3.1.1 I/O Device types

Block Devices:

- Reads and writes data in fixed sized, addressable blocks.
- May be random access.
- Examples:

  Disks (usually random access/block addressable).

  Mag-tape (may be block addressable, normally read and written sequentially.

Character Devices:

- Delivers and accepts a stream of characters.
- Not block addressable and do not have a seek operation.
- Examples: printer and keyboard

Other Devices:

- Clocks; readable as a single word or set of words, or may be used to generate interrupts. Not block addressable nor accept/deliver characters.
- Bit-mapped screens; special hardware interprets values in memory as pixels.

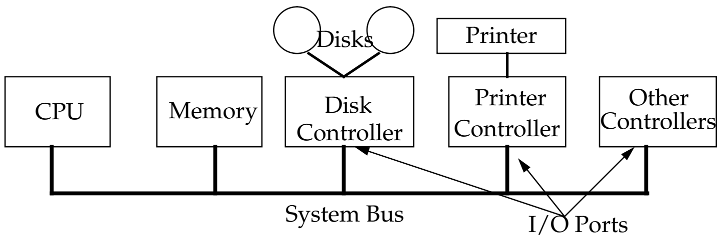
Device Drivers:

Low level device-dependent OS software that controls the device.

Device drivers translate abstract, device-independent requests into device-dependent commands.

For example, a disk device driver might take requests to read or write block N of a particular device, and translate this to commands for a specific type of disk drive.

Device Drivers issue commands to **Device Controllers** (cards that plug onto the I/O bus) through I/O ports or registers.



I/O ports can be:

- Memory-mapped: Registers are part of the memory address space.
- Mapped into a special address space if system has a separate I/O bus.

  In this case, a mechanism is needed in order to address them.

  For 80x86 machines, the assembly instructions **in** and **out** are used.

Device Controller responsibilities:

- Convert serial bit streams into a blocks of bytes.
- Control drive motors and analog circuitry for video.
- Perform error correction and bad block mapping (in newer drives, e.g. IDE).

Device Controllers signal CPU using hardware interrupts.

- Hardware Interrupt Request line (**IRQ**) is a wire from the controller to an interrupt controller chip, e.g. PIC. Newer cards (controllers) allow IRQ to be selected using software.
- Interrupt controller maps each IRQ line into an interrupt vector.
- Interrupt vector is an address of an interrupt service routine (ISR).

3.1.3 Direct Memory Access (DMA)

DMA introduced to free the CPU from moving data between controller's registers and main memory.

Basic idea: let a device "talk" directly to main memory, bypassing the CPU.

In systems with DMA controllers, OS gives the controller:

- Disk address of data and a read/write flag.
- Memory address (buffer) in which to read or write the data.
- Number of bytes of data to transfer.

For read requests, when the data is ready, the controller will grab the bus and do the DMA transfer.

Therefore, the transfer is performed without going through the CPU.

Controller sends interrupt when finished.

Without DMA, data would have to be transferred a byte at a time by the CPU, through I/O registers to the desired memory buffer address.

DMA controller **buffers** data from the mechanical device and main memory.

This is advantageous because:

- Mechanical device delivers bits at constant rate.
- Data can not be placed in main memory buffer at a constant rate because bus is shared with other controllers resulting in contention.
- Buffers provide a mechanism to increase data **bandwidth** (throughput) of the device.
- For disks, data block **interleaving** can be used to match bus transfer bandwidth with device bandwidth.

DMA controller can be detrimental to system performance:

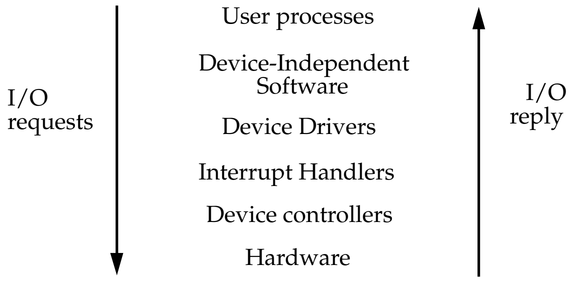
if the DMA controller is slower than the CPU (often the case) AND

if the device is able to produce data faster than the controller can transfer it AND

if the CPU is NOT able to do other useful work.

3.2 Principles of I/O Software

Basic idea: Organize software as a series of layers.



Lower levels hide peculiarities of hardware from higher layers.

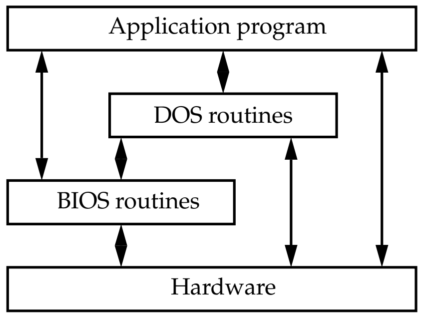
Upper layers concerned with providing a nice, clean, regular interface to the users.

Issue #1: **Device Independence**

The ability to write generic programs that can read and write to a device independent of its type, e.g. floppy, hard disks, CDROMs, etc., e.g.

'sort < input > output' should work for input coming from keyboard, floppy, etc. and going to floppy, hard disk or screen.

However, not all OSs restrict user access through the device-independent software layer, e.g. DOS:



What are the advantages and disadvantages of such a system ?

Issue #2: **Uniform naming**

The name of a file or device should simply be a string or an integer and not depend on the device in any way.

For example, in UNIX, devices are identified using filenames (e.g. /dev/fd0) that can be **mounted** at an arbitrary place in the file system of another device, e.g. /home/jimp/floppy.

Copying a file to /home/jimp/floppy copies it onto the floppy disk, not the hard drive.

Therefore, all files and devices are addressed the same way (pathname).

Issue #3: **Error Handling**

Errors should be handled as close to the hardware as possible, e.g. at the controller and device driver level.

Most errors are transient, e.g. dust particle on disk head.

Therefore, error recovery, in many cases, is transparent to upper layers.

Issue #4: **Synchronous (blocking) vs. asynchronous (interrupt-driven) transfers.**

Most physical I/O is asynchronous - CPU starts transfer and switches to another task until an interrupt is delivered.

User program requests for input are usually blocking since, in most cases, the program can not continue without the requested data.

OS takes care of making interrupt-driven I/O handling appear as blocking to user programs.

Issue #5: **Shared vs. dedicated I/O devices.**

Disks are typically shared among users. OS must handle interleaved requests for data operations correctly.

Tapes are dedicated devices. OS must prevent shared access to such devices.

3.2.2 Interrupt Handlers

Processes block when doing an I/O operation or a Receive for a message that is not available.

Interrupt Handlers are responsible for removing the blocked status of a process, making it eligible to run again.

We talked about how interrupt handlers are called in the introduction.

3.2.3 Device Drivers

All device-dependent code goes into the device driver.

e.g. a floppy device driver is the only piece of OS software that knows about the I/O control registers of a floppy disk and how to control them.

A device driver accepts abstract requests from device-independent software above it and makes sure the request is carried out, e.g. read block n.

In general, a device driver waits on either:
- Requests (system call from a user process).
- An interrupt or message (e.g., from a disk controller).

Typical sequence of low level-events:

- Device driver receives I/O request (e.g., get disk block K), writes commands to I/O ports of controller.
  For operations that can not be carried out immediately, the device driver must block. So user processes block in the OS device driver routine.
- Controller issues interrupt when request is performed (block K is ready) and ISR wakes up blocked device driver.
- The awakened device driver checks for errors and, if none, passes the data and status information through the device-independent software layer to the user process.

For disks,

- Device drivers translate abstract addresses, e.g., block n, into head, track cylinder values (at least before IDE and SCSI drives !).
- They determine and execute the proper sequence of controller operations, e.g., start motor, position head, etc.

Device drivers queue up multiple requests. If no additional requests are pending, they block again waiting for the next request.

3.2.4 Device-Independent I/O Software

Functions of the device-independent I/O software:

- Perform the I/O functions that are common to all devices and provide a uniform interface to user-level software, e.g.,
  - Unix library routines printf, fprintf, sprintf, etc.
- Maps symbolic device names onto the proper device driver, e.g.,
  - In UNIX, a device name is a file, e.g., /dev/tty00 that contains a **major device number** and a **minor device number**.
    - The major device number is used to locate the appropriate device driver.
    - The minor device number is passed to the device driver and specifies the unit, e.g., the 1.44MByte floppy vs. the 1.2MByte floppy.
- Provide user access restriction to devices, e.g.,
  - In UNIX, access is restricted using the file protection mechanism, *rwx*.
  - In DOS, there is no protection mechanism.

- Provide a device-independent block size, e.g.,
  - Different physical sector sizes (e.g. hard drives vs floppy disks) are mapped into a single **logical** block size for higher level functions.
  - Higher layer functions can then be written generically to operate on abstract devices which read and write logical blocks.
- Provide buffering for both block and character devices, e.g.,
  - For block devices:
    - Hardware reads and writes entire blocks.
    - Programs generally read and write bytes at a time.
    - OS maintains buffers where read and write user operations are carried out.
    - OS writes changes out to disk in units of blocks when necessary.
  - For character devices:
    - User programs can usually generate data faster than device can transmit it (e.g. modem). Therefore, buffering is required else characters are lost.

- Provide storage allocation on block devices.
  - OS must maintain list of free disk blocks in order to satisfy user allocation requests.
  - Algorithm for locating a free disk block is device independent and can be done above the level of the driver.
- Allocate and release dedicated devices.
  - OS examines requests for devices and determines whether or not the device is available.
  - For devices like CDROM recorders, only a single process is allowed access.
  - OS can provide system call ‘Open’ that reserves access to the device if it is available or fails if it is not.

- Perform error reporting.
  - Error handling performed by device drivers.
  - If error can not be corrected by device driver, it informs the device-independent software.
  - How the error is treated from here is device independent, e.g.,
    - Floppy disk device driver fails to read a sector on a damaged floppy disk.
    - Driver tries several times and gives up.
    - Driver reports failure to layer above.
    - If damaged sector was boot sector, device-independent software can report “system panic” in UNIX, or “system halted” in DOS.
    - If it is a user file, error message is reported (possibly recommending Norton’s disk doctor).