Operating Systems

Eric Lo

14 – I/O

I/O device controller

- Block devices vs. Character devices

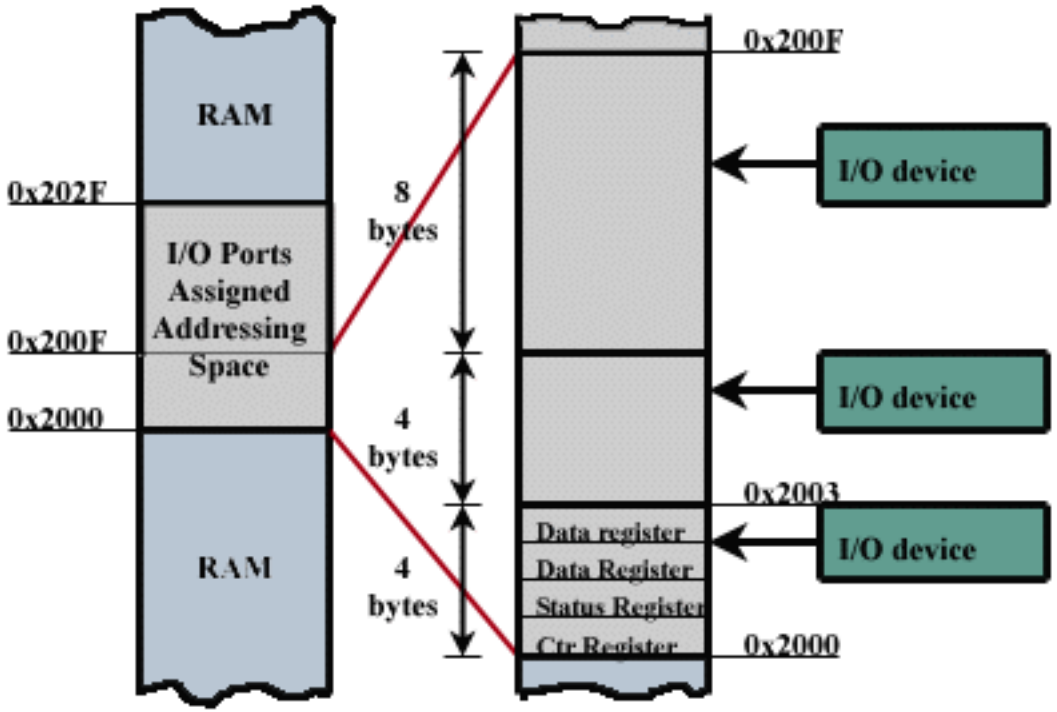
- E.g., Character device controller's tasks
  - Raw serial **bit** stream ←→ **bytes** for controller's registers
  - Checksum error handling
  - Built-in **registers** and data buffer for communicating with the CPU

I/O device controller

- Built-in registers and data buffer for communicating with the CPU
  - Direct I/O
    - CPU places data into (and read data from) register/data buffer directly through instructions like
      - `IN REG, PORT` //read the value at PORT of the device to CPU register REG
  - Memory-mapped I/O
    - Map the device's registers and data buffer to the memory space
      - » Both kernel (driver) and user-space can do that

Memory-mapped I/O

- Kernel Space



Ack: McGill

Device Drivers

- Follow the standard programming interface offered by the OS
  - After all, the programming interface for all printers are the same, right?
  - The device's manufacturer follows the interface to implement a driver that includes code to read and write the device's registers
- E.g., Microsoft Windows Interface:
  - `printer.print(Byte b);`

```
• Canon-C123Driver implements printer.print(Byte b){
      MOV CanonPrinterRegister b  //none of the OS's business here
  }
   HP-T125Driver implements printer.print(Byte b){
      MOV EpsonPrinterRegister b  //none of the OS's business here
  }
```

I/O Communication protocol

- Polling
  - CPU puts one byte to (e.g., printer) device's DATA register
  - CPU keeps polling the device's READY register in order to put the next byte
- Interrupt
  - CPU waits for interrupt and does something else in between
- Direct Memory Access (DMA)
  - DMA controller on system bus
  - Offloading the **per-byte** polling/interrupt job from CPU to DMA controller

DMA

- OS sets up a memory-mapped region (i.e., memory-mapped I/O) in the kernel space
- OS pins those memory-mapped pages in memory
  - So that those pages are kept for the I/O communication
    - Won't let paging swap them away

Direct Memory Access

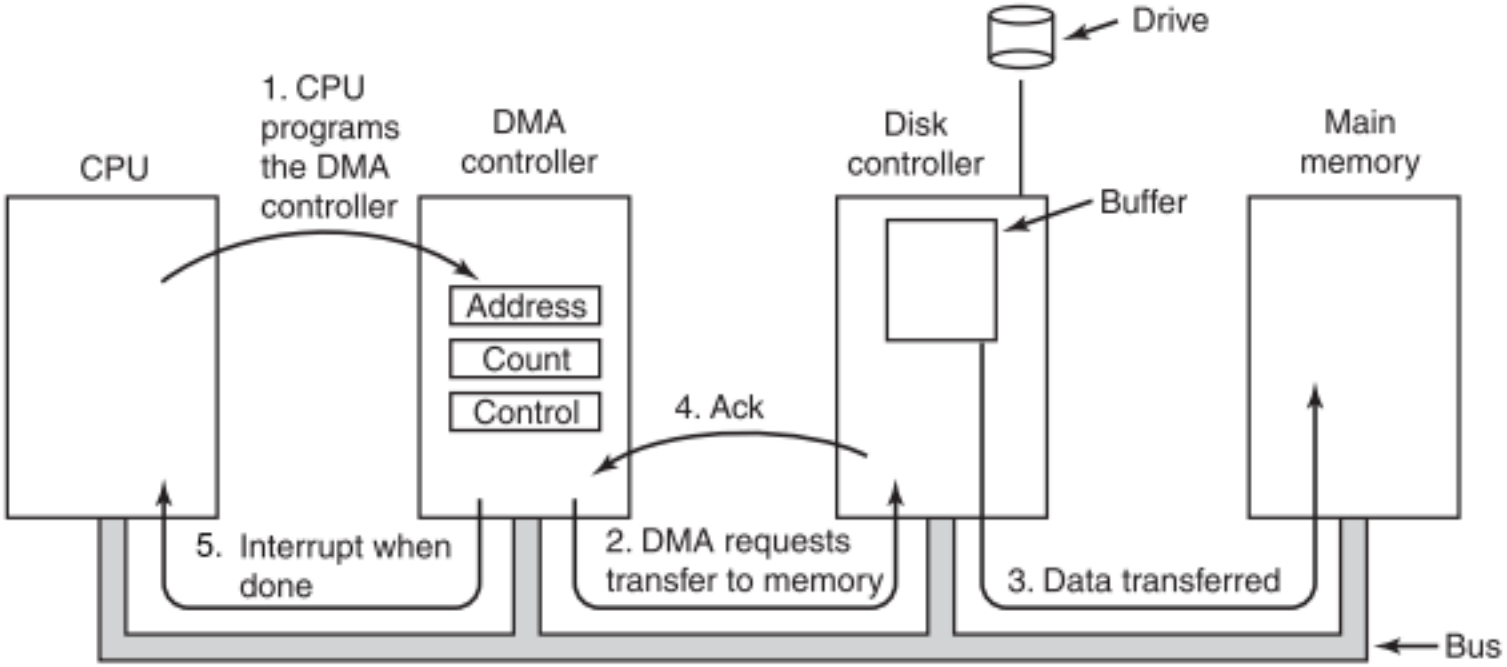
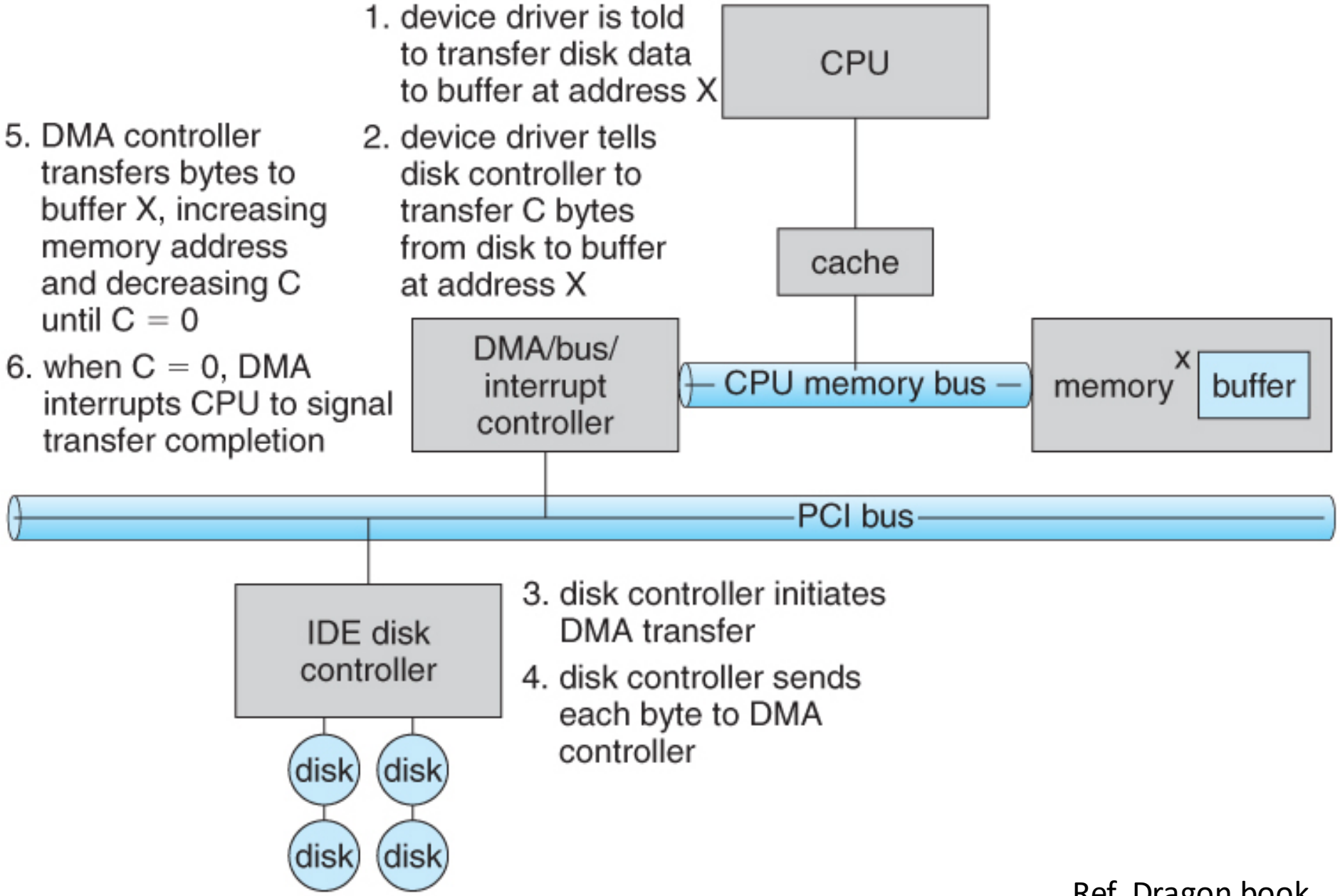


Figure 5-4. Operation of a DMA transfer.

Direct Memory Access



Ref. Dragon book

Print Spooling

- A printer obviously shouldn't be a concurrent sharable "file" (character special file)
  - No >1 process can print to the printer together
- A user process exclusively opens the printer but does nothing for hours?
  - No other processes can print!
- Create a root level printer **daemon** process, and a **spooling** directory
  - To print a file, a user process has to
    - generate the entire file to be printed
    - put it to the spooling directory
  - Only the daemon can access the printer device file

Wiki:

- documents formatted for printing are stored in a queue at the speed of the computer, then retrieved and printed at the speed of the printer.
- Multiple processes can write documents to the spool without waiting, and can then perform other tasks, while the "spooler" process operates the printer

BTW, all daemon background processes have a naming convention

inetd, httpd, nfsd, sshd, named, and lpd

Memory-Mapped I/O on user-process

- Can also be used in user-space
- Set up an mmap to link to a file
- Write to an address = write to a file
- Advantage?