Input/output (I/O)

Operating Systems

I/O and OS

- An operating system also controls all the computer's I/O devices
  - Disks, clocks, keyboards, displays, network interfaces
- Provides an interface between the devices and the rest of the system
  - This interface should be the same for all devices (wherever possible)
  - Device independent interfaces
- Issues commands to the devices, catch interrupts, and handle errors
- If I/O is not part of the OS, each application has to program that

I/O devices

- I/O devices can be *roughly* divided into two categories:
- **Block devices**
  - Stores information in fixed-size blocks, each one with its own address
  - All transfers are in units of one or more entire (consecutive) blocks
  - Each block can be written or read independently
  - Hard disks, Blu-ray discs, and USB sticks
- **Character devices**
  - Delivers or accepts a stream of characters, without any block structure
  - Not addressable and does not have any seek operation
  - Printers, network interfaces, keyboards

I/O devices

- I/O devices cover a huge range in speeds
- Most of these devices tend to get faster as time goes on
- Software must cope with varying speeds of such devices

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Scanner at 300 dpi	1 MB/sec
Digital camcorder	3.5 MB/sec
4x Blu-ray disc	18 MB/sec
802.11n Wireless	37.5 MB/sec
USB 2.0	60 MB/sec
FireWire 800	100 MB/sec
Gigabit Ethernet	125 MB/sec
SATA 3 disk drive	600 MB/sec
USB 3.0	625 MB/sec
SCSI Ultra 5 bus	640 MB/sec
Single-lane PCIe 3.0 bus	985 MB/sec
Thunderbolt 2 bus	2.5 GB/sec
SONET OC-768 network	5 GB/sec

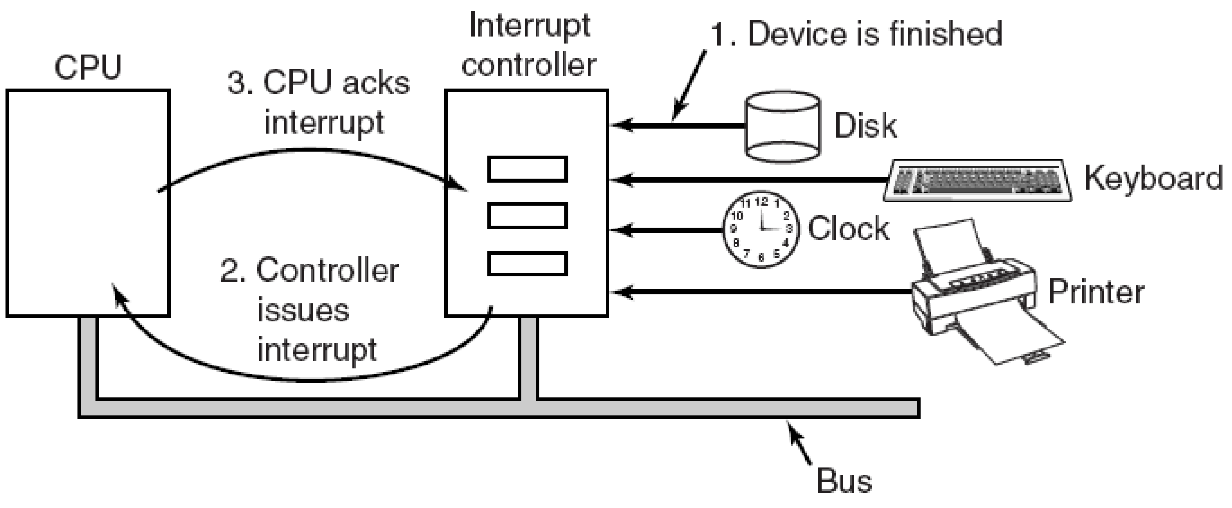
Device Controllers

- I/O devices consists of,
  - A mechanical component – the device
  - An electronic component – the device controller
- The device can usually be plugged into the controller card
- Many controllers can handle multiple identical devices
- The interface between the controller and device can be a standard
  - ANSI, IEEE, or ISO standard or a de facto one
  - SATA, SCSI, USB, Thunderbolt, or FireWire (IEEE 1394) for disc drives
- Without device controllers, operating system should be programmed for low level operations of the device

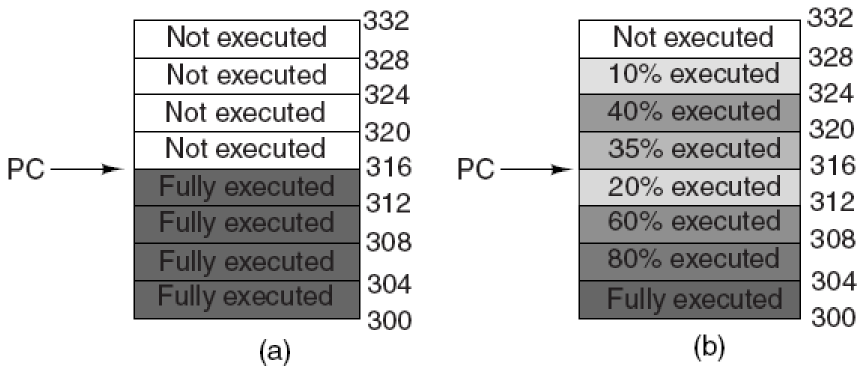
Interrupts

- When an I/O device has finished the work given to it, it causes an interrupt.
- It does this by asserting a signal on a bus line that it has been assigned.
- This signal is detected by the interrupt controller chip, which then decides what to do.

- The interrupt signal causes the CPU to stop what it is doing and start doing something else.



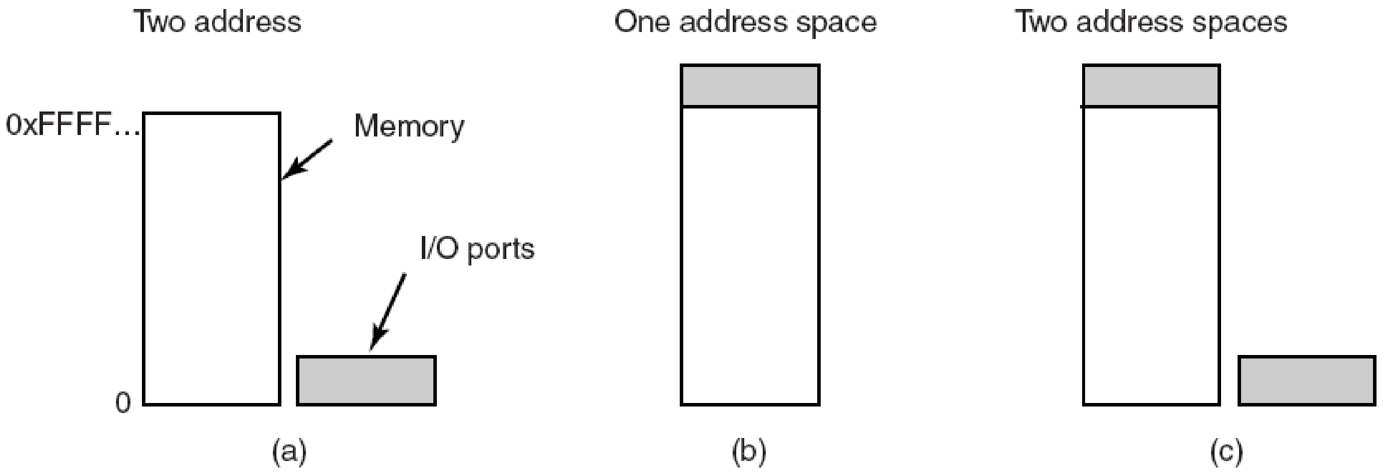
Precise Interrupts



- An interrupt that leaves the machine in a well-defined state is called a **precise interrupt**
- Such an interrupt has four properties:
    1. The PC (Program Counter) is saved in a known place.
    2. All instructions before the one pointed to by the PC have completed.
    3. No instruction beyond the one pointed to by the PC has finished.
    4. The execution state of the instruction pointed to by the PC is known.
- An interrupt that does not meet these requirements is called an **imprecise interrupt**. It is difficult for the operating system to figure out what has happened and what still has to happen.

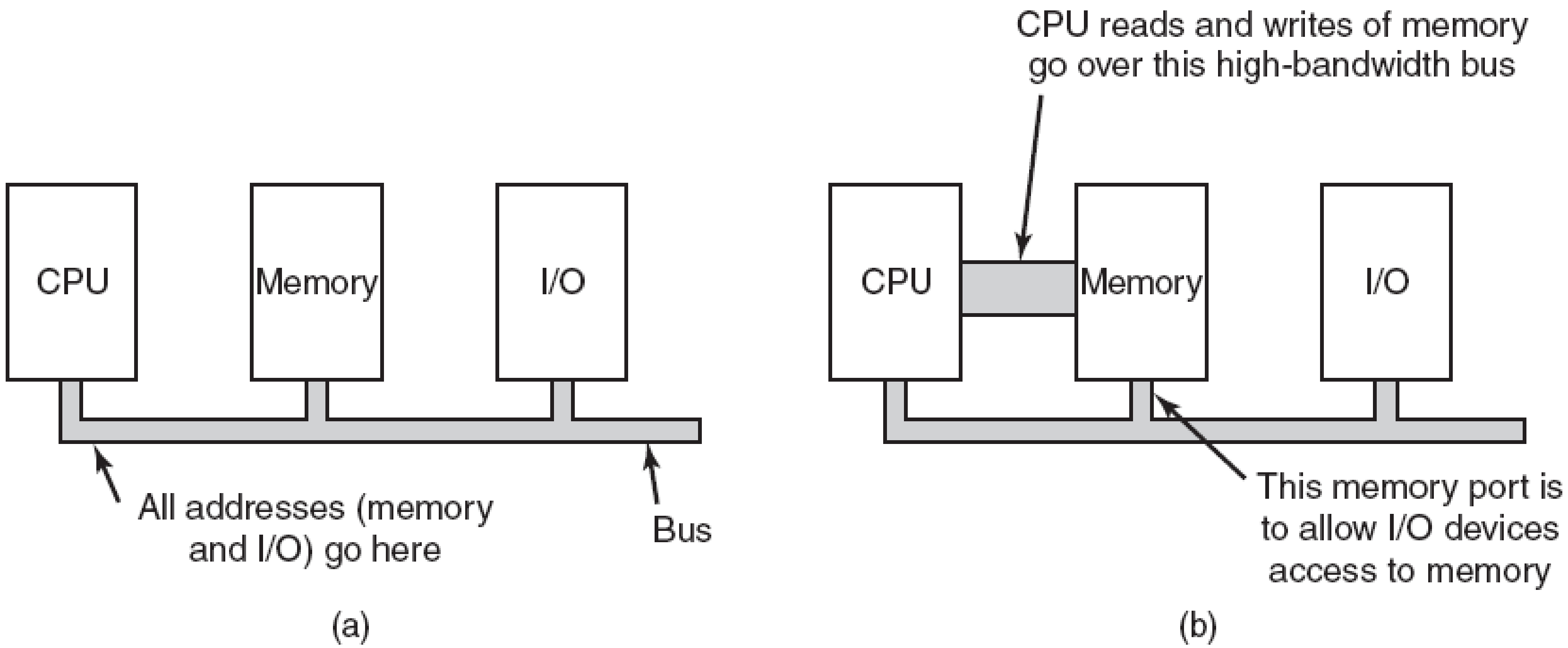
Memory-Mapped I/O

- Device controllers have registers – to command to perform actions
- Devices have data buffers – where programs or OS can read/write
- Two approaches to manage this space:
  a) address spaces for memory and I/O are different
  b) map all the control registers into the memory space – Memory-Mapped I/O



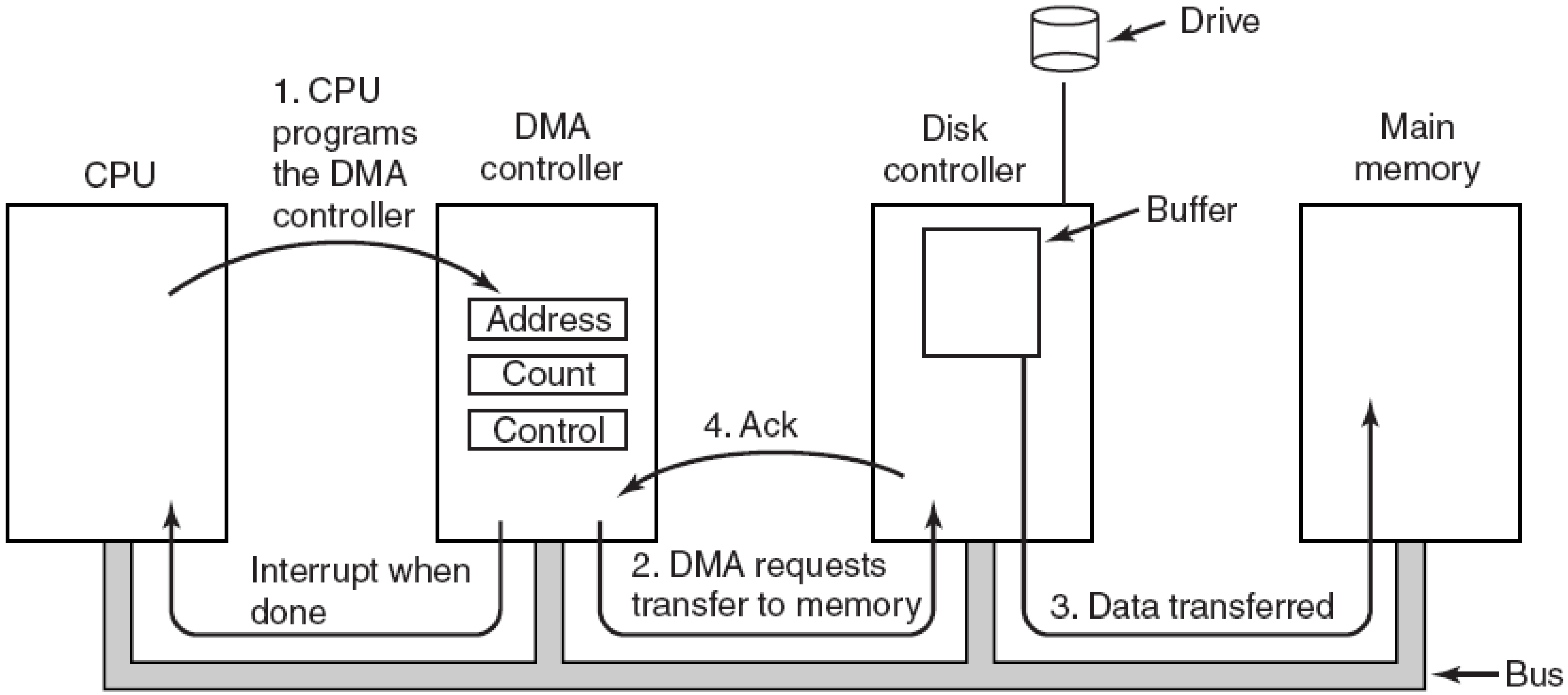
Memory-Mapped I/O

- Modern personal computers is to have a dedicated highspeed memory bus, tailored to optimize memory performance



Direct Memory Access (DMA)

- Managing I/O access for CPU and loading data directly into memory
- Regulating transfers to multiple devices, often concurrently



Goals of the I/O Software

- **Device independence**
  - We should be able to write programs that can access any I/O device without having to specify the device in advance.
  - *E.g. read a file on a hard disk, a DVD, or on a USB stick the same way*
- **Uniform naming**
  - The name of a file or a device should simply be a string or an integer and not depend on the device in any way.
  - *E.g. a USB stick can be mounted on top of the directory /usr/ast/backup*
- **Error handling**
  - Errors should be handled as close to the hardware as possible
  - *E.g. if the controller discovers a read error, it should try to correct the error*

Goals of the I/O Software

- **Synchronous** (blocking) vs. **asynchronous** (interrupt-driven) transfers
  - Blocking—after a read system call the program is automatically suspended until the data are available in the buffer.
  - Asynchronous—CPU starts the transfer and goes off to do something else until the interrupt arrives.
- **Buffering**
  - Used when data from/to device cannot be stored directly to final destination.
  - Involves considerable copying and has a major impact on I/O performance.
- **Sharable** vs. **dedicated devices**
  - Some I/O devices, such as disks, can be used by many users at the same time.
  - Other devices, such as printers, have to be dedicated to a single user until that user is finished.

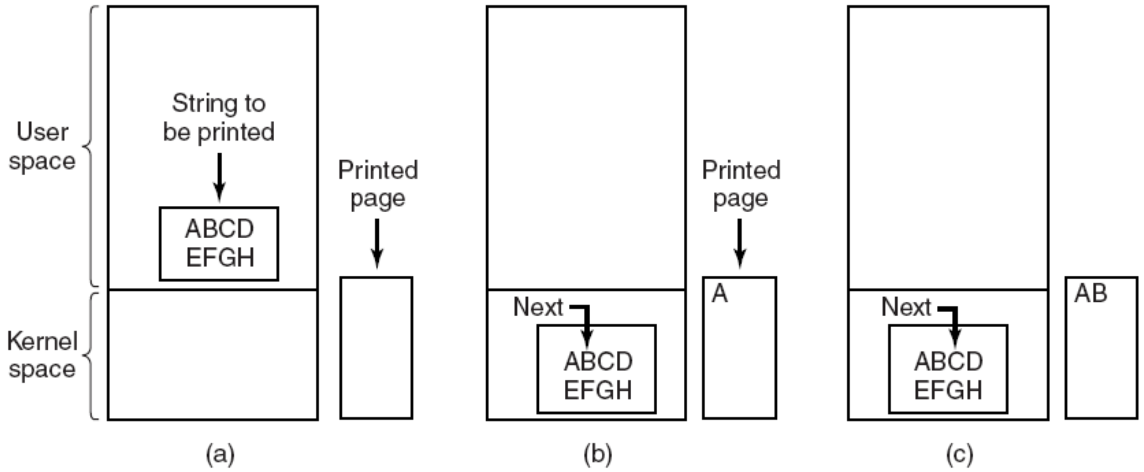
Types of I/O methods

1. Programmed I/O
2. Interrupt-driven I/O
3. I/O using DMA

Programmed I/O

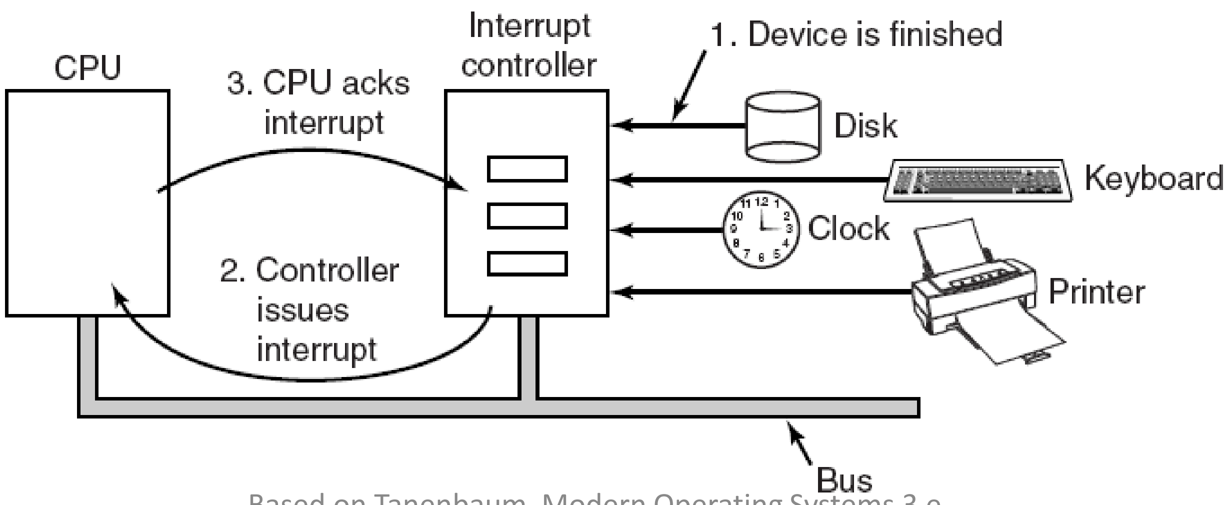
- CPU does all the work, simplest form of I/O
- Has the disadvantage of tying up CPU full time until all I/O is done
  - Busy waiting / polling
- Fine if waiting is short or CPU has nothing else to do
- In most complex systems, CPU has other things to do

- E.g.
  - First the data are copied to the kernel. Then the operating system enters a tight loop, outputting the characters one at a time.



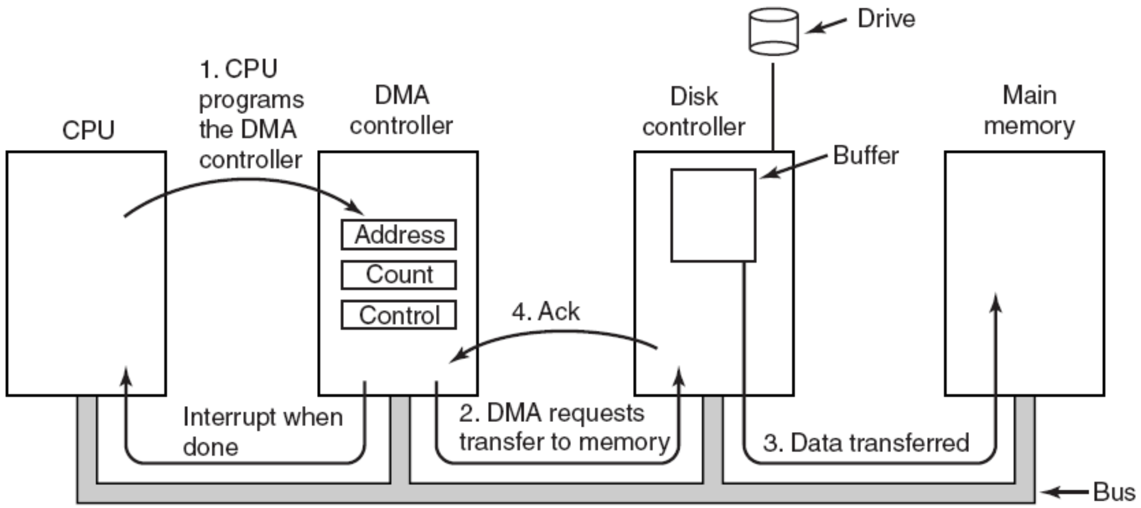
Interrupt-Driven I/O

- Allow the CPU to do something else while waiting for I/O.
- Whenever CPU is waiting for some I/O, it can switch to another process, until an interrupt is received from I/O on completion.
- However, too frequent interrupts within each I/O request can waste CPU time.



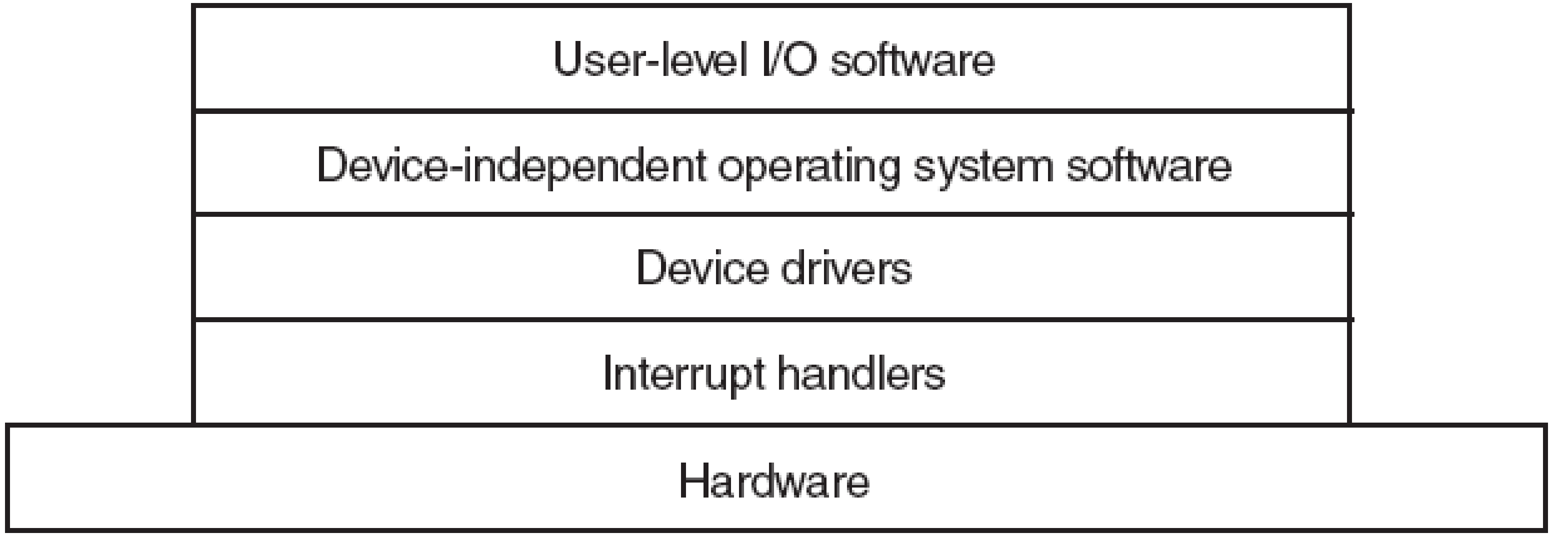
I/O Using DMA

- DMA controller manages the I/O request as a whole
- CPU is not interrupted within a I/O request
- Reduces the number of interrupts
- However, DMA controllers are usually much slower than main CPU.
  - So if CPU has nothing else to do, it may have to wait longer than if it did I/O on it's own.



I/O Software Layers

- Each layer has a well-defined function to perform.
- Each layer has a well-defined interface to the adjacent layers.
- The functionality and interfaces can differ from system to system.



Interrupt Handlers

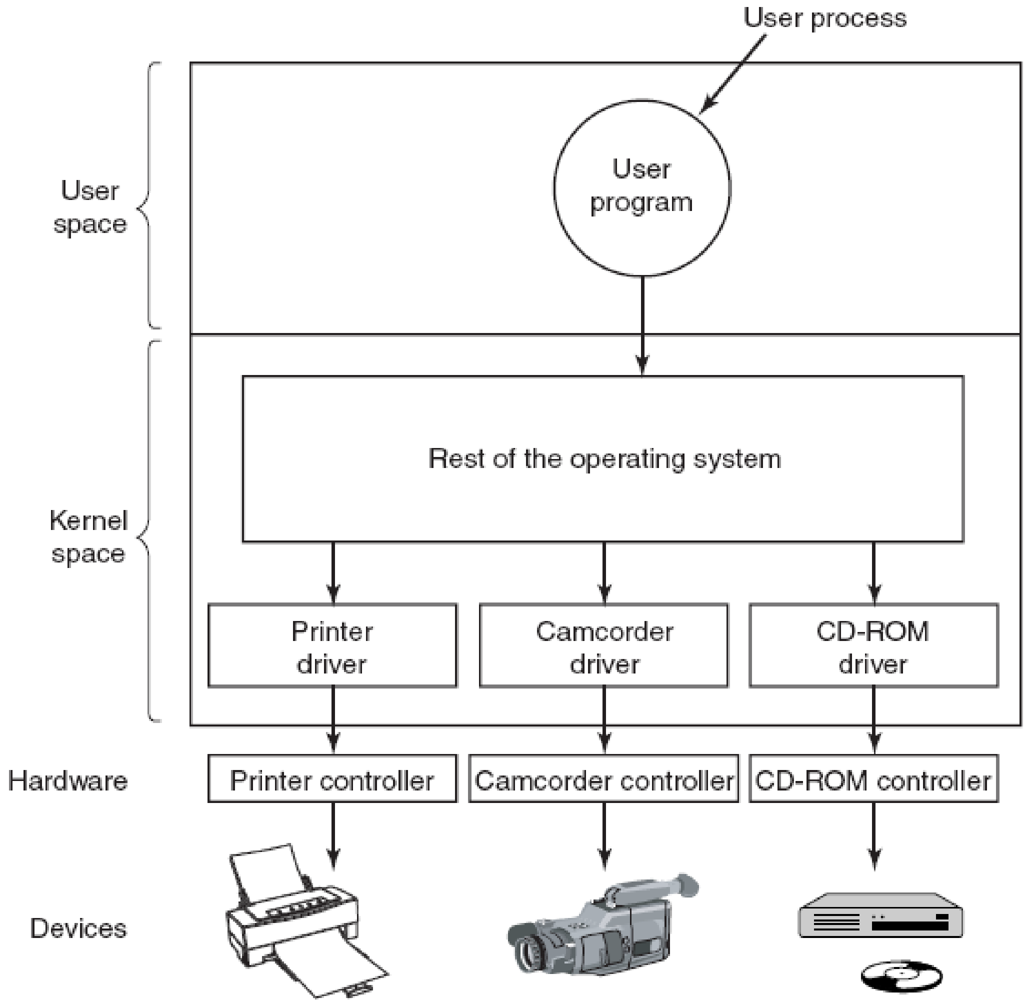
- Interrupts are unavoidable in I/O
- Interrupt processing is complex, involving multiple steps
- It takes a considerable number of CPU instructions
- It is best if they can be hidden away from other parts of the OS

Interrupt Handlers

1. Save any registers (including the PSW) that have not already been saved by the interrupt hardware.
2. Set up a context for the interrupt-service procedure. Doing this may involve setting up the TLB, MMU and a page table.
3. Set up a stack for the interrupt service-procedure.
4. Acknowledge the interrupt controller. If there is no centralized interrupt controller, reenable interrupts.
5. Copy the registers from where they were saved (possibly some stack) to the process table.
6. Run the interrupt-service procedure. It will extract information from the interrupting device controller's registers.
7. Choose which process to run next. If the interrupt has caused some high-priority process that was blocked to become ready, it may be chosen to run now.
8. Set up the MMU context for the process to run next. Some TLB setup may also be needed.
9. Load the new process' registers, including its PSW.
10. Start running the new process.

Device Drivers

- The piece of program that translates between the programmer's interface (read, write, seek) and the hardware interface is called a **device driver**.
- The device driver usually operates as part of the operating system kernel.
- In early OS (including first versions of Unix) all device drivers had to be compiled together with the kernel.
- Later systems allow device drivers to be loaded.



Logical positioning of device drivers.
In reality all communication between drivers and device controllers goes over the bus.

Device Drivers

- A device driver has several functions.
- They accept abstract read and write requests from the device-independent software above it and ensure that they are carried out.
- They also need to perform other functions such as initialising devices, manage power requirements and logging events.
- In a hot-pluggable system, devices can be added or removed while the computer is running.
- Driver must manage any events of sudden removal of devices.

Device-Independent I/O Software

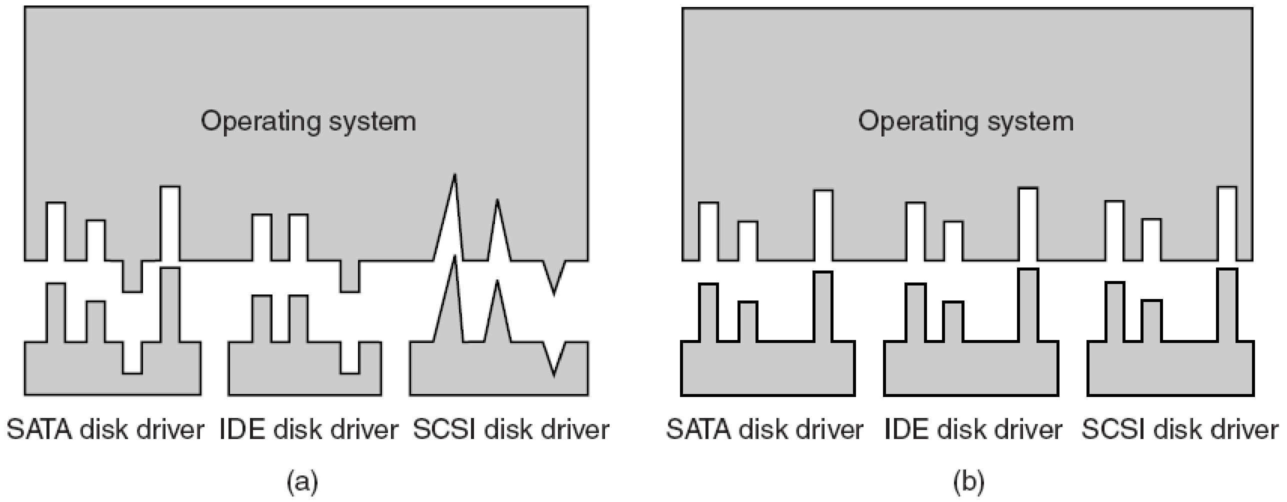
- Although some of the I/O software is device specific, other parts of it are device independent.
- Device-independent software performs the I/O functions that are common to all devices
- It provides a uniform interface to the user-level software.

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

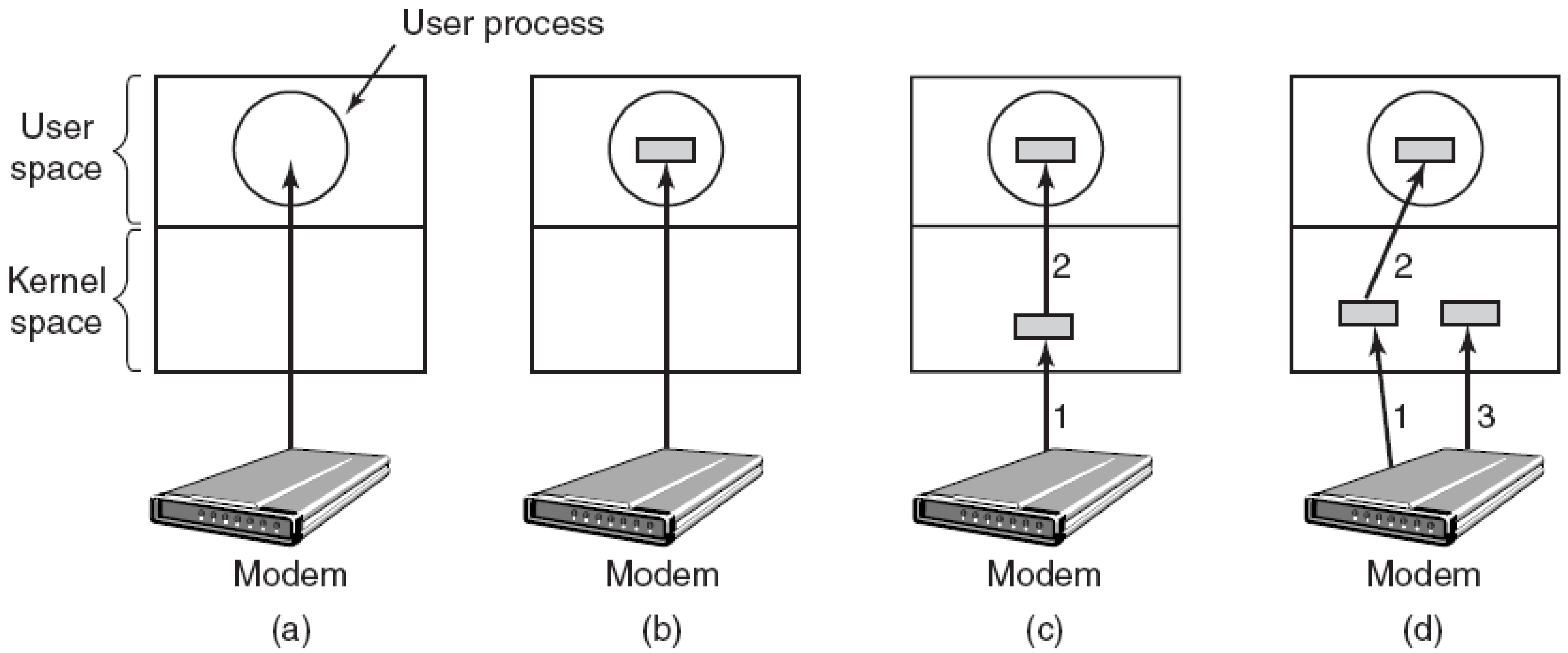
Uniform Interfacing for Device Drivers

a) If each device driver has a different interface to the operating system, every time a new device comes along, the operating system must be modified.

b) If all drivers have the same interface, new drives can be plugged in easily.
   - OS defines a set of functions a driver must supply for a class of devices
   - The device-independent software maps symbolic device names onto the proper driver



Buffering



a) Unbuffered input.
b) Buffering in user space.
c) Buffering in the kernel followed by copying to user space.
d) Double buffering in the kernel.

Error Reporting

- Errors are far more common in the context of I/O than in others.
- The operating system must handle errors as best it can.
- Device specific errors must be handled by device drivers.
- Programming errors
  - This occurs when a process asks for something impossible, such as writing to an input device.
- Actual I/O errors
  - For example, trying to write a disk block that has been damaged
  - If the driver does not know what to do, it may pass the problem back up to device-independent software.

Allocating and Releasing Dedicated Devices

- Some devices, such as printers, can be used only by a single process at any given moment.
- A simple way to handle these requests is to require processes to perform opens on the special files for devices directly.
  - If the device is unavailable, the open fails.
  - Closing such a dedicated device then releases it.
- An alternative approach is to have special mechanisms for requesting and releasing dedicated devices.

Device-Independent Block Size

- Different disks may have different sector sizes.
- It is up to the device-independent software to hide this fact and provide a uniform block size to higher layers.
- In this way, higher layers deal only with abstract devices that all use the same logical block size, independent of the physical sector size.

User-Space I/O Software

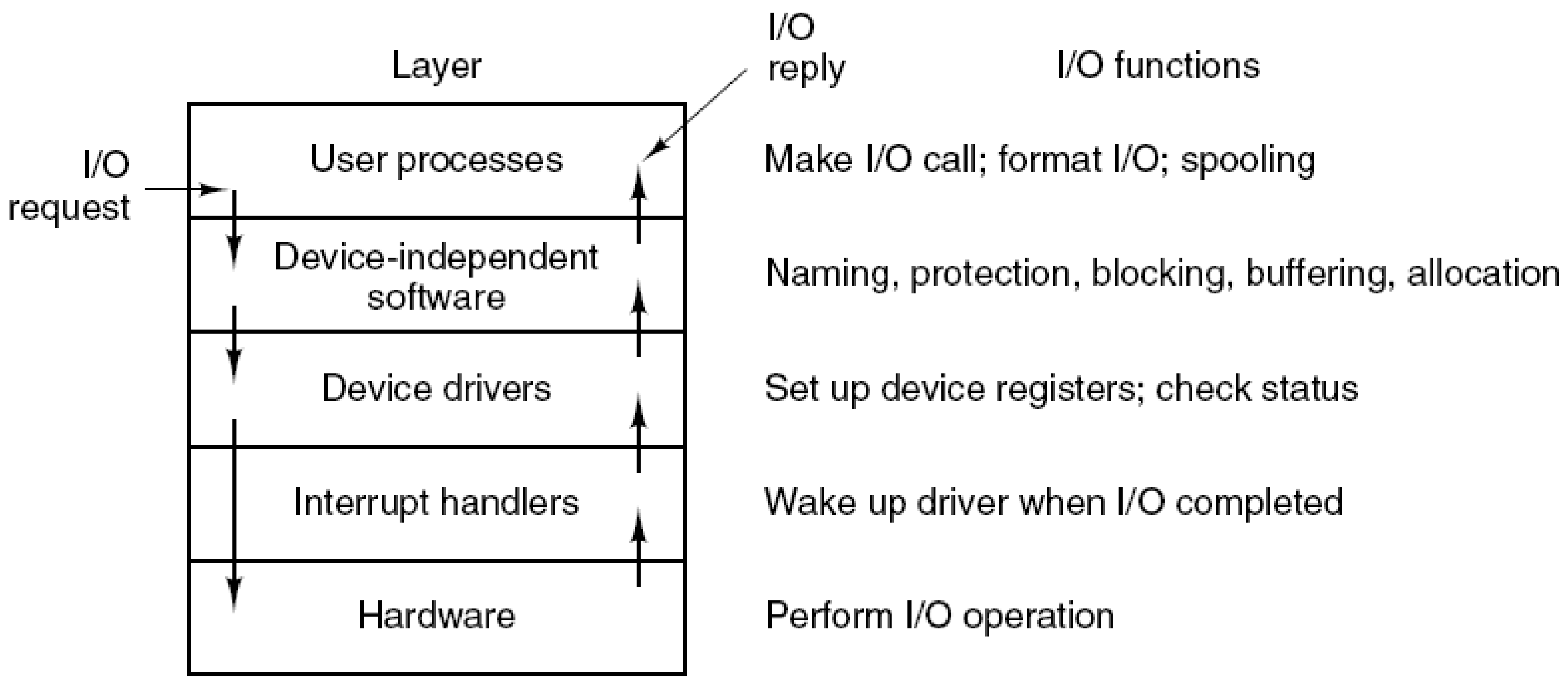
- Most of the I/O software is within the operating system.
- A small portion of it consists of libraries linked together with user programs.
- System calls, including the I/O system calls, are normally made by library procedures.

Spooling system

- Not all user-level I/O software consists of library procedures.
- **Spooling** is a way of dealing with dedicated I/O devices in a multiprogramming system.
- Uses a special process, called a **daemon**, and a special directory, called a **spooling directory**.
- E.g. Printing
  - A process first generates the entire file to be printed and puts it in the spooling directory.
  - Printer daemon prints the files in the directory.

Summary of I/O Software

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



Online Quiz via LMS

- June 20, Saturday, 12:00 AM – 11:59 PM
- 24 hour timeframe to complete
- 1 attempt only, 20 minutes per attempt
- 10 multiple choice questions, from randomised question bank
- Questions from:
  - 04 Memory Management
  - 05 File Systems
  - 06 Input Output
- **Students with access issues must contact PRIOR TO START OF QUIZ for alternative arrangements**