Ollscoil na Gaillimhe
University of Galway

Dr Takfarinas Saber
takfarinas.saber@universityofgalway.ie

CT213 Computing System & Organisation

Lecture 8: Device Management

Content

- Device management
- Device Communication Approaches
- Buffering

Device Management

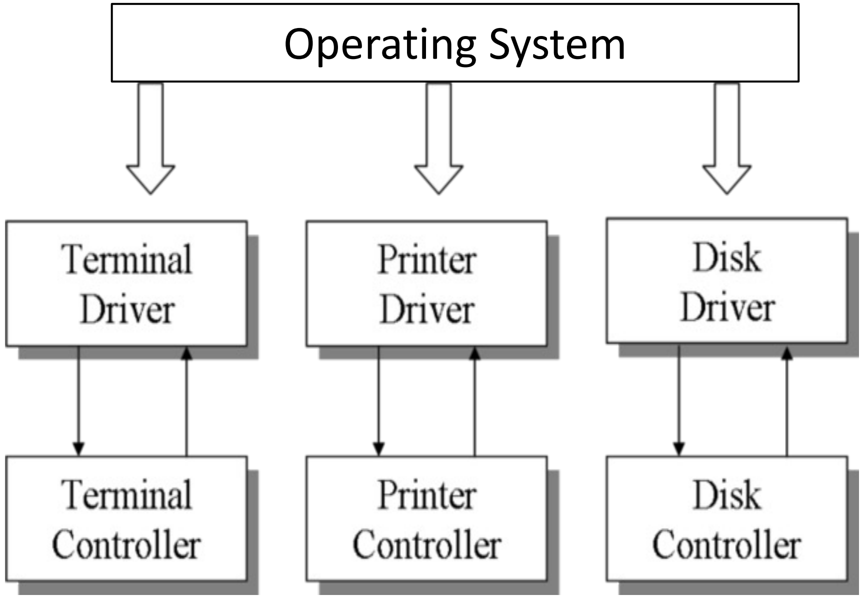
- Device management is the process of managing the **implementation**, **operation** and **maintenance** of physical and/or virtual devices.

- It is a broad term that includes **various administrative tools and processes** for the maintenance and upkeep of a computing, network, mobile and/or virtual device.

- The status of any computing device (internal or external), may be either **free** or **busy**.
  - If a device requested by a process is free at a specific instant of time, the operating system allocates it to the process.

Device Management

- The Operating System manages the devices with the help of:
  - **Device controllers:** hardware components that contain some buffer registers to store the data temporarily.
    - E.g., disk controller, printer controller and a terminal controller
  - **Device drivers:** software programs that are used by an operating system to control the functioning of various devices in a uniform manner.



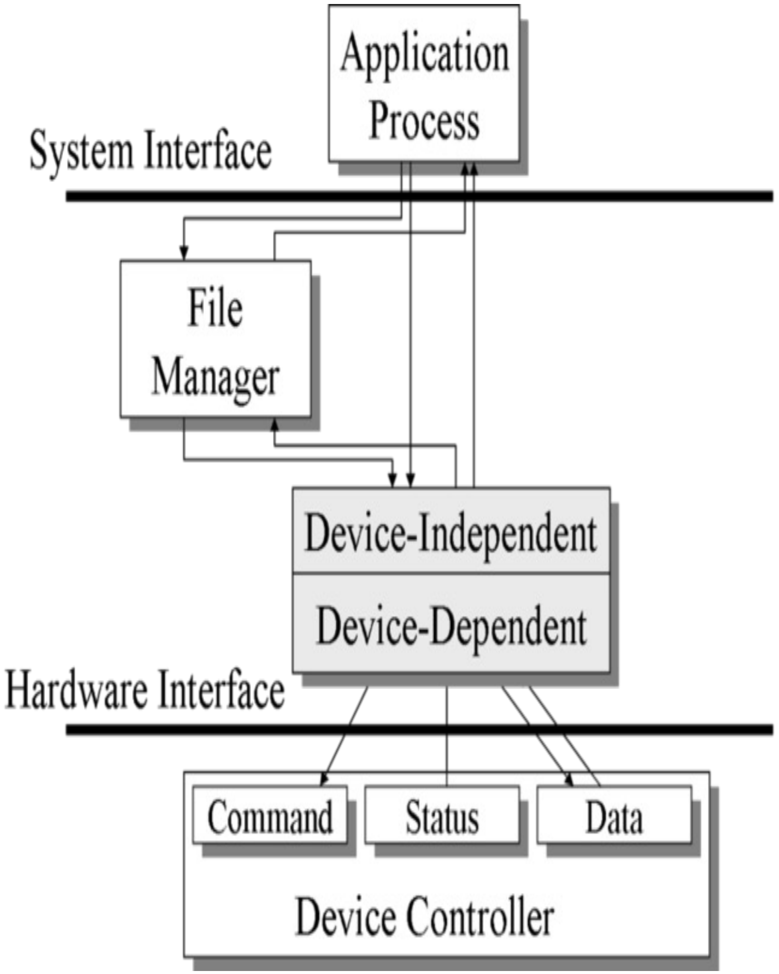
Device Management

- The device controller used in a device management operation includes three different registers: **command**, **status**, and **data**.

- The other major responsibility of the device management function is to implement **Application Programming Interfaces (APIs)**.

- Each device controller is **specific to a particular device**
    - ➢ the device driver implementation will be device specific

- Why?
    - To provide correct commands to the controller
    - To interpret the Controller Status Register (CSR) correctly
    - To transfer data to and from device controller data registers as required for correct device operation



Device Communication Approaches

- A computer must have a way of detecting the arrival of **any type of input**

- There are various ways to enable I/O devices to communicate with the processor:
  - Polling
  - Interrupts
  - Direct I/O
  - Memory Mapped I/O

Polling

- Implementation
  - ***Periodically checking status*** of the device to see if it is time for the next I/O operation
  - I/O device simply puts the information in a Status register, and the processor must come and get the information.

- Efficiency
  - ***Simplest*** way for an I/O device to communicate with the processor.
  - ***Inefficient method***: most of the time, devices will not require attention and when one does it will have to wait until it is next interrogated by the polling program.
    - ➢ Much of the processor's time is wasted on unnecessary polls.

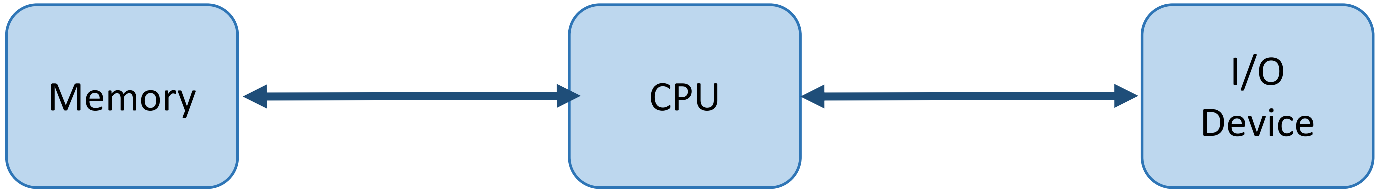
Interrupts

- Implementation
  - A device controller puts out an ***interrupt signal*** when it needs CPU's attention
  - When CPU receives an interrupt, it ***saves its current state*** and invokes the appropriate interrupt handler using the interrupt vector (addresses of OS routines to handle various events).
  - When the interrupting device has been dealt with, the ***CPU continues with its original task*** as if it had never been interrupted.

- Efficiency
  - Interrupts allow the processor to deal with events that can happen at any time.
  - Interrupts remove the need for the CPU to constantly check the Controller Status register.

Direct I/O

- Implementation
  - Uses software which explicitly transfers data to/from the **controller's data registers**
    - Separate I/O and memory address spaces.
    - The control indicates whether address information is for memory or I/O.



- Efficiency
  - Reduced CPU utilisation (no caches or buffers)

Memory Mapped I/O

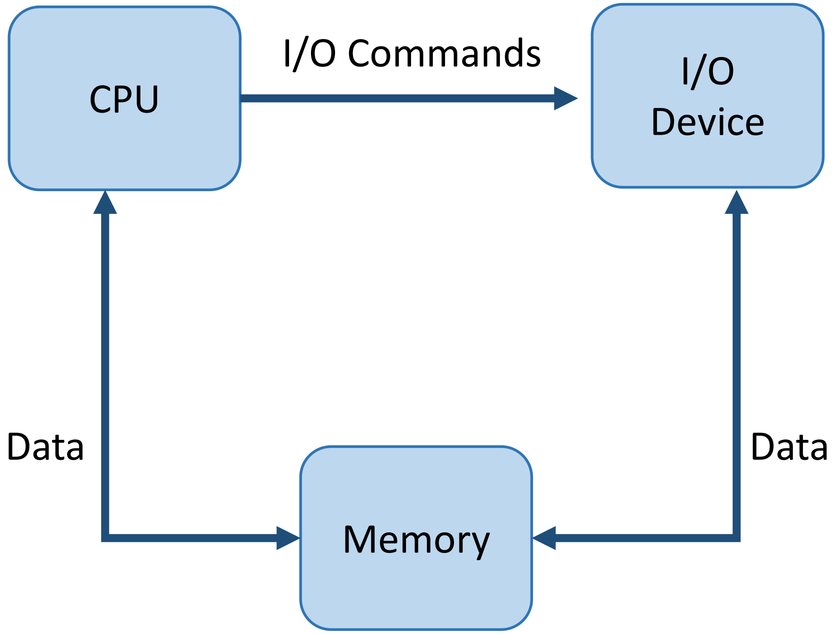
Implementation

- Direct connection between I/O device and certain main memory locations so that I/O device can **transfer block of data** to/from memory without going through CPU.

- OS allocates buffer in memory to the I/O device to send data to the CPU.
    - I/O device operates **asynchronously** with CPU
    - Interrupts CPU when finished.

Efficiency

- Memory mapped IO is ideal for most high-speed I/O devices like disks, communication interfaces.



Design Objectives

- **Efficiency**
  - Most I/O devices are extremely slow compared with the processor and main memory
    - Buffering is one way to deal with this issue

- **Generality**
  - It is desirable to handle all devices in a uniform and consistent manner
    - In the way user processes see the devices
    - In the way the Operating System manages the I/O devices and operations

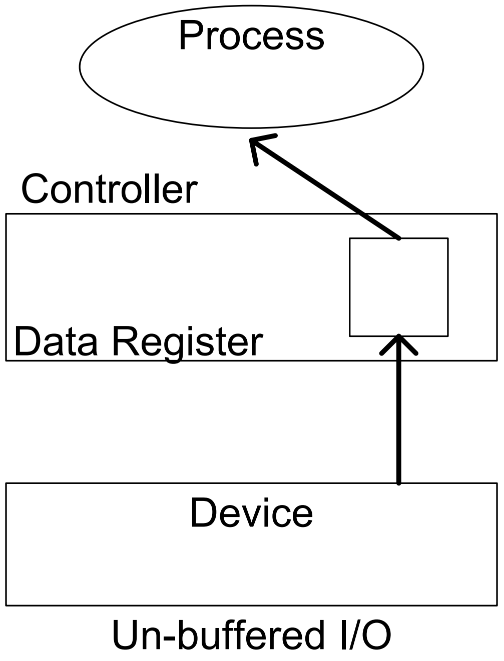
Buffering

Buffering is the technique by which the device manager can keep slower I/O devices busy during times when a process is not requiring I/O operations.

- **Input buffering:** having the input device read information into the primary memory before the process requests it.
- **Output buffering:** saving information in memory and then writing it to the device while the process continues execution

Hardware Level Buffering

Consider a simple character device controller that reads a single byte from a router for each input operation.



Normal operation:

1. Read occurs
2. The driver passes a read command to the controller
3. The controller instructs the device to put the next character into one-byte data controller's register
4. The process calling for the byte **waits** for the operation to complete

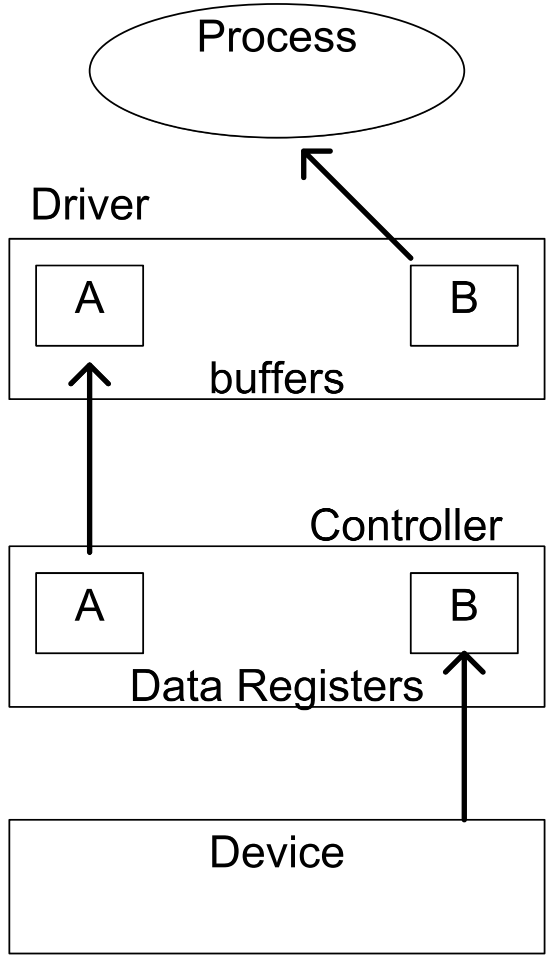
- then retrieves the character from the data register

Buffered operation:

- The next character to be read by the process has already been placed into the data register, even though the process has not yet called for the read operation
- Adding a hardware buffer to the controller **decreases** the amount of time the process has to **wait**

Driver Level Buffering

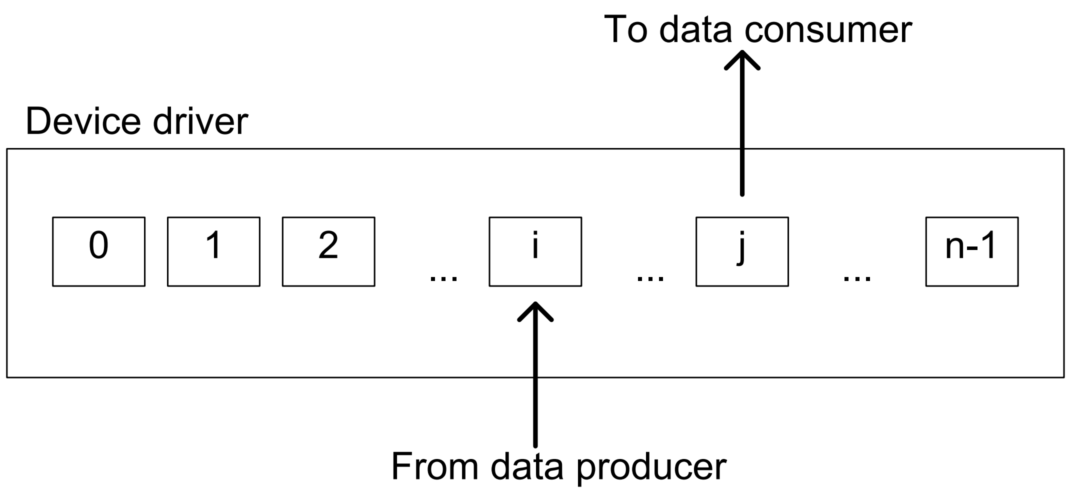
- This is generally called **double buffering**
  - One buffer is for the driver to store the data while waiting for the higher layers to read it
  - The other buffer is to store data from the lower-level module



Reading into driver buffer A

Using Multiple Buffers

- The number of buffers is extended from two to n
- The data producer is writing into buffer i while the data consumer is reading form buffer j
  - In this configuration:
    - If i<j: buffers [j+1, n-1] and [0, i-1] are full
    - If j<i: buffers [j+1, i-1] are full



- This is known as ***circular buffering***

References

- “Operating Systems”, William Stallings, ISBN 0-13-032986-X
- “Operating Systems – A modern perspective”, Garry Nutt, ISBN 0-8053-1295-1