Introduction to I/O subsystem

Mythili Vutukuru
CSE, IIT Bombay

Input/Output Devices

- CPU and memory connected via high speed system (memory) bus
  - Bus = set of wires carrying signals
- I/O devices connect to the CPU and memory via other separate buses
  - High speed bus, e.g., PCI
  - Other: SCSI, USB, SATA
- Point of connection to the system: port

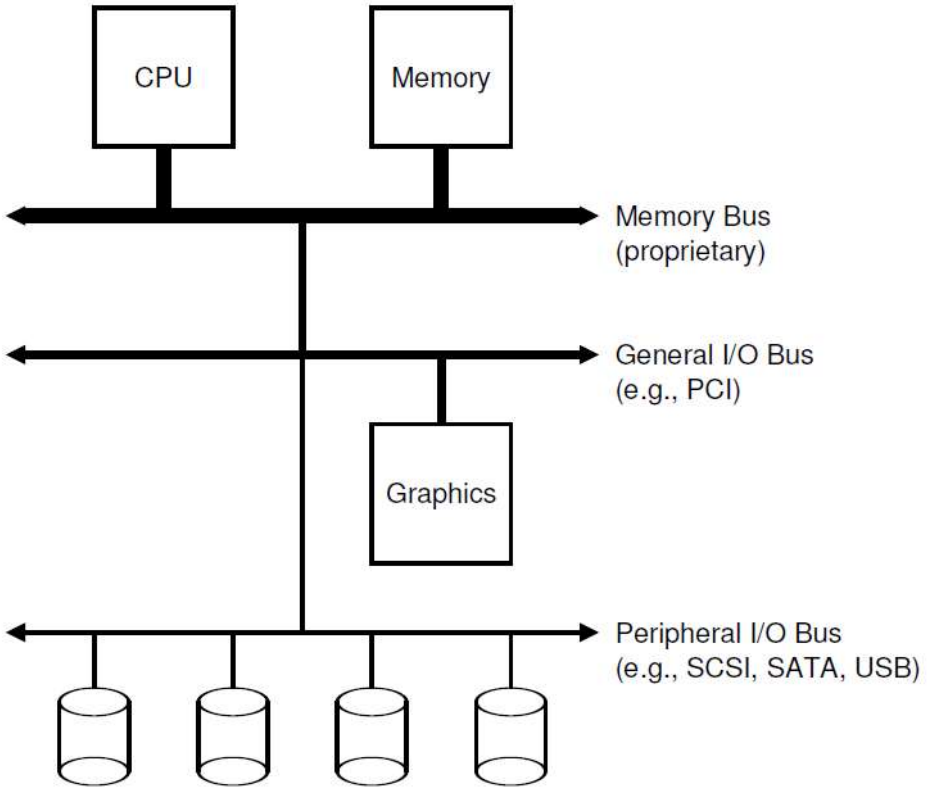


Image credit: OSTEP

Simple Device Model

- Block devices store a set of numbered blocks (disks)
- Character devices produce/consume stream of bytes (keyboard)
- Devices expose an interface of memory registers
  - Current status of device
  - Command to execute
  - Data to transfer
- Device controller manages device, internals of device are usually hidden

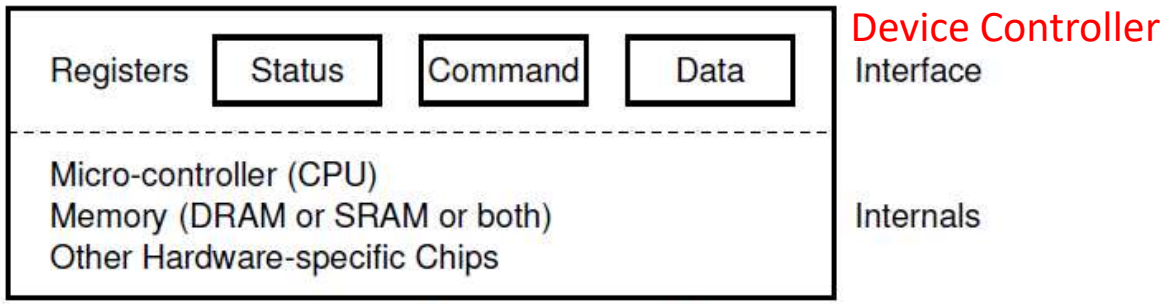


Figure 36.2: **A Canonical Device**

Image credit: OSTEP

How does OS read/write to device registers?

- OS communicates with device via device controller on the device
- OS code that communicates with device is called device driver
- OS reads/writes registers in the I//O device: how?
- Explicit I/O instructions
  - E.g., on x86, `in` and `out` instructions can be used to read and write to specific registers on a device
  - Privileged instructions accessed by OS
- Memory mapped I/O
  - Device makes registers appear like memory locations
  - OS simply reads and writes from memory
  - Memory hardware routes accesses to these special memory addresses to devices

A simple execution of I/O requests

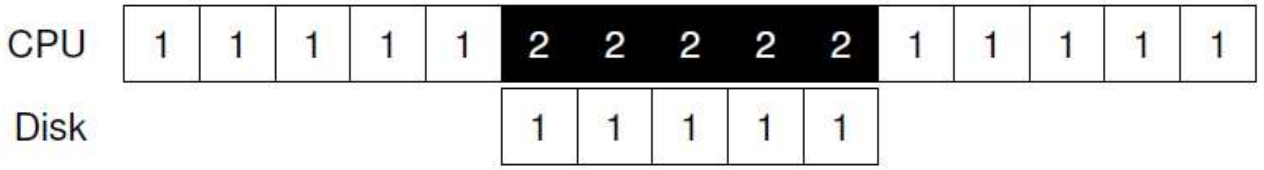
```
While (STATUS == BUSY)
    ; // wait until device is not busy
Write data to DATA register
Write command to COMMAND register
    (Doing so starts the device and executes the command)
While (STATUS == BUSY)
    ; // wait until device is done with your request
```

- Simple model of I/O, e.g., read/write block of data from disk
  - Give command to device via command register
  - Write: transfer data to device via data register
  - Poll status register to see if I/O operation completes
  - Read: Copy data from data register to main memory after I/O completes
- Polling status to see if device ready constantly – wastes CPU cycles

Interrupts

- Polling wastes CPU cycles

- Instead, OS can put process to sleep and switch to another process



- When I/O request completes, device raises interrupt, OS can switch back to original process after that request has completed
  - Note: context switch to original process need not be immediate

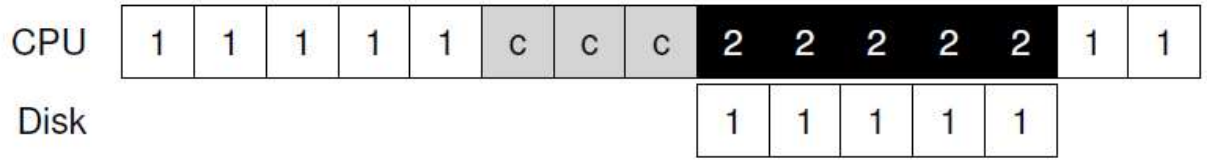
Image credit: OSTEP

Interrupt handler

- Interrupt from I/O device causes trap, switches process to kernel mode
- Interrupt Descriptor Table (IDT) stores pointers (value of PC) to OS interrupt handlers (interrupt service routines)
  - Interrupt (IRQ) number identifies the interrupt handler to run for a device
- Interrupt handler processes notification from device, unblocks the process waiting for I/O (if any), and starts next I/O request (if any pending)
- Handling interrupts imposes kernel mode transition overheads
  - Note: polling may be faster than interrupts if device is fast

Direct Memory Access (DMA)

- In spite of interrupts, CPU cycles wasted in copying data to/from device



- Instead, a special piece of hardware (DMA engine) copies from main memory to device and vice versa, without involving CPU
  - CPU gives DMA engine the memory location of data
  - In case of disk read, device copies data via DMA to RAM and then raises interrupt
  - In case of write, device copies data via DMA from RAM and then starts writing

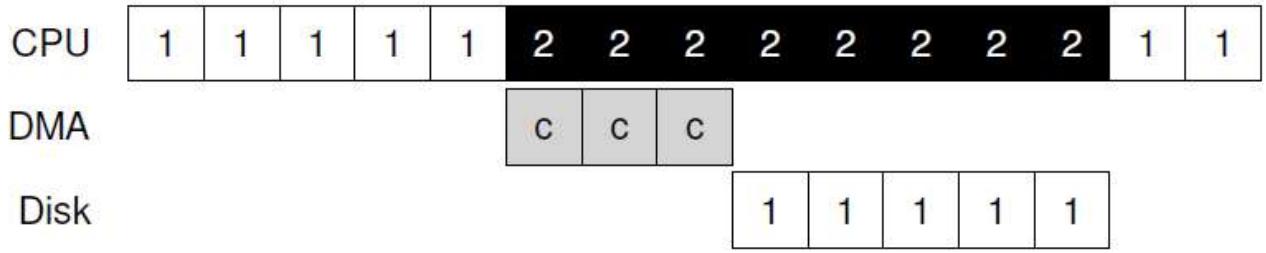


Image credit: OSTEP

Summary of disk read with interrupt + DMA

- Process P1 makes read system call to read data from disk
- OS gives command to disk via command register
- OS switches to another process P2 (P1 cannot run anymore, blocked)
- Disk completes reading data, copies data into main memory directly via DMA, then raises interrupt
- OS handles interrupts, disk data is ready, marks P1 as ready to run
  - Interrupt handled in the kernel mode of P2
- OS scheduler switches back to P1 at a later time

I/O stack in the kernel

- Device driver: part of OS code that talks to specific device, gives commands, handles interrupts etc.
  - Rest of OS code abstracts out the device-specific details
- I/O subsystems (file system / networking) built as layers: system calls, block read/write, device drivers that communicates with I/O device

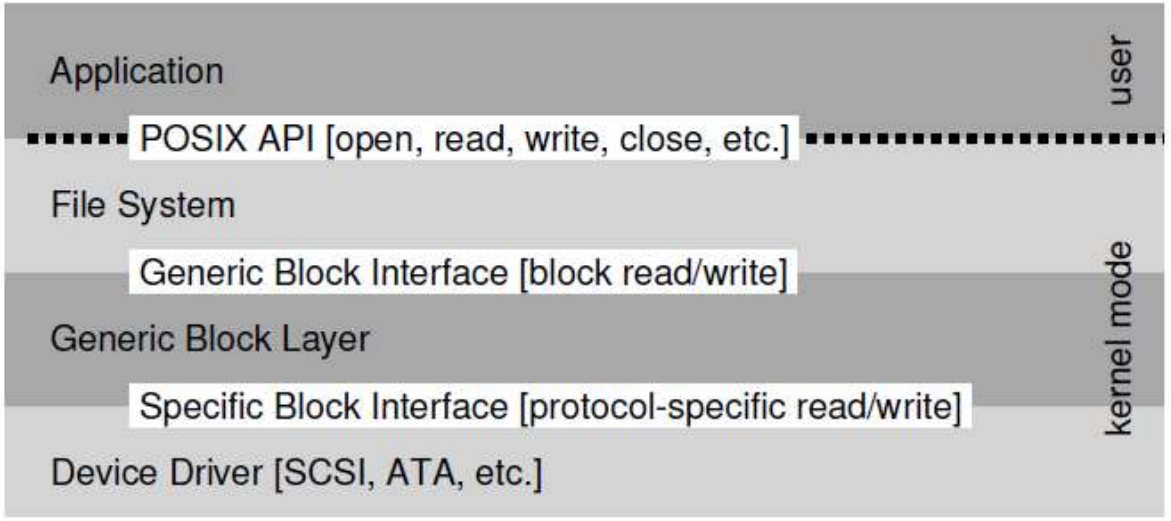


Image credit: OSTEP