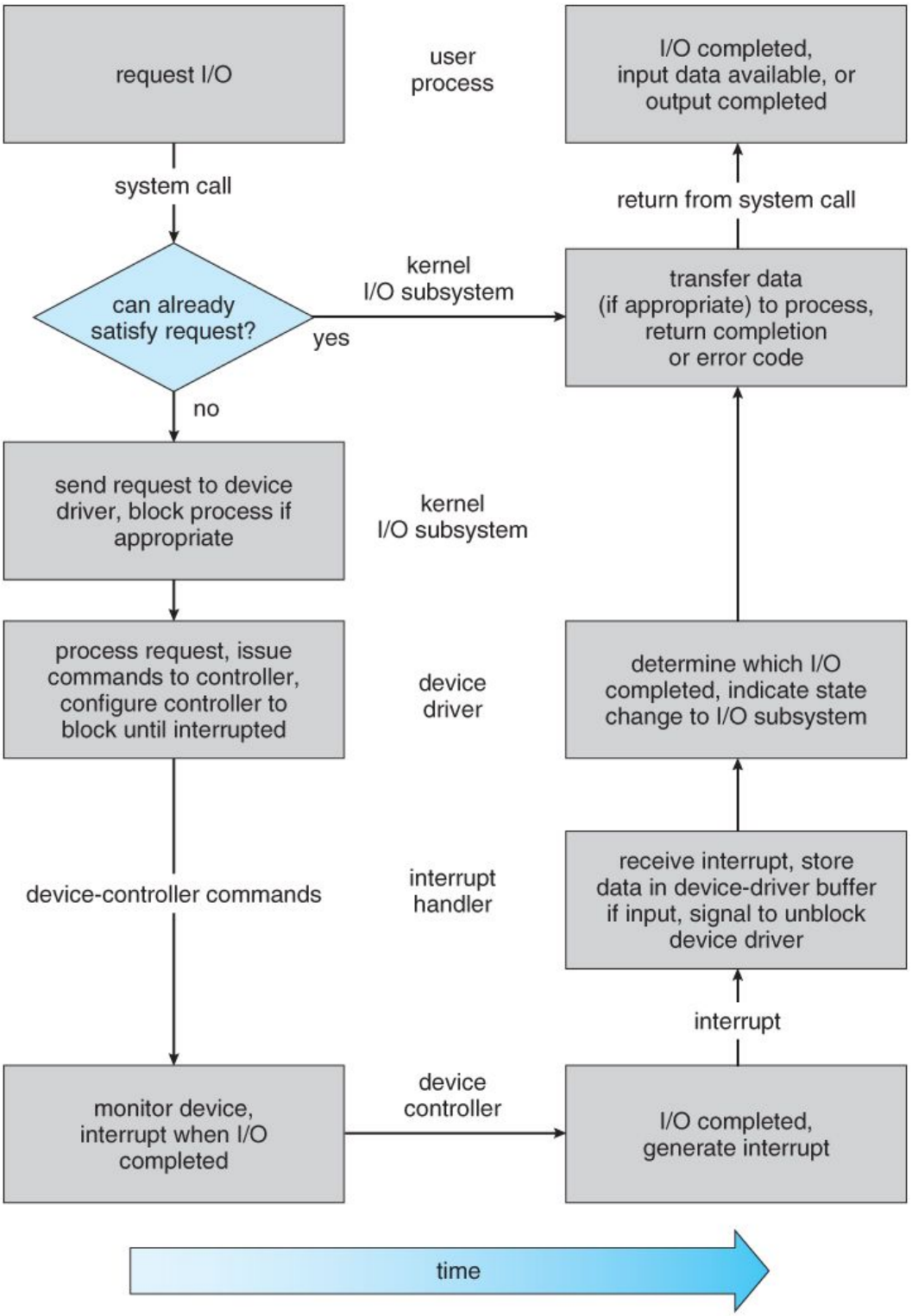


Two I/O methods: (a) synchronous and (b) asynchronous.

Kernel I/O Subsystem



1. **I/O Scheduling:** On systems with many devices, separate request queues are often kept for each device as previous figure.
2. **Buffering of I/O operations:** Speed differences, data transfer size differences, To support copy semantics.
3. **Caching:** Similar to the buffering.
4. **Spooling and Device Reservation :** A spool (Simultaneous Peripheral Operations On-Line) buffers data for (peripheral) devices such as printers that cannot support interleaved data streams.
5. **Error Handling** (as we did in our codes)
6. **I/O Protection:** The I/O system must protect against either accidental or deliberate erroneous I/O.



The life cycle of an I/O request