Device Management

Device Management

- The two main jobs of a computer are **I/O** and **computing**.
- In many cases, the main job is I/O, and the computing or processing is merely incidental.
- For example, when we browse a web page or edit a file, our immediate interest is to read or enter some information, not to compute an answer.
- The role of the operating system in computer I/O is to manage and control I/O operations and I/O devices.
- The control of devices connected to the computer is a major concern of operating-system designers. Because I/O devices vary so widely in their function and speed (consider a mouse, a hard disk, a flash drive, and a tape robot), varied methods are needed to control them.

Device Management

- Device management implies the management of the **I/O devices** such as a keyboard, magnetic tape, disk, printer, microphone, USB ports, scanner, camcorder etc.
- OS manages **communication** with the devices through their **respective drivers**.
- OS component provides a uniform interface to access devices of varied physical attributes.

Device Management

I/O devices may be divided into three categories:

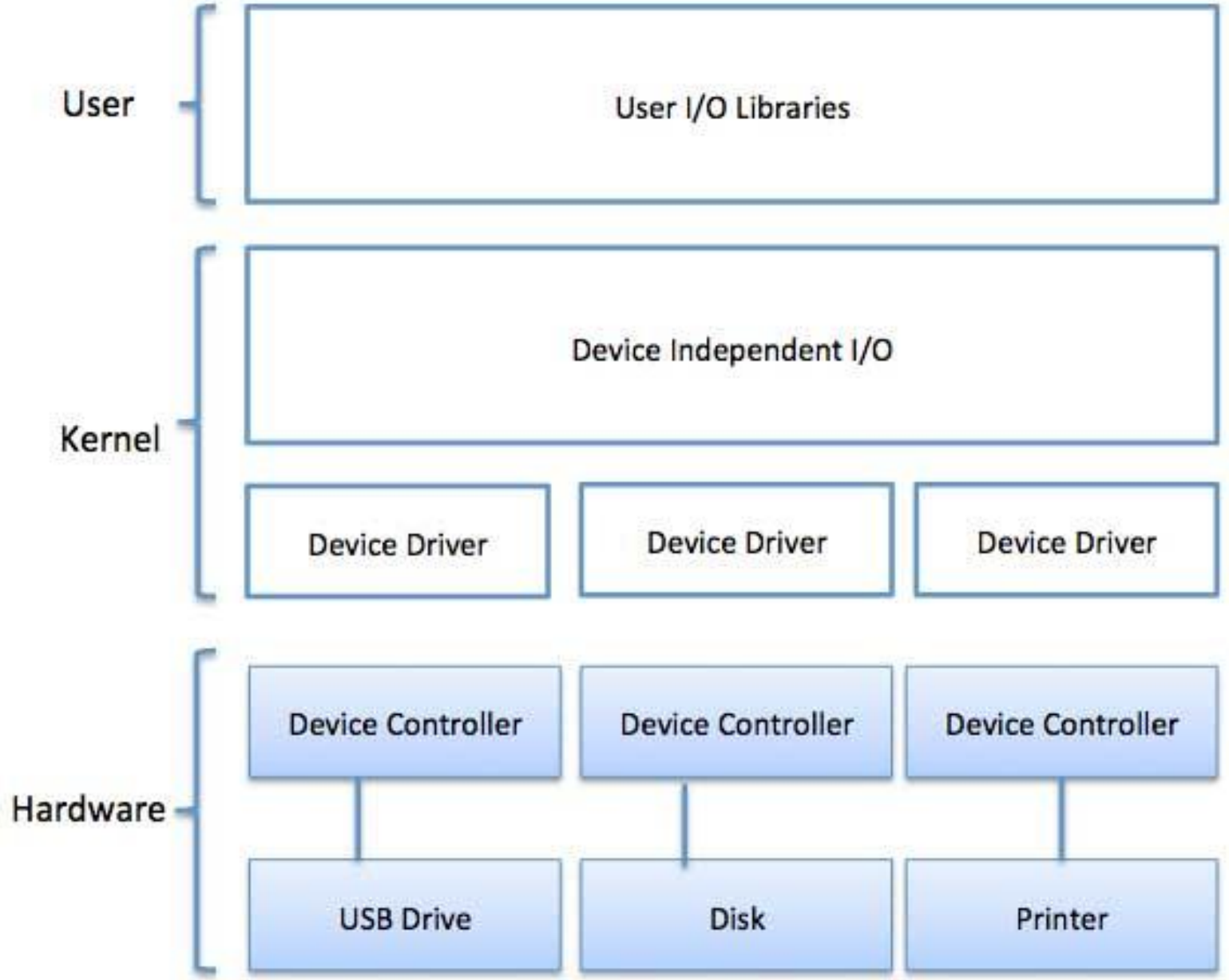
- **Block device:** it stores information in fixed-size block, each one with its own address. For example, disks.
- **Character device:** delivers or accepts a **stream of characters**. The individual characters are not addressable. For example printers, keyboards etc.
- **Network device:** For transmitting data packets.

I/O Software

- **I/O Software** is used for interaction with I/O devices like mouse, keyboard, USB devices, printers, etc.
- A key concept in the design of I/O software is that it should be **device independent** where it should be possible to write programs that can access any I/O device without having to specify the device in advance.
- For example, a program that reads a file as input should be able to read a file on a floppy disk, on a hard disk, or on a CD-ROM, without having to modify the program for each different device.

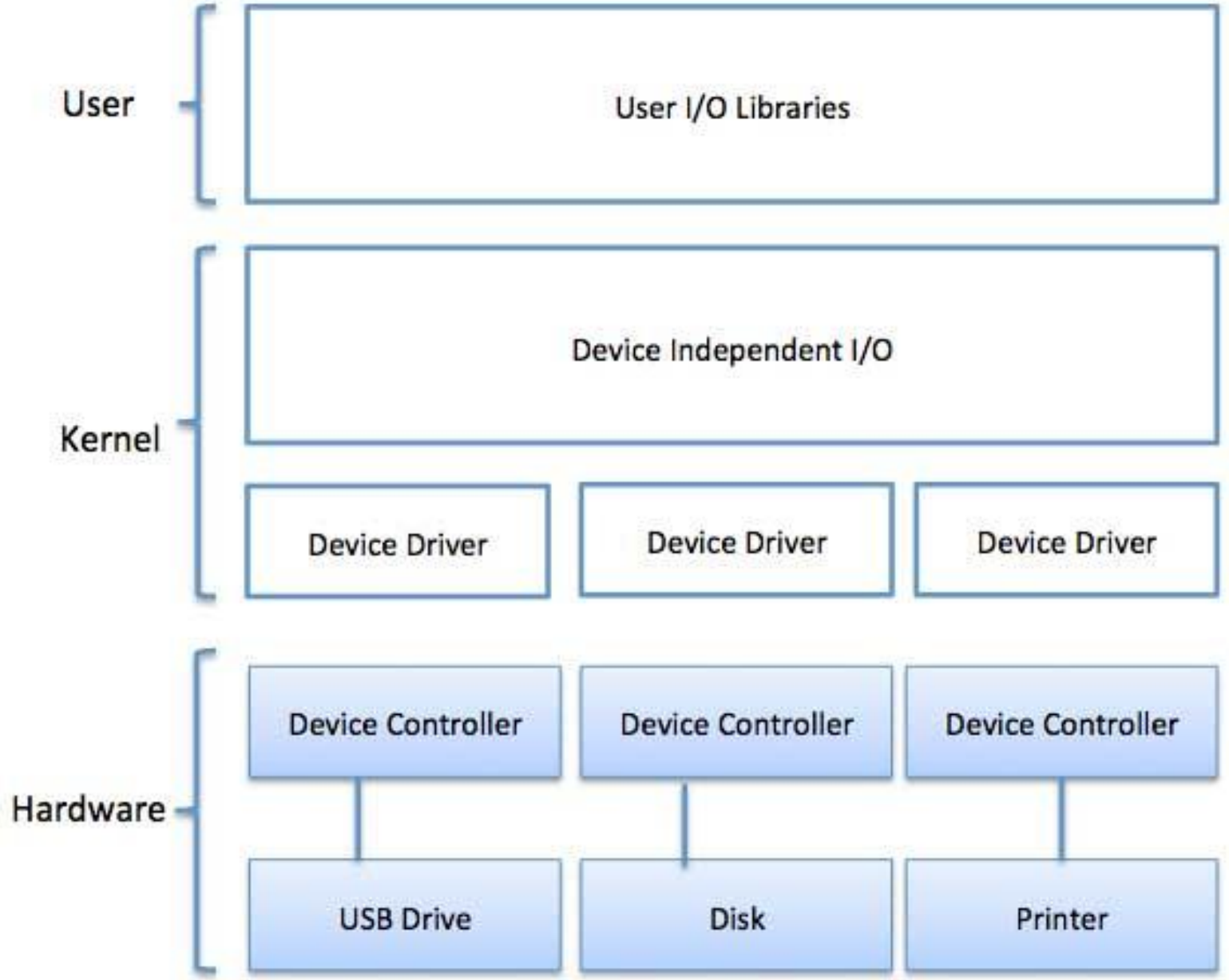
I/O Software

- **I/O Software** is used for interaction with I/O devices like mouse, keyboard, USB devices, printers, etc.
- I/O software are organized in following ways:
- **User Level Libraries:** This provides simple interface to the user program to perform input and output. For example, stdio is a library provided by C and C++ programming languages.



I/O Software

- **Kernel Level Modules:** This provides device driver to interact with the device controller and device independent I/O modules used by the device drivers.
- **Hardware:** This layer includes actual hardware and **hardware controller** which interact with the device drivers and makes hardware alive.



I/O Software

Device Drivers:

- Device drivers are software modules that can be plugged into an OS to handle a particular device.
- Operating System takes help from device drivers to handle all I/O devices.
- Device drivers encapsulate **device-dependent code** and implement a standard interface in such a way that code contains **device-specific register reads/writes**.
- A device driver performs the following jobs −

  To accept request from the device independent software above to it.

  Interact with the device controller to take and give I/O and perform required error handling.

  Making sure that the request is executed successfully.

- How a device driver handles a request is as follows: Suppose a request comes to read a block N. If the driver is idle at the time a request arrives, it starts carrying out the request immediately. Otherwise, if the driver is already busy with some other request, it places the **new request** in **the queue of pending requests**.

I/O Software

Interrupt handlers:

- An interrupt handler, also known as an interrupt service routine or ISR, is a piece of software or more specifically a callback function in an operating system.
- When the interrupt happens, the interrupt procedure does whatever it has to in order to handle the interrupt, updates data structures and wakes up process that was waiting for an interrupt to happen.
- The interrupt mechanism accepts an address – a number that selects a specific interrupt handling routine/function from a small set.
- In most architectures, this address is an offset stored in a table called the interrupt vector table. This vector contains the memory addresses of specialized interrupt handlers.

I/O Software

Device-Independent I/O Software:

- The basic function of the device-independent software is to perform the I/O functions that are **common to all devices** and to provide a **uniform interface to the user-level software**.
- Though it is difficult to write completely device independent software but we can write some modules which are common among all the devices.
- Following is a list of **functions** of device-independent I/O Software –

 Uniform interfacing for device drivers

 Device naming - Mnemonic names mapped to Major and Minor device numbers

 Device protection

 Providing a device-**independent block size**

 Buffering because data coming off a device cannot be stored in final destination.

 Storage allocation on block devices

 Allocation and releasing dedicated devices

 Error Reporting

I/O Software

User-Space I/O Software:

- These are the **libraries** which provide richer and simplified interface to access the functionality of the kernel or ultimately interactive with the device drivers.
- Most of the user-level I/O software consists of library procedures with some exception like **spooling system** which is a way of dealing with dedicated I/O devices in a multiprogramming system.
- I/O Libraries (e.g., stdio) are in user-space to provide an interface to the OS resident device-independent I/O SW.
- For example putchar(), getchar(), printf() and scanf() are example of user level I/O library stdio available in C programming.

I/O Software

Kernel I/O Subsystem:

- Kernel I/O Subsystem is responsible to provide many services related to I/O. Following are some of the services provided.

✔ **Scheduling** − Kernel schedules a set of I/O requests to determine a good order in which to execute them. When an application issues a blocking I/O system call, the request is placed on the queue for that device. The Kernel I/O scheduler rearranges the order of the queue to improve the overall system efficiency and the average response time experienced by the applications.

✔ **Buffering** − Kernel I/O Subsystem maintains a memory area known as buffer that stores data while they are transferred between two devices or between a device with an application operation. Buffering is done to cope with a speed mismatch between the producer and consumer of a data stream or to adapt between devices that have different data transfer sizes.

I/O Software

Kernel I/O Subsystem:

- Kernel I/O Subsystem is responsible to provide many services related to I/O. Following are some of the services provided.

✔ **Caching** − Kernel maintains cache memory which is region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original.

✔ **Spooling** and Device Reservation − A spool is a buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams. The spooling system copies the queued spool files to the printer one at a time. In some operating systems, spooling is managed by a system daemon process. In other operating systems, it is handled by an in kernel thread.

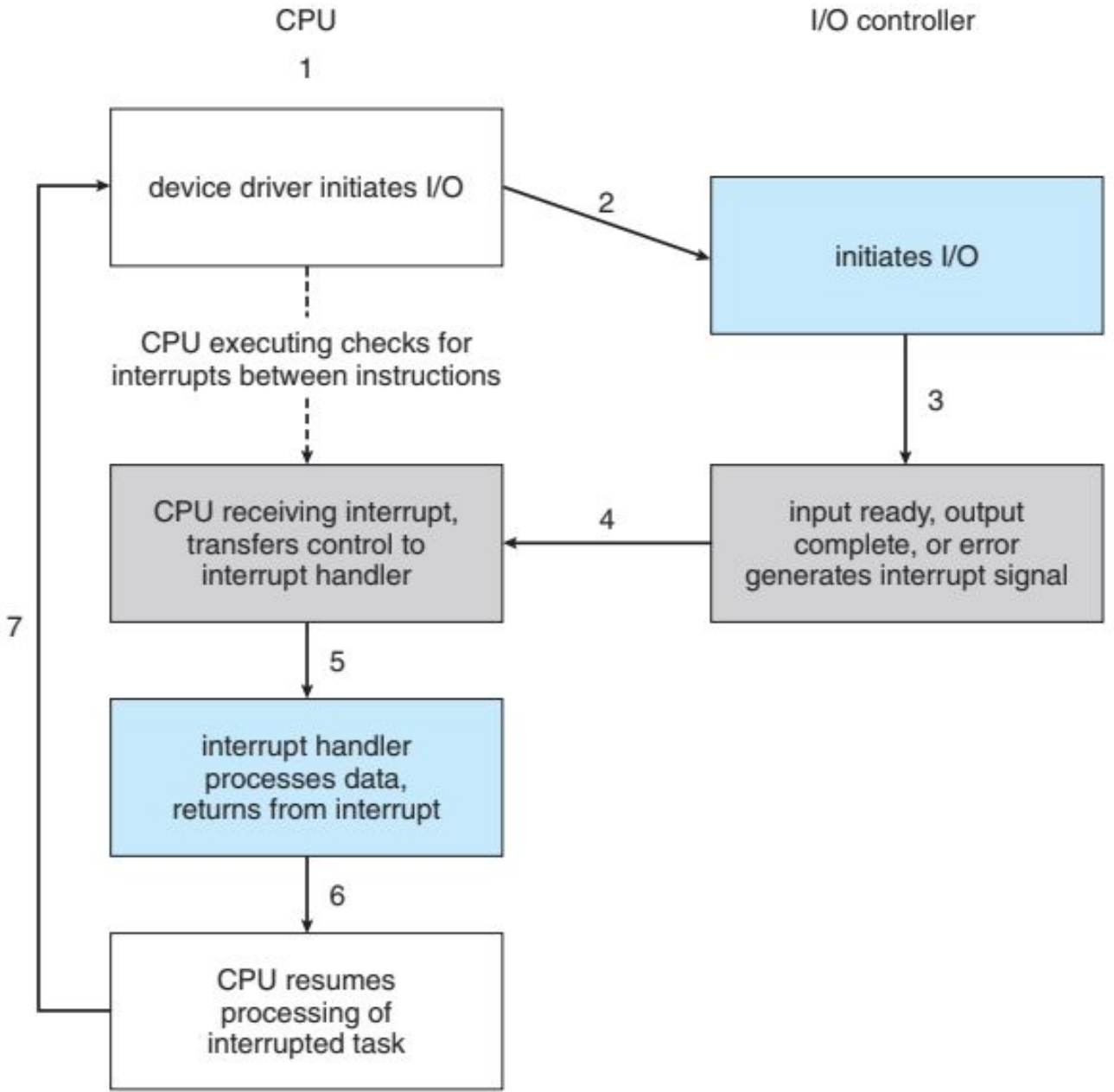
✔ **Error Handling** − An operating system that uses protected memory can guard against many kinds of hardware and application errors.

Interrupt Service Routine or Interrupt Handler

- The CPU hardware has a wire called the **interrupt-request line** that the CPU senses after executing every instruction.
- When the CPU detects that a controller has asserted a signal on the interrupt-request line, the CPU performs a state save and jumps to the interrupt-handler routine at a fixed address in memory.
- The interrupt handler determines the cause of the interrupt, performs the necessary processing, performs a state restore, and executes a return from interrupt instruction to return the CPU to the execution state prior to the interrupt.

Interrupt Service Routine or Interrupt Handler

- The device controller raises an interrupt by asserting a signal on the interrupt request line.
- The CPU catches the interrupt and dispatches it to the interrupt handler, and the handler clears the interrupt by servicing the device.



Interrupt Service Routine or Interrupt Handler

Features of a good interrupt handling:

- We need the ability to defer interrupt handling during **critical processing**.
- We need an efficient way to dispatch to the **proper interrupt handler** for a device without first polling all the devices to see which one raised the interrupt.
- We need multilevel interrupts, so that the operating system can distinguish between **high- and low-priority interrupts** and can respond with the appropriate degree of urgency when there are multiple concurrent interrupts.
- We need a way for an instruction to get the operating system's attention directly (separately from I/O requests), for activities such as page faults and errors such as division by zero. As we shall see, this task is accomplished by **traps**.