Input / Output Subsystem

Using Tanenbaum's
Modern Operating Systems (3rd edition)

Basic Terms

- Significant part of the OS.
- Command to devices / catch interrupts.
- Interface between the devices and the rest of the system.
- Same interface to all devices (device independence).
- To view devices on Linux:
  - `lspci`: PCI devices.
  - `lsusb`: USB devices

Principles of I/O HW

Types of I/O Devices

- Block devices
  – Addressable blocks of fixed size.
  – HDD, CD-ROM, USB,…

- Character devices
  – A stream of un-addressable characters.
  – Mouse, keyboard, network interfaces.

- Other devices
  – HW clock.

Block Device

- Long access time (measured in milliseconds).
- Fast data transfer (using DMA).
- Using caching and read ahead.
- E.g. disk minimal transfer unit is a sector:
  - Best when identical to page size.

Character Device

- No buffering is needed.
- For example:
  - Every mouse moment generates an interrupt

```
$ lsusb
Bus 008 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 007 Device 002: ID 04b3:310c IBM Corp. Wheel Mouse
$ sudo cat /dev/usbmon7
```

Device Controller

- The electronic part of the device.
- Maps bus traffic to device specific low-level commands:
  - CPU sets values in the devices registers via the BUS.
  - The controller transfer commands to the device.
  - When the device finishes, the controller issues an interrupt.
- Error checking.

I/O Memory Space

Communications with I/O Device

- Each controller has its registers that used to communicate with the CPU.
- Some controllers have also separate data buffers (e.g. graphics).
- Two alternatives for communications:
  - Port-mapped I/O.
  - Memory-mapped I/O.

IO Ports & Memory-mapped IO

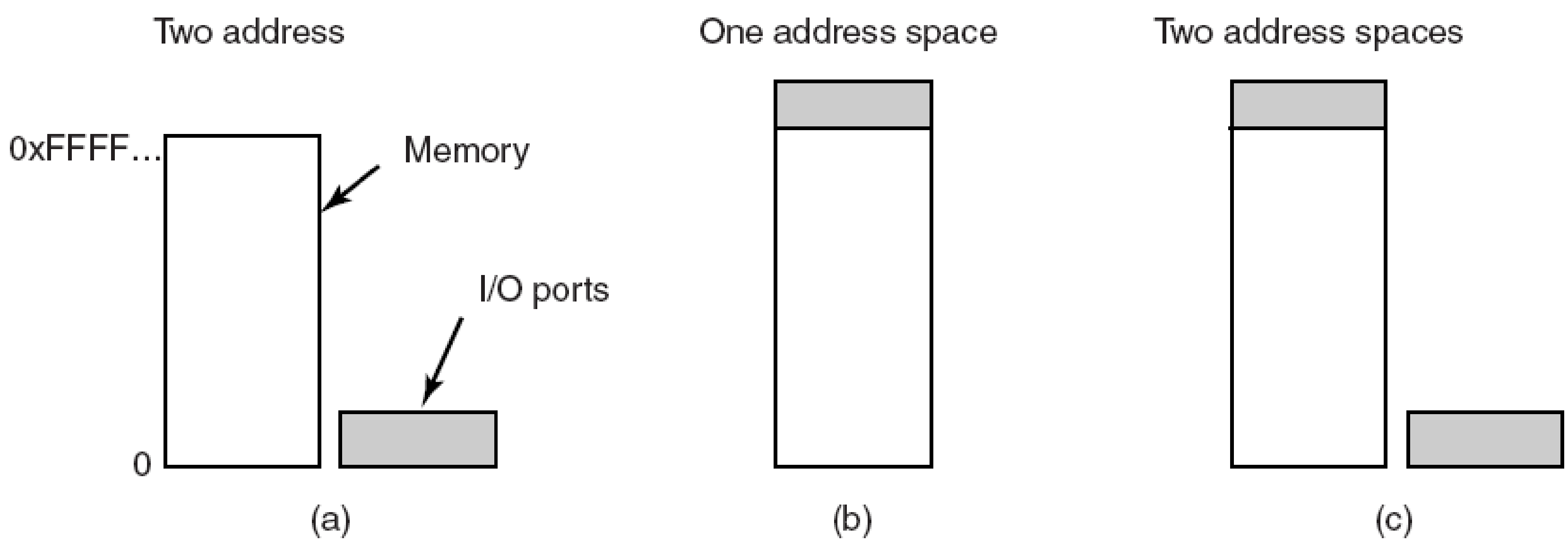


Figure 5-2. (a) Separate I/O and memory space.
(b) Memory-mapped I/O. (c) Hybrid.

Port-mapped IO

- Each control register is assigned an I/O port number from the I/O port space (e.g. 64K).

- Special machine (assembly) commands are needed:
  - IN REG#, PORT#
  - OUT PORT#, REG#

- When bus is used for I/O it is called IO bus:
  - Data bus used to deliver data.
  - Control bus to deliver commands.
    - On 8086 16-bit control bus supports I/O port space of 64K.

Memory-mapped IO

- Control registers are mapped at the top of the address space.

- Read/Write to the device's shared memory (on device data buffer) using memory commands:
  - MOV REG#, ADDR
  - MOV ADDR, REG#

- Pros/Cons:
  - May be written in C.
  - All memory instructions are usable.
  - Must take care not to use caching (Why?)

Polling vs. Interrupt

Two I/O Operation Modes

Polling Mode:

- Synchronous.
- Simple code.
- I/O is slow – device driver keeps the CPU busy.

Interrupt mode

- Asynchronous.
- Difficult to write good multi-threaded interpretable code.
- OS makes asynchronous devices looks synchronous via blocking system calls (i.e. *read()*).
  - Blocking operation calls *sleep_on(device)*
  - completion routine calls *wake_up(device)*

Interrupt Flow (I)

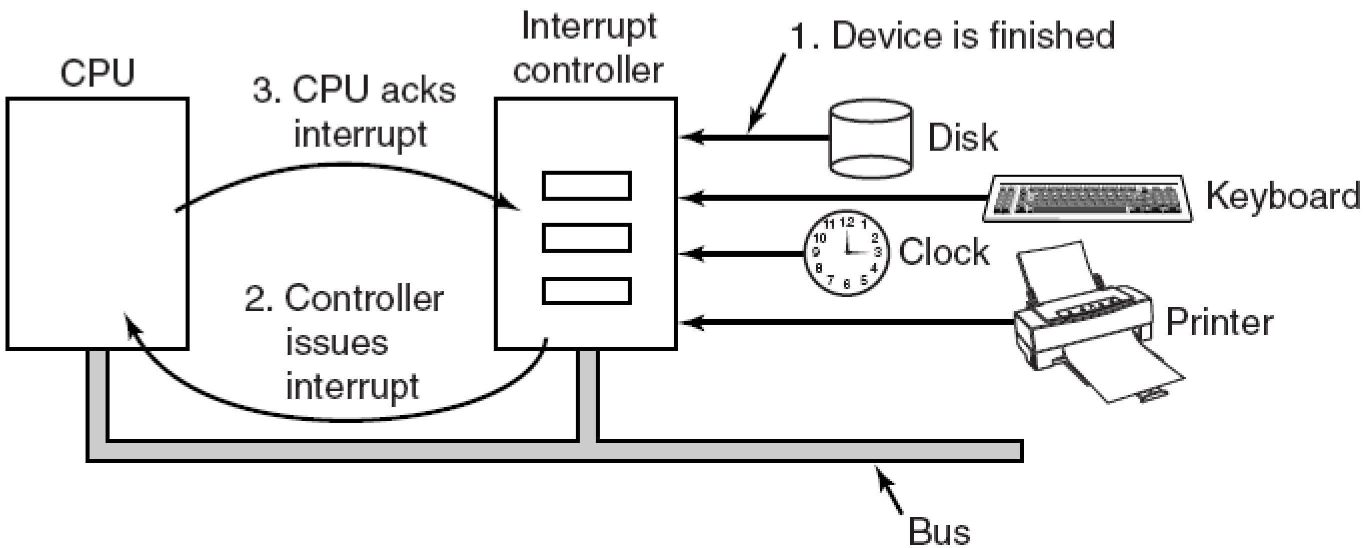
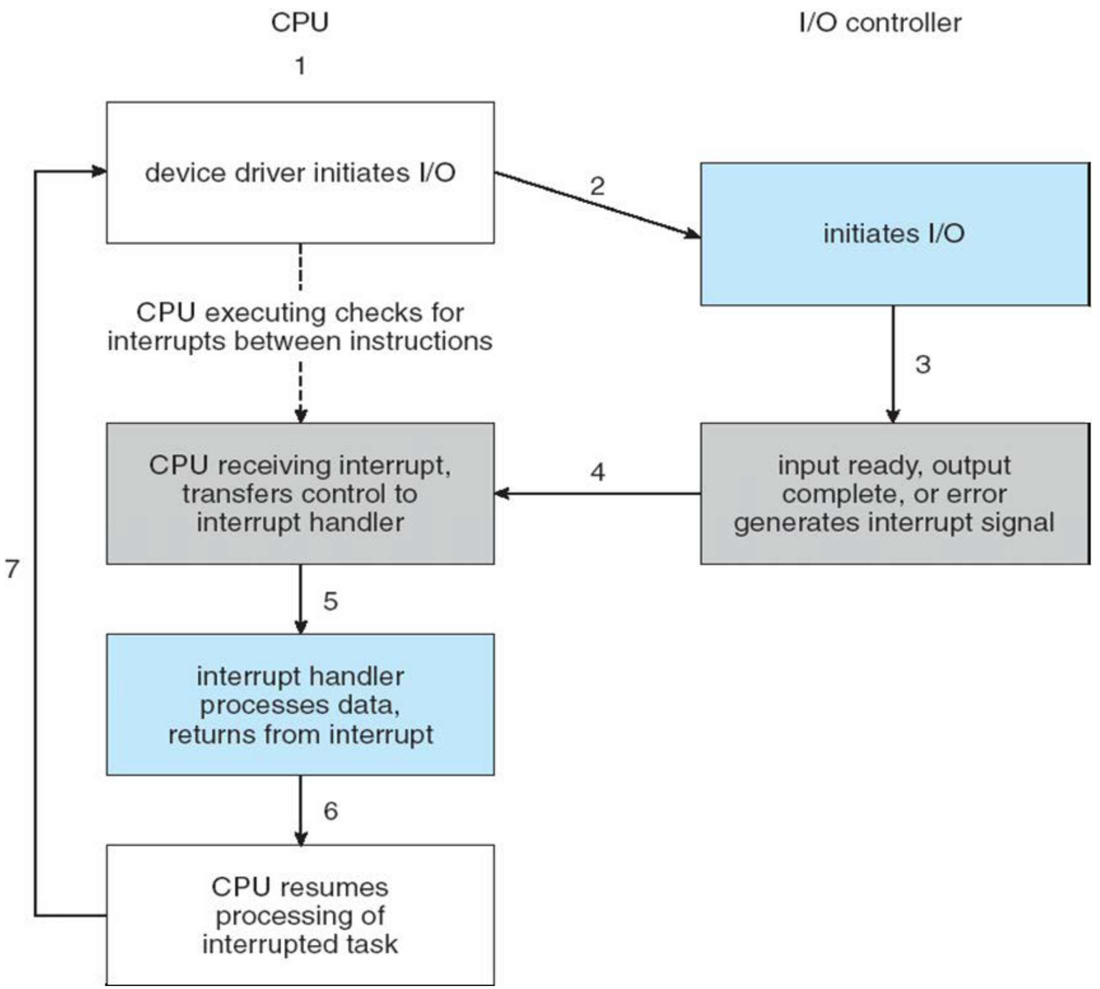


Figure 5-5. How an interrupt happens. The connections between the devices and the interrupt controller actually use interrupt lines on the bus rather than dedicated wires.

Interrupt Flow (II)



Interrupt Flow (II)

1. Interrupts are issued by interval timers and I/O devices:
   - The arrival of a keystroke from a user sets off an interrupt.
   - When device finishes the work assigned to it, it causes an interrupt on the bus.
2. The interrupt is detected by the interrupt controller
   - If another interrupt is in process – it may be ignored
3. The interrupt value on the bus is used as index into the Interrupt Vector
4. Each entry in the vector has pointer to an Interrupt Handler (a.k.a. Interrupt Service Procedure)
5. The interrupt handler operation:
   - Extracts information from the device controller registers.
   - Acknowledges the device by writing to the controller’s I/O ports.
6. CPU Resumes the interrupted task

Two Parts of the Interrupt Handler

- Interrupt handlers need to finish up quickly and not to keep interrupts blocked for long time.

- Interrupt handler is split into two "halves" :
  - Top-half is registered with `request_irq`
    - Moves data to/from a device-specific buffer.
  - Bottom-half is scheduled by the top-half to be executed later
    - Awakening of the processes, starting up another I/O operation.
    - Takes place on free cycles.

- Interrupts are enabled during execution of the bottom half.

An Example: Printing a string

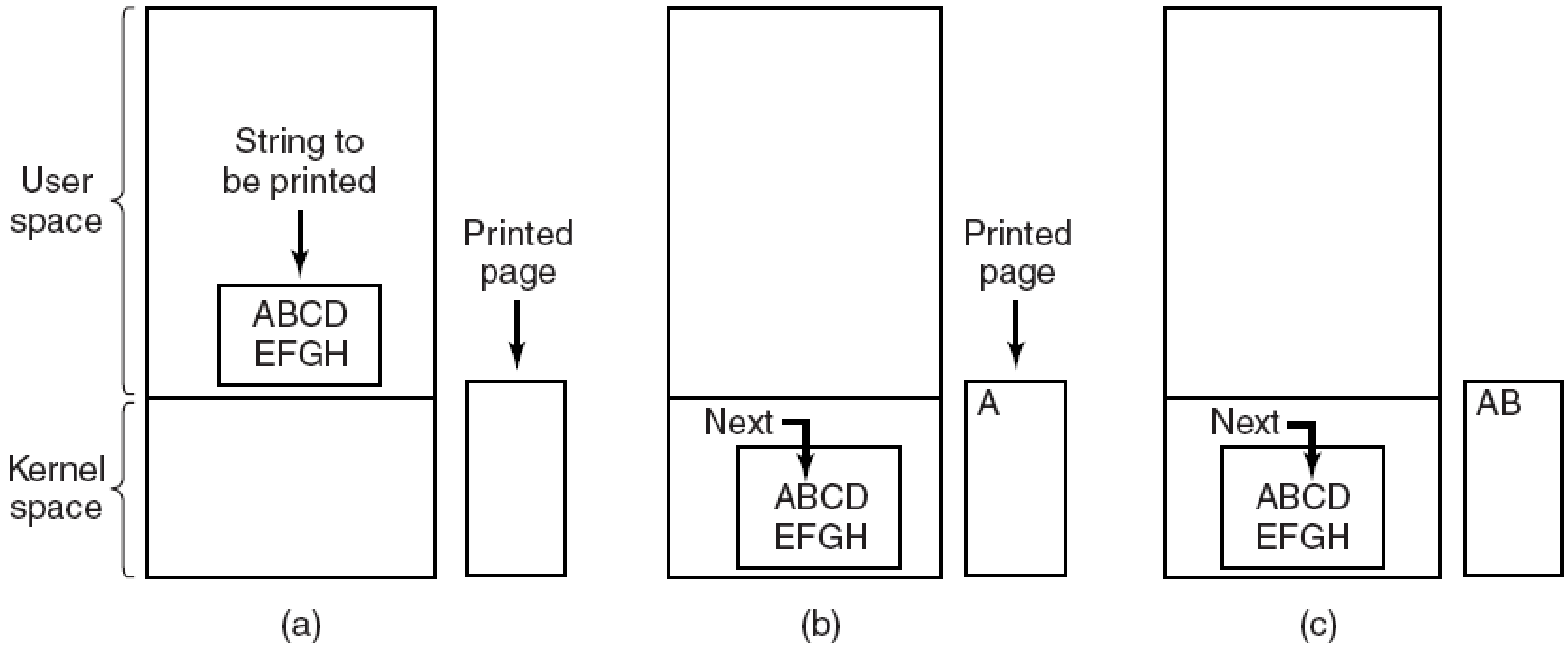


Figure 5-7. Steps in printing a string.
(a) The user is calling a print operation.
(b) The buffer is copied to the kernel space for faster access
(c) The buffer contents are sent to the printer device

I/O Using Polling

```
copy_from_user(buffer, p, count);                    /* p is the kernel buffer */
for (i = 0; i < count; i++) {                        /* loop on every character */
        while (*printer_status_reg != READY) ;      /* loop until ready */
        *printer_data_register = p[i];               /* output one character */
}
return_to_user();
```

Figure 5-8. Writing a string to the printer using programmed I/O.
User polls the device until ready

Interrupt-driven I/O

```
copy_from_user(buffer, p, count);
enable_interrupts( );
while (*printer_status_reg != READY) ;
*printer_data_register = p[0];
scheduler( );
```

(a)

```
if (count == 0) {
    unblock_user( );
} else {
    *printer_data_register = p[i];
    count = count – 1;
    i = i + 1;
}
acknowledge_interrupt( );
return_from_interrupt( );
```

(b)

Figure 5-9. Writing a string to the printer using interrupt-driven I/O.
(a) Code executed at the time the print system call is made.
(b) Interrupt service procedure for the printer.

Direct Memory Access – DMA

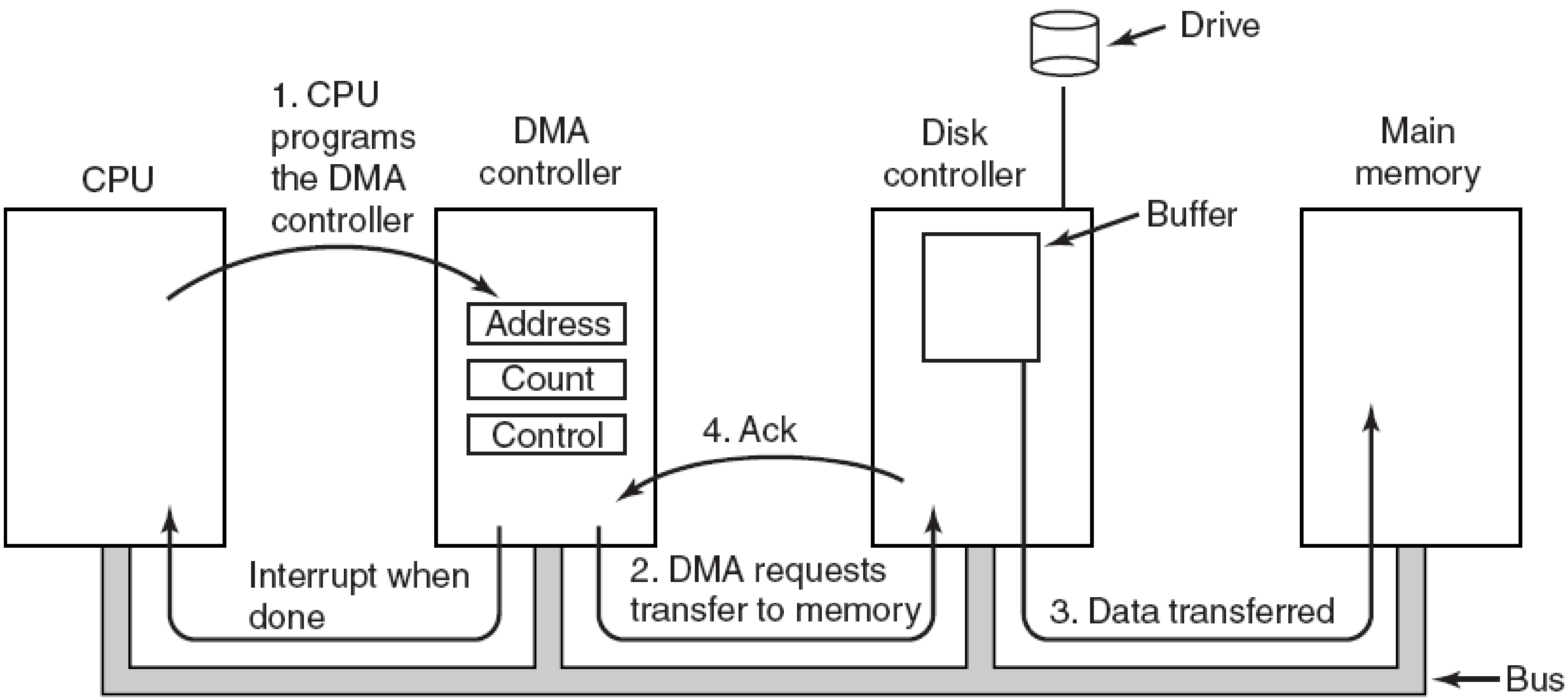


Figure 5-4. Operation of a DMA transfer.

DMA (II)

- DMA has access to the system bus independent of the CPU.

- DMA elements:
  - Memory address register.
  - Byte count register.
  - Control registers (1-2):
    - Specify the I/O port to use.
    - The transfer direction.

DMA (III)

Transfer w/o DMA (polling)

1. The device controller reads from disk bit by bit.
2. When its internal buffer is full – computes the checksum.
3. Cause an interrupt.
4. Data is read from the buffer – one byte or word at a time.

Transfer with DMA

1. CPU sets the DMA registers.
2. The DMA initiate a read command over the bus to the disk controller.
3. Loop on the read command till count is zero.
4. Interrupt the CPU – transfer is completed.

DMA (IV)

1. The CPU programs the DMA to copy entire buffer.
2. The DMA interrupts the CPU when done.
3. The DMA is slower than the CPU.
4. If the CPU is not busy, than DMA reduces performance.

```
copy_from_user(buffer, p, count);
set_up_DMA_controller( );
scheduler( );
```

(a)

```
acknowledge_interrupt( );
unblock_user( );
return_from_interrupt( );
```

(b)

Figure 5-10. Printing a string using DMA.
(a) Code executed when the print system call is made.
(b) Interrupt service procedure.

/proc

- Interrupts are shown in `/proc/interrupts`.
- Linux kernel tries to divide interrupt traffic evenly across the processors.
- `/proc/stat` records several low-level statistics about system activity:
  - The number of interrupts received since system boot.
- The first number is the total of all interrupts, while each of the others represents a single IRQ line, starting with interrupt 0.

Device Driver Interfaces

I/O Software Layers

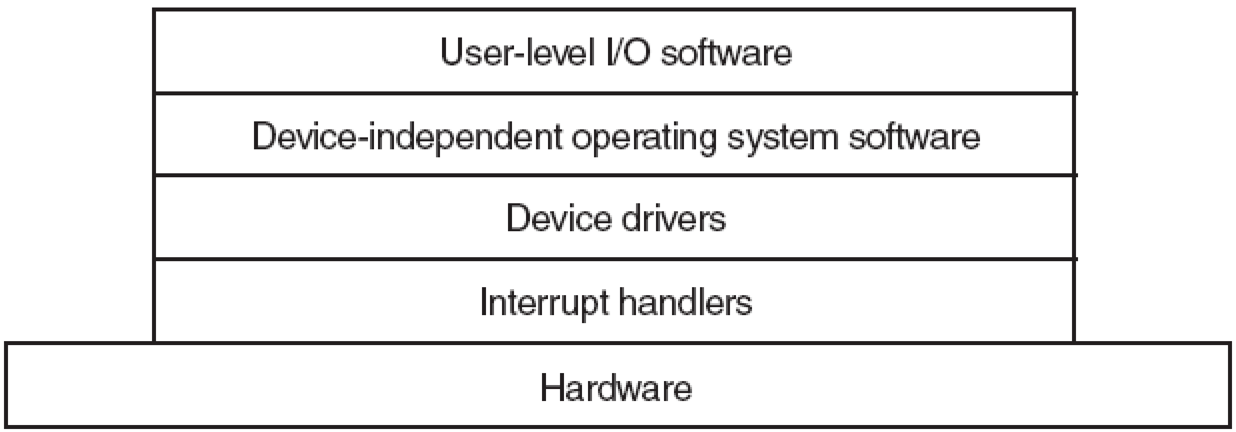


Figure 5-11. Layers of the I/O software system.

Device Drivers

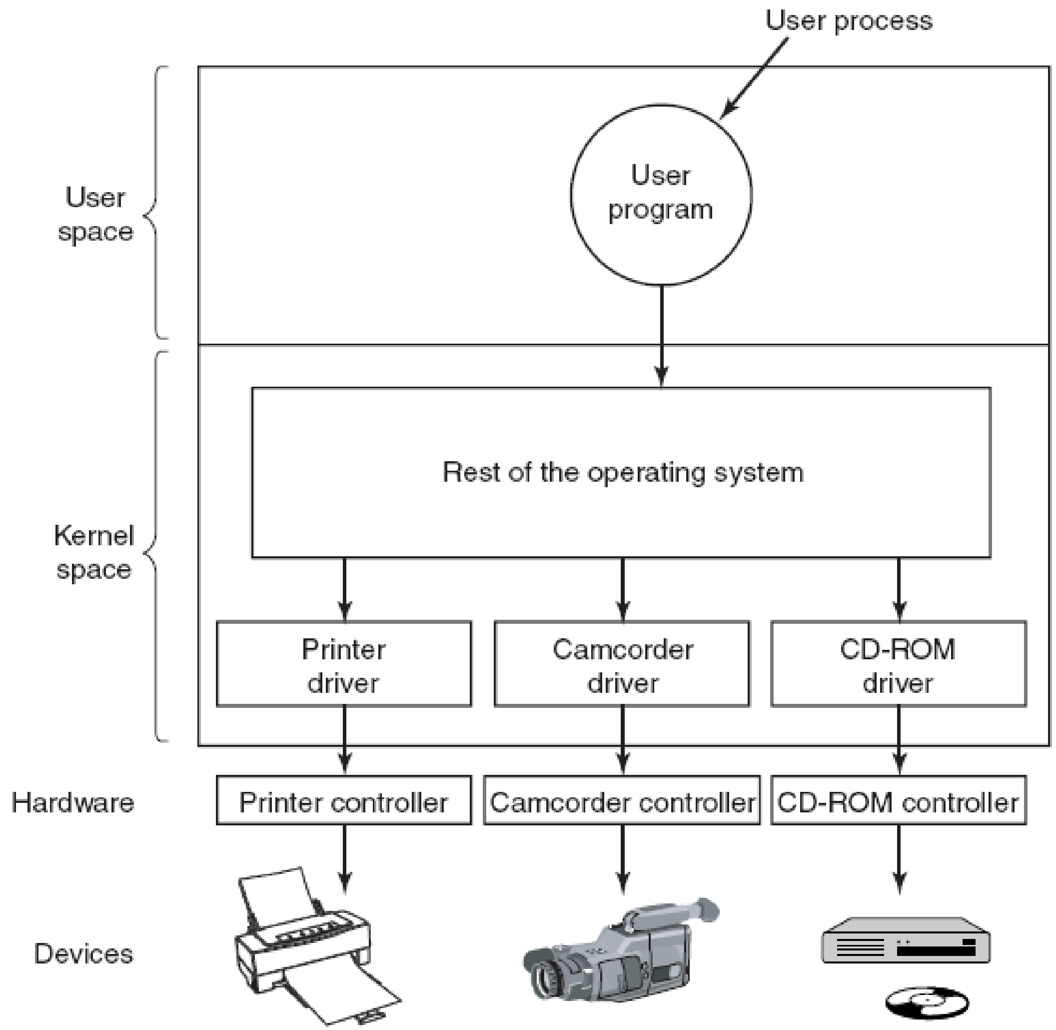


Figure 5-12. Logical positioning of device drivers.

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

Device-Independent Drivers

- Supply uniform interface for:
  - Buffering.
  - Error reporting.
  - Allocating and releasing devices.
  - Block size.
  - Device naming.

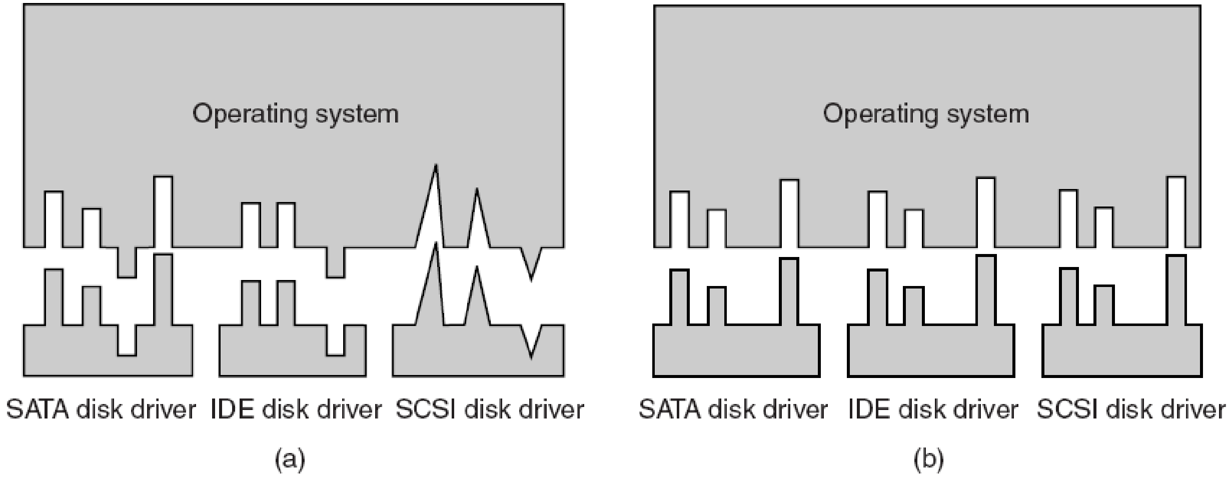


Figure 5-14. (a) Without a standard driver interface. (b) With a standard driver interface.

Interrupt Handler

- A device driver initiates a call, and than blocks itself on some IPC (semaphore, condition variable).

- When the device is ready for next interaction with the device driver it issues an interrupt.

- The interrupts handler unblocks the driver.

Device Drivers

- Device controllers has registers that are used to give it commands and read its status.
- The nature and number of this registers vary greatly between different devices:
  - HDD needs to know about sector, tracks, cylinders, heads, arm motion, motor drives…
  - Mouse…
- The manufacturer is writing the device driver, that maps the particular needs to an abstract interface of particular OS.

I/O System Layers

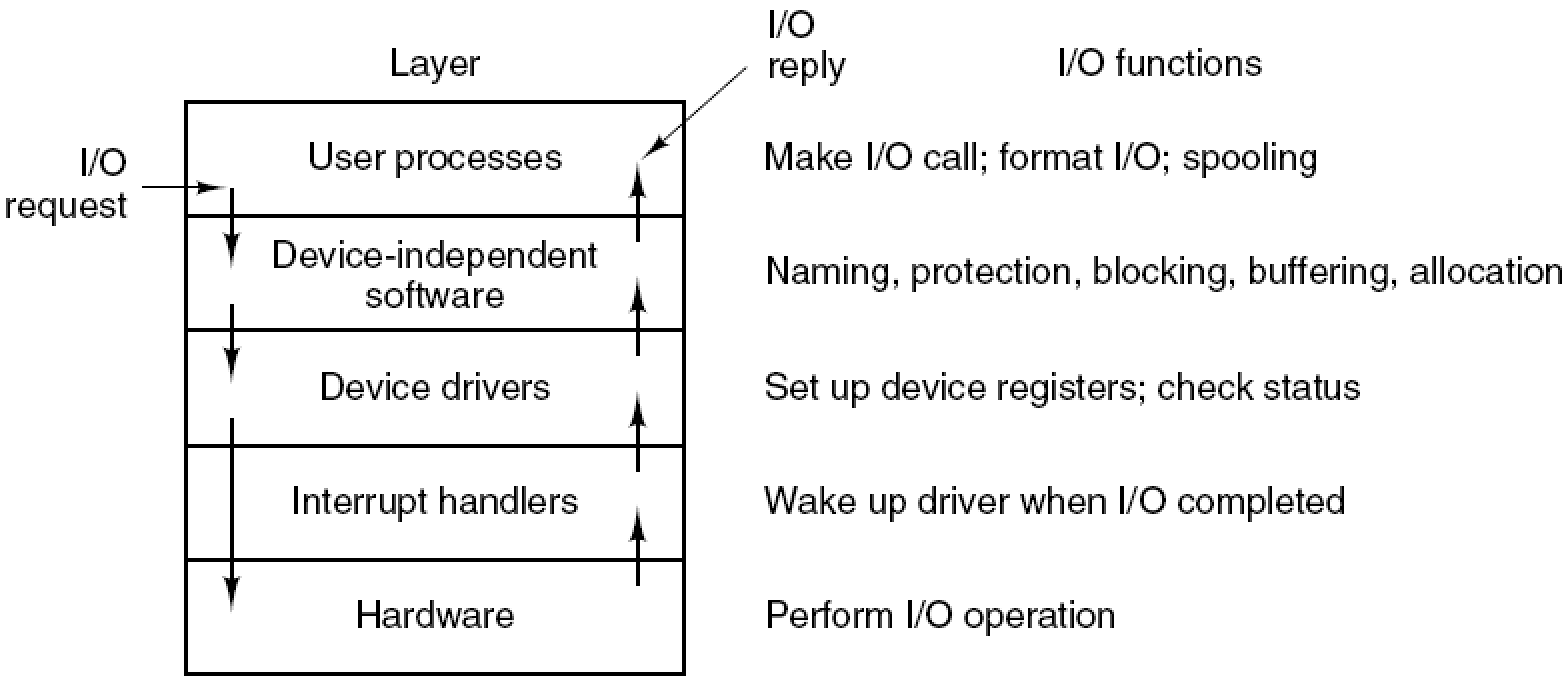


Figure 5-17. Layers of the I/O system and the main functions of each layer.

Clocks

Clock Hardware

- Clocks are neither block nor character device.
- Crystal may vibrate at a GHz, causing an interrupt when the counter reaches zero.
- These periodic interrupts are called – **clock ticks**.

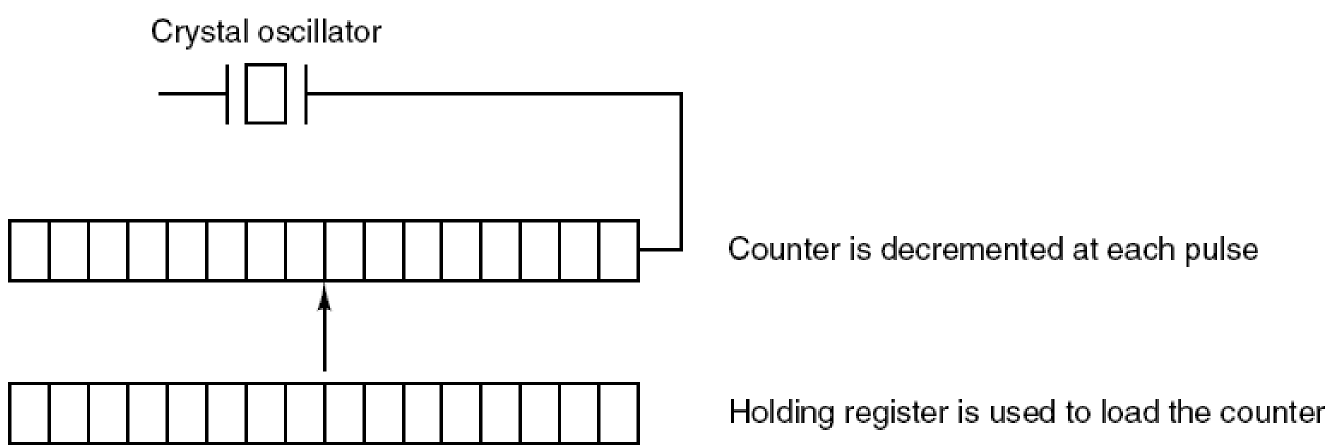


Figure 5-32. A programmable clock.

Programmable Clocks

- Programmable clocks allow to set the counter value.
- Low power circuit and battery keeps the time clock alive on power down.
- Unix counts ticks since Jan, 1, 1970 UTC time (formally know a Greenwich mean rime).

Clock Uses

- Maintaining time of day.
- Implement time quantum.
- Keeps track of CPU usage:
  - Use a second timer that count only when the process is running.
- Alarm system call:
  - Used for timeouts.
  - Sends SIGALRM.
- Watchdog timers for the system:
  - Start spinning the HD and wait prior to using it.
- Profiling, monitoring and statistics.

Homework + Interview Questions

- Explain in details the flow of the interrupt
- Explain the following terms:
  - DMA
  - Memory-mapped IO
  - IO port
  - Device driver
  - Interrupt handler