

COMP3301

Lecture 4 I/O: concepts

School of Information Technology and Electrical Engineering
The University of Queensland

Operating Systems and I/O

- One of major functions is to control I/O
 - issue commands
 - handle interrupts
 - handle errors
 - provide interface
 - ideally device independent

I/O Devices

- UNIX systems typically divide most I/O devices into **block devices** and **character devices**.
- **Block devices** such as disks, tapes, CDs, etc. Can often read/write any block independently.
- **Character devices** read/write a stream of characters serially, eg. the console.

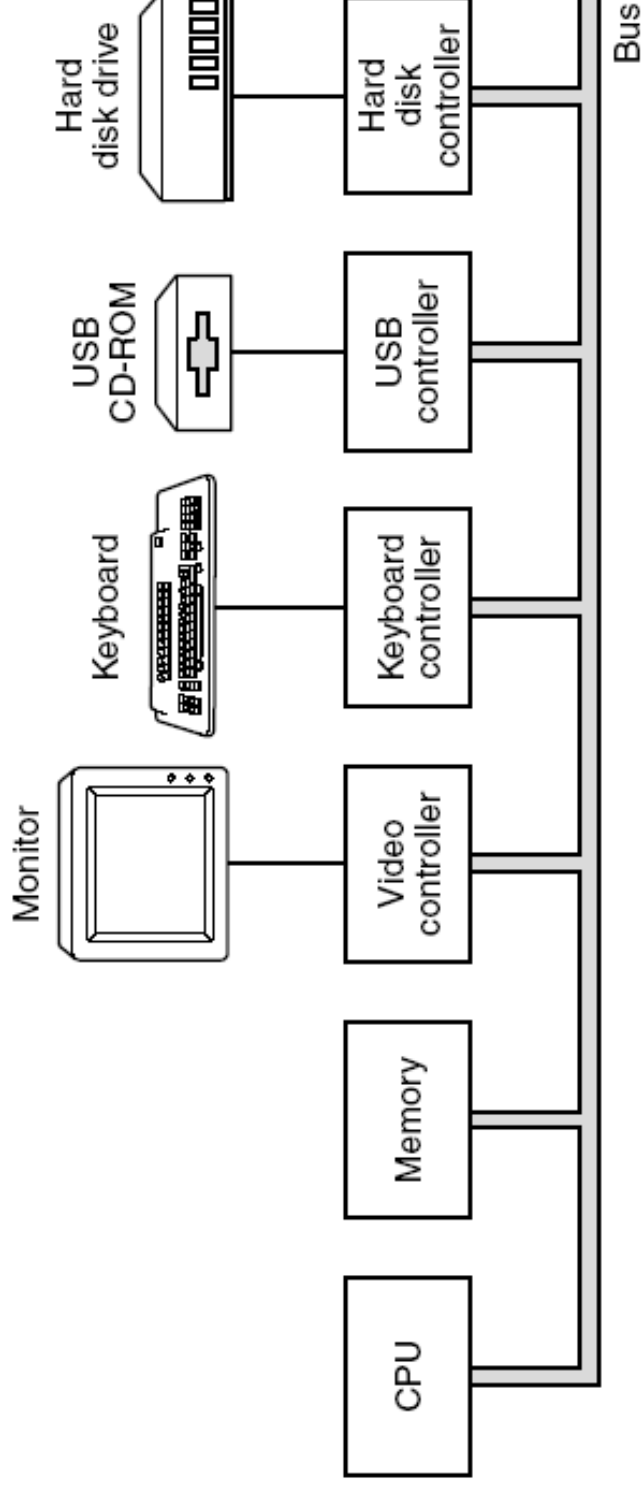
I/O Device Data Rates

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Scanner	400 KB/sec
Digital camcorder	4 MB/sec
52x CD-ROM	8 MB/sec
FireWire (IEEE 1394)	50 MB/sec
USB 2.0	60 MB/sec
XGA Monitor	60 MB/sec
SONET OC-12 network	78 MB/sec
Gigabit Ethernet	125 MB/sec
Serial ATA disk	200 MB/sec
SCSI Ultrawide 4 disk	320 MB/sec
PCI bus	528 MB/sec

- Figure 3-1. Some typical device, network, and bus data rates.

Device controllers

- Mechanical component
 - Connectors etc
- Electronic component
 - device controller



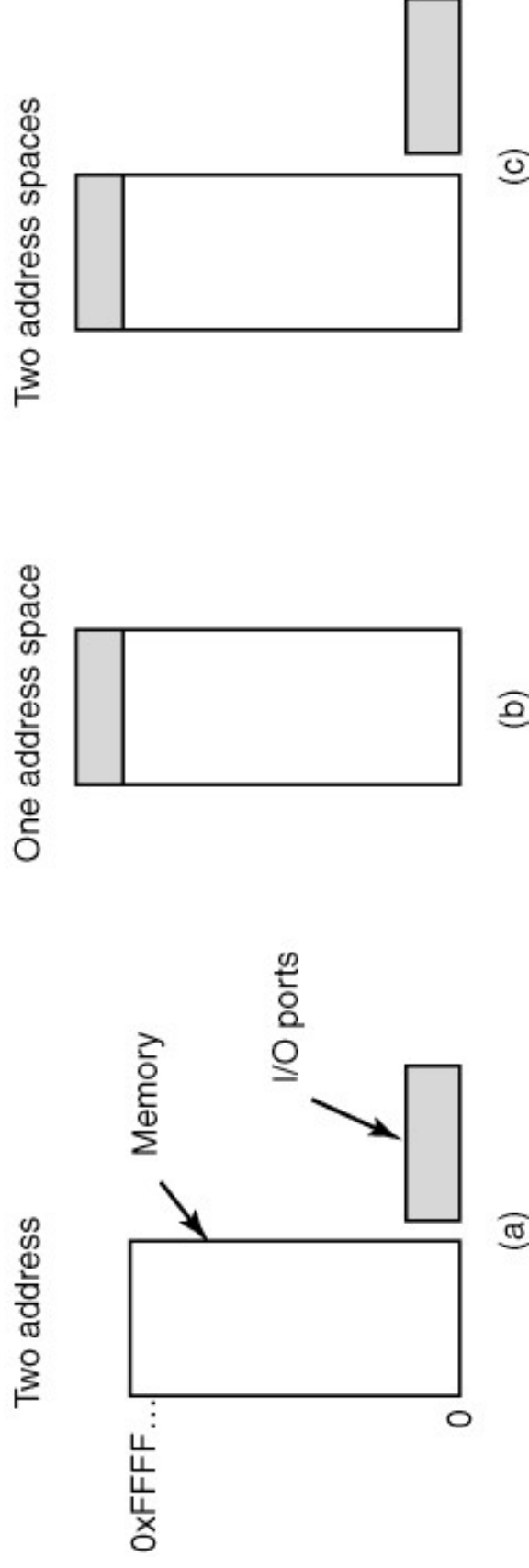
Device Controllers

- Have two interfaces.
- On one side, interface to the CPU using a standard protocol, such as PCIe.
- On the other side, interface to the device, with a device specific protocol, maybe standardised (IDE)
- USB controller controls serial I/O stream to another external controller!

Memory Mapped I/O

- Here the controller appears to the CPU as a set of read/write memory locations, perhaps in a special I/O address space.
- Control & Status Registers are for commands and status information for the controller
- Data Registers are the data to be actually input or output.

Memory-Mapped I/O



- Figure 3-3. (a) Separate I/O and memory space. (b) Memory-mapped I/O. (c) Hybrid.

Interrupts

- I/O is usually much slower than CPU computation. For fast I/O, CPU only send blocks, so still slow compared to CPU.
- Wasteful to wait for I/O to complete by a **busy wait** on status bits, typically a process will be blocked until I/O is complete, and another is scheduled to run.

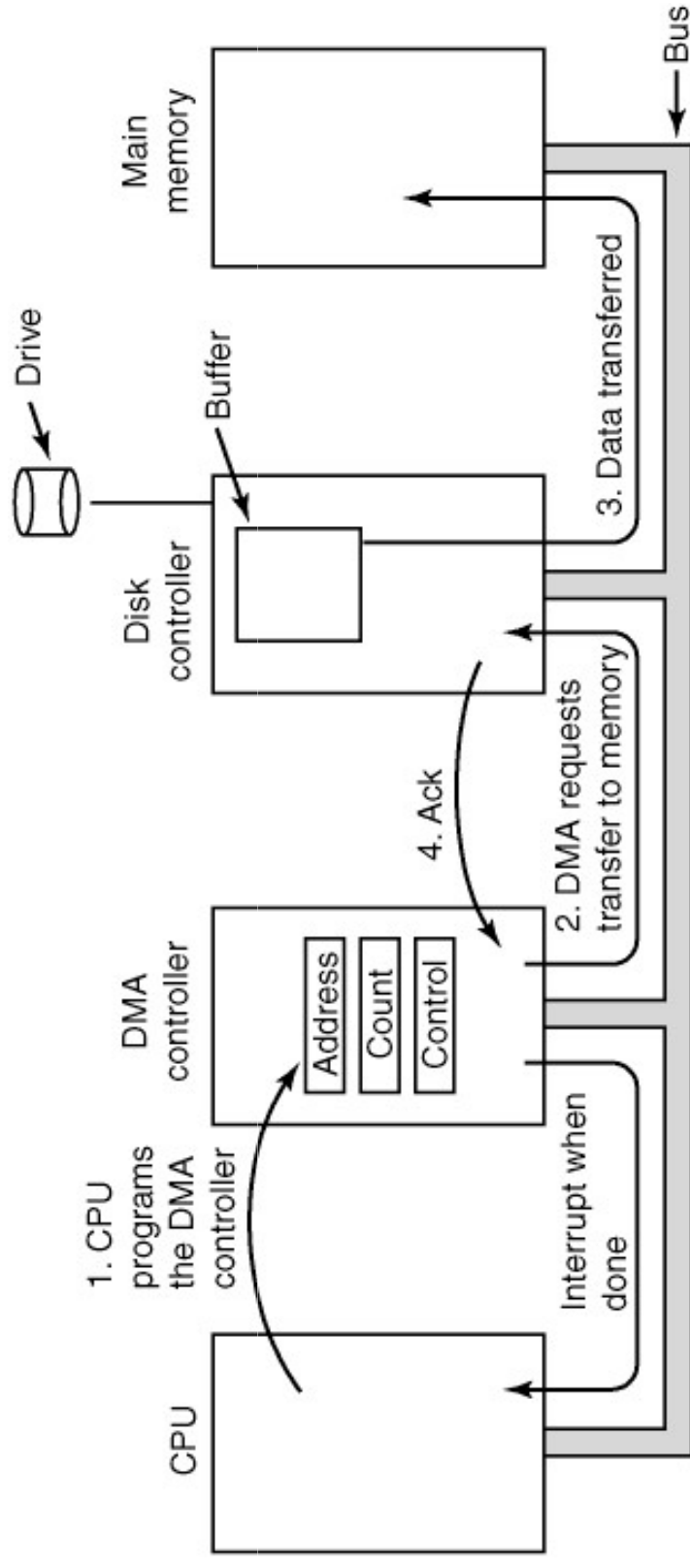
Interrupts

- Most controllers can directly signal the CPU that an operation has completed (I/O finished, keyboard pressed, timer expired) by a hardware interrupt line.
- When an interrupt occurs, the CPU jumps to an interrupt service routine, which “unblocks” process, and calls the scheduler.
- Many systems have priority interrupt controllers, so that multiple (15 on PC) different interrupts can be quickly distinguished.

Direct Memory Access (DMA)

- To write a block of data to memory would require CPU to read each word from memory, and write to the memory mapped disk controller data registers. Every word appears on the bus twice, and CPU is busy.
- A DMA controller takes over the memory bus, and transfers a block between memory and I/O registers with (perhaps) only one bus transaction, also frees the CPU.

Direct Memory Access (DMA)



- Figure 3-4. Operation of a DMA transfer.

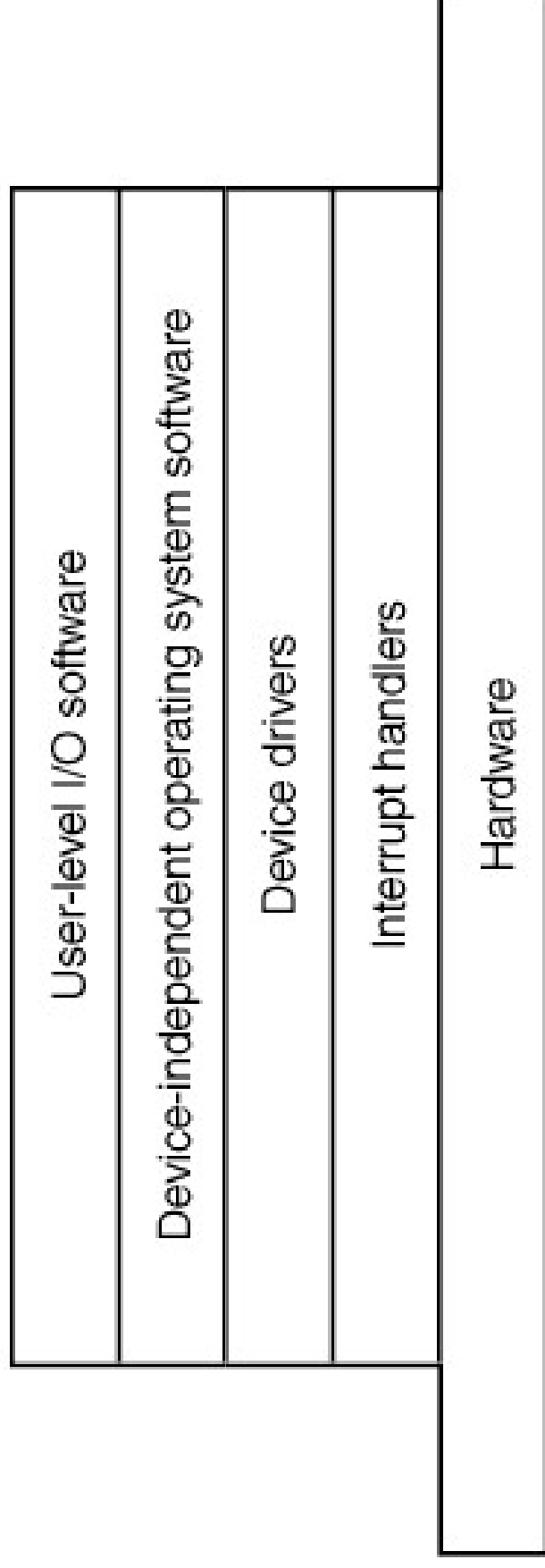
I/O Software

- How does the programmer see I/O?
- A major goal is device independence
 - A program should be able to be written, so that it can receive I/O from a variety of sources, without needing to know the source at compile time.
 - In UNIX, stdin and stdout are I/O stream devices, or even to I/O from other programs.
- Uniform naming
 - Any device can be accessed as, say, a text string, rather than different sorts of names.

I/O Software

- Error Handling
 - Errors handled as close to the device as possible
- Synchronous vs Asynchronous
 - OS lets asynchronous I/O (interrupts) look like synchronous I/O (blocking)
- Buffering – eg., disk reads 1024 bytes at once, but program reads one-by-one

Goals of the I/O Software



- Figure 3-5. Layers of the I/O software system.

Device Drivers

- This software converts between a device-independent I/O call from a program, and a device-specific set of instructions to the device controller (via control, status, data registers).
- Device drivers are typically written by device manufacturers, and are very specific to an operating system.
 - Linux 2.4, Linux 2.6, Minix, other UNIX's are all different, Windows different again.

Device Drivers

- Some classes of devices have similar controller interfaces so that a single driver can be used, eg. IDE disks and CD-ROMs.
- Sometimes a new device will mimic another device so it can share its device driver, eg. mouse, touch-screen, joystick.
- Operating systems typically have different software interfaces for character devices and block devices.

Kernel vs User-Space Drivers

- Device drivers are often the “buggiest” part of an OS, since they are supplied by third parties.
- Compiling them into the kernel makes them fast, but inflexible.
- Dynamically linking them into the kernel is flexible, but error-prone.
- Micro-kernels run device drivers as user-mode processes which need to do a kernel call for physical I/O – safer but slower.

Interrupt Handlers

- What happens when an interrupt occurs.
- The CPU jumps to an **interrupt service routine** or **interrupt handler**– but what happens then.
- Interrupt handling is done in many different ways in different operating systems, the usual approach is to do as little as possible in the ISR, and leave the bulk of processing to the device driver.
- May exit ISR via the scheduler, since some processes now unblocked.

Interrupt Handler

- For user-mode device drivers, an ISR may simply do a signal or “up” on a semaphore, which unblocks the waiting driver.
- Most device drivers need to also include some ISR code, which indicates what to do when the device interrupts.

Device-Independent I/O Software

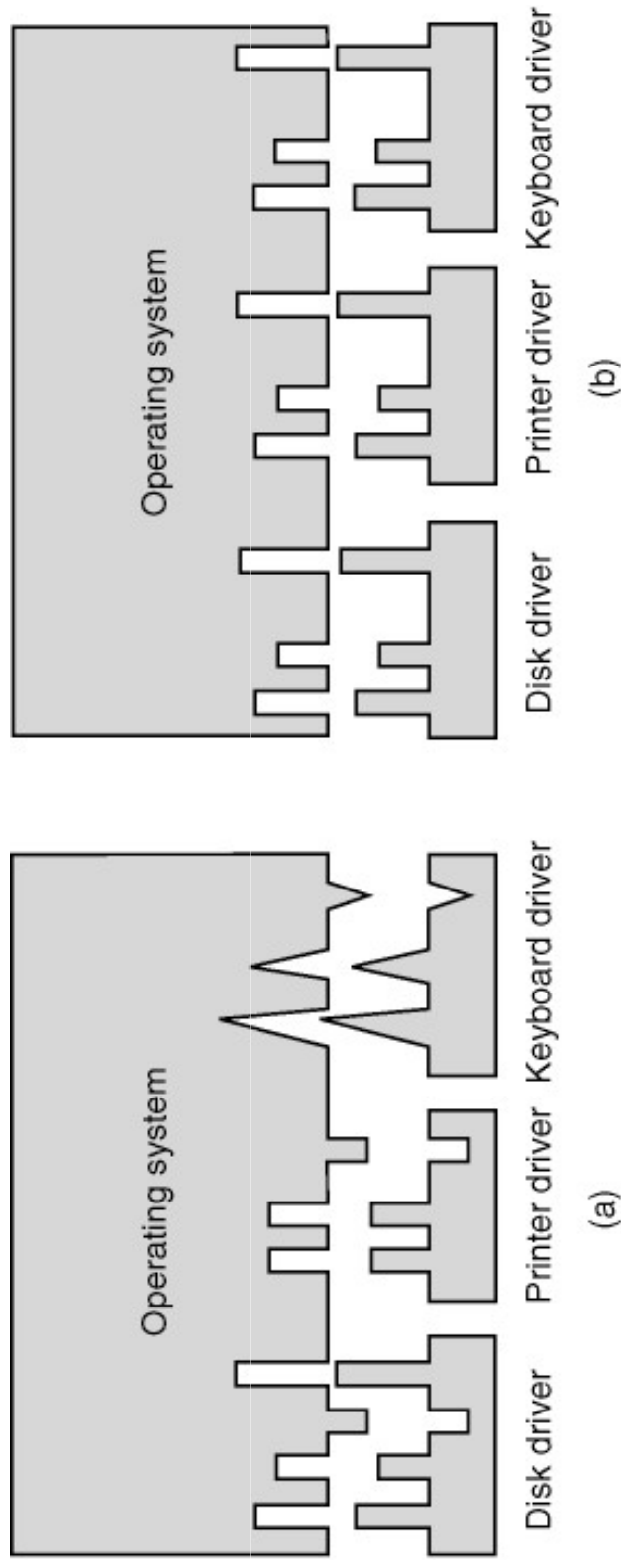
Uniform interfacing for device drivers
Buffering
Error reporting
Allocating and releasing dedicated devices
Providing a device-independent block size

- Figure 3-6. Functions of the device-independent I/O software.

Uniform Device Driver Interface

- This is the interface between the OS and the device driver.
- Uniformity makes drivers easier to write - often just modify an existing driver.
- Note every driver needs every feature so may be less efficient
- I/O device Naming - use a **Major Device Number** (which driver to use) and a **Minor Device Number** (which device out of a set)
- Protection - not every user can always access every device.

Uniform Interfacing for Device Drivers



- Figure 3-7. (a) Without a standard driver interface.
- (b) With a standard driver interface.

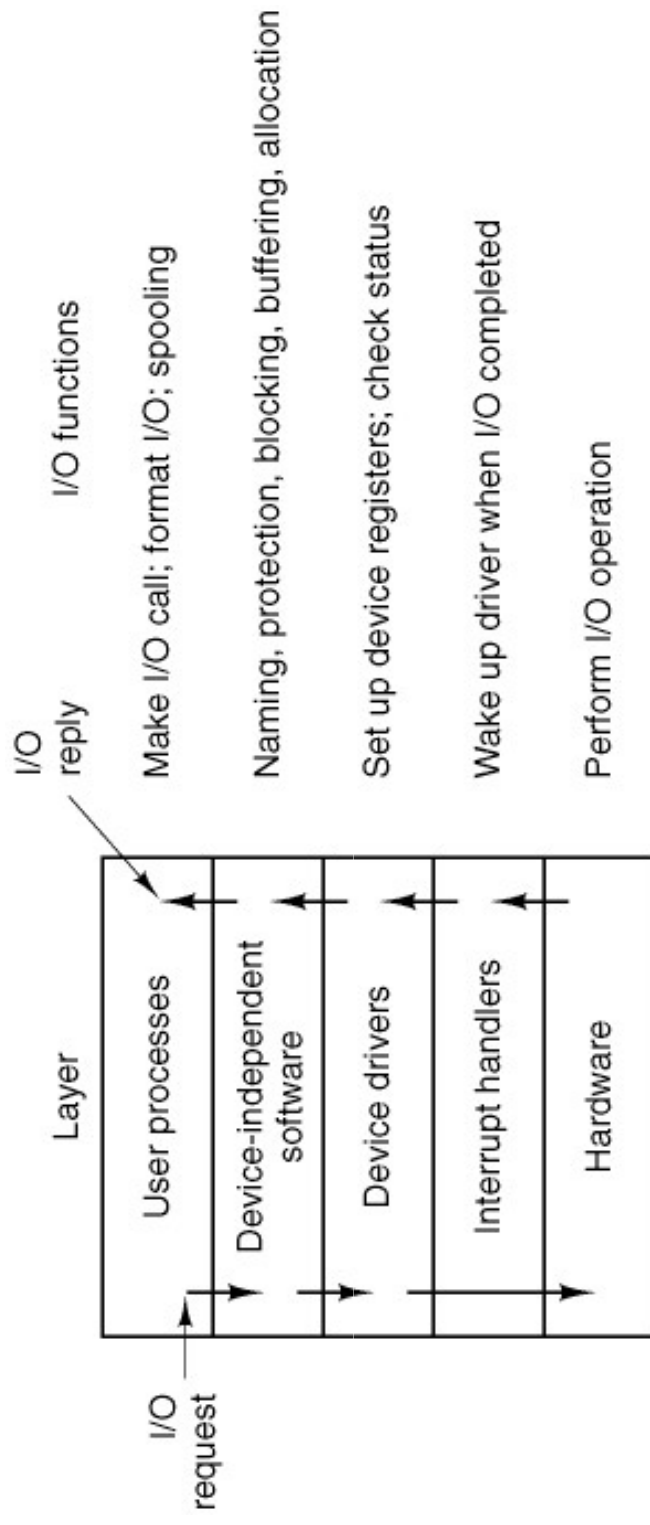
I/O Software features

- Buffering - OS handles block buffering and stream buffering
- Error handling and reporting
- Resource Allocation - Use "Open" and "Close" on devices
- Device-independent block size - user programs do not (usually) need to know the size of disk blocks, etc.

User-space I/O Software

- I/O Libraries – common function such as `printf`, `scanf`, `fopen`, are library routines linked into the executable to make the appropriate system calls.
- Spooling – queues requests by multiple users for printer, email, etc., which are serviced by a daemon.

User-Space I/O Software



- Figure 3-8. Layers of the I/O system and the main functions of each layer.

Tutorial 4 (to be discussed Week 6)

- Q1. (revised TW 3.6) CD-quality music requires sampling the sound signal 44,100 times per second. Suppose a timer generates an interrupt at this rate, and that each interrupt requires 1 microsecond on a 1 GHz CPU. What is the slowest CPU clock that could be used and not lose any data? What sort of things might the interrupt handler do?
- Q2. (TW: 3.7) An alternative to interrupts is polling (eg., busy waiting). Are there circumstances you can think of in which polling is a better choice?

Tutorial 4 (to be discussed Week 6)

- Q3. Does an I/O call such as read() or write() to a disk file always cause the calling process to block? Why?
- Q4. (TW 3.2) Many disks contain a Checksum or an ECC (Error Correcting Code) at the end of each block. What might be taken if the Checksum or ECC indicated an error in the disk block?, and by which piece of hardware or software?

Summary

- Operating Systems and I/O
 - Provide a uniform interface to programmers
 - Hide ugly details, such as asynchronous I/O and interrupts
 - Provide a uniform interface for device drivers
 - Provide protection from I/O errors which could crash the system