Operating Systems

Christopher Kruegel
Department of Computer Science
UC Santa Barbara
http://www.cs.ucsb.edu/~chris/

Input and Output

Input/Output Devices

- The OS is responsible for managing I/O devices
  - Issue requests
  - Manage corresponding interrupts
- The OS provides a high-level, easy-to-use interface to processes
- The interface, in principle, should be as uniform as possible
- The I/O subsystem is the part of the kernel responsible for managing I/O
- Composed of a number of *device drivers* that deal directly with the hardware

I/O Devices

- Two categories:
  - Block devices
    - Store information in blocks of a specified size
    - Block can be accessed (read or written) independently
    - Example: disk
  - Character devices
    - Deal with a stream of characters without a predefined structure
    - Characters cannot be addressed independently
    - Example: mouse, printer, keyboard

- Classification not perfect
  - Example: Clocks

Device Data Rates

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Telephone channel	8 KB/sec
Dual ISDN lines	16 KB/sec
Laser printer	100 KB/sec
Scanner	400 KB/sec
Classic Ethernet	1.25 MB/sec
USB (Universal Serial Bus)	1.5 MB/sec
Digital camcorder	4 MB/sec
IDE disk	5 MB/sec
40x CD-ROM	6 MB/sec
Fast Ethernet	12.5 MB/sec
ISA bus	16.7 MB/sec
EIDE (ATA-2) disk	16.7 MB/sec
FireWire (IEEE 1394)	50 MB/sec
XGA Monitor	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec
Sun Gigaplane XB backplane	20 GB/sec

Device Controllers

- I/O devices typically have two components
  - Mechanical component
  - Electronic component (e.g., connected to the mechanical component through a cable)

- The electronic component is the *device controller*
  - Often a PCI/ISA card installed on the motherboard (host adapter)
  - May be able to handle multiple devices (e.g., daisy chained)
  - May implement a standard interface (SCSI/EIDE/USB)

- Controller's tasks
  - Convert serial bit stream to block(s) of bytes (e.g., by internal buffering)
  - Perform error correction as necessary
  - Make data available to CPU/memory system

Accessing the Controller

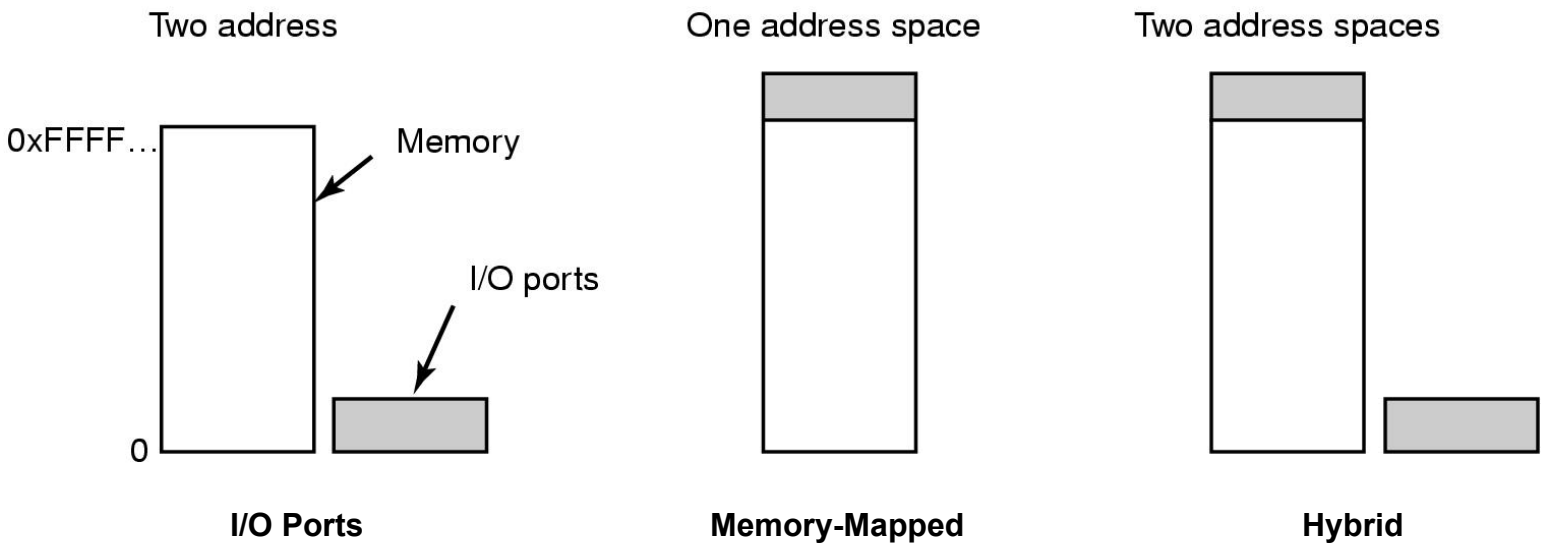
- The OS interacts with a controller
  - By writing/reading registers (command/status)
  - By writing/reading memory buffers (actual data)

- Registers can be accessed through dedicate CPU instructions
  - Registers mapped to I/O ports
  - IN REG, PORT and OUT REG, PORT
    transfer data from CPU's registers to a controller's registers

- Registers can be mapped onto memory (Memory-Mapped)

- Hybrid approach
  - Registers are accessed as I/O ports
  - Buffers are memory mapped
  - Used by the Pentium (640K-1M mem-mapped buffer, 0-64K ports)

Accessing the Controller



I/O Ports **Memory-Mapped** **Hybrid**

Accessing the Controller

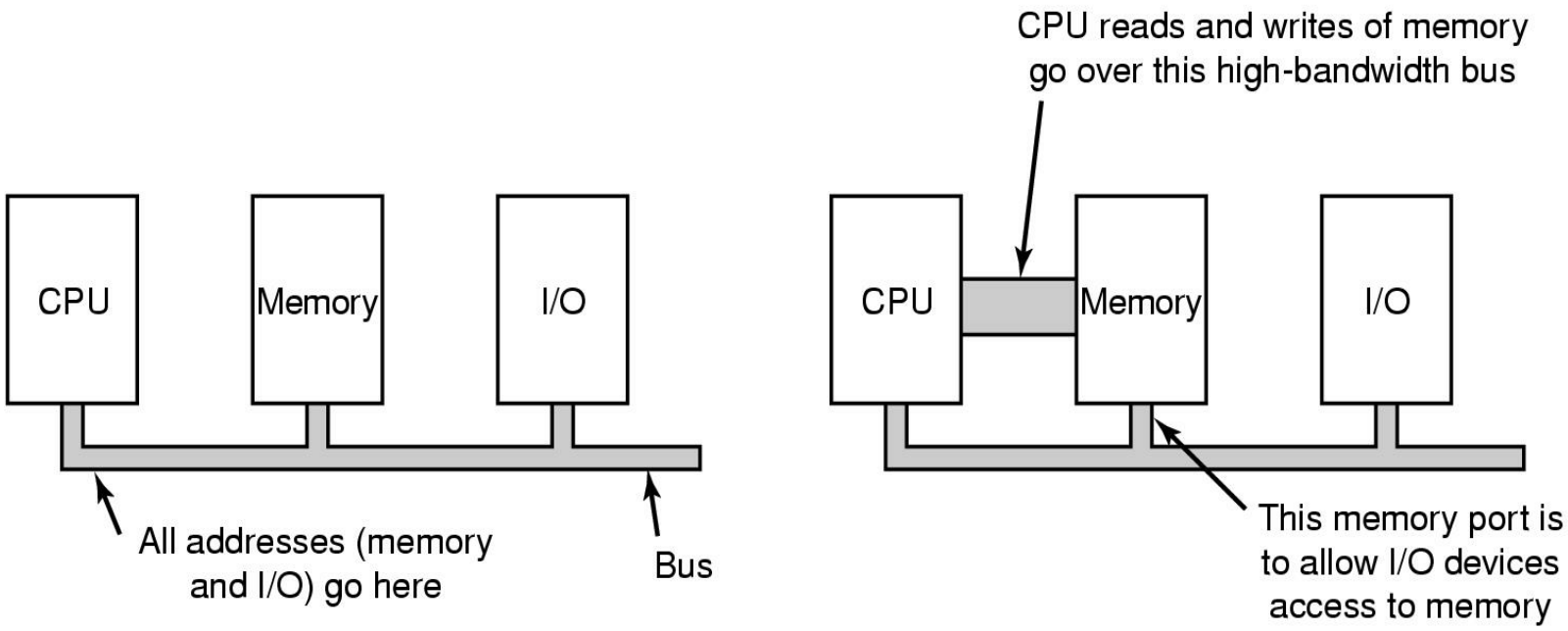
- When a controller register has to be accessed
  - CPU puts address on the bus
  - CPU sets a line that tells if this address is a memory address or an I/O port
  - In case the register/buffer is memory-mapped, the corresponding controller is responsible for checking the address and service the request if the address is in its range

Memory-Mapped I/O

- Advantages
  - Does not require special instructions to access the controllers
  - Protection mechanisms can be achieved by not mapping processes' virtual memory space onto I/O memory

- Disadvantages
  - Caching would prevent correct interaction (hardware must provide a way to disable caching)
  - If the bus connecting the CPU to the main memory is not accessible to the device controllers, the hardware has find a way to let controllers know which addresses have been requested

Memory-Mapped I/O



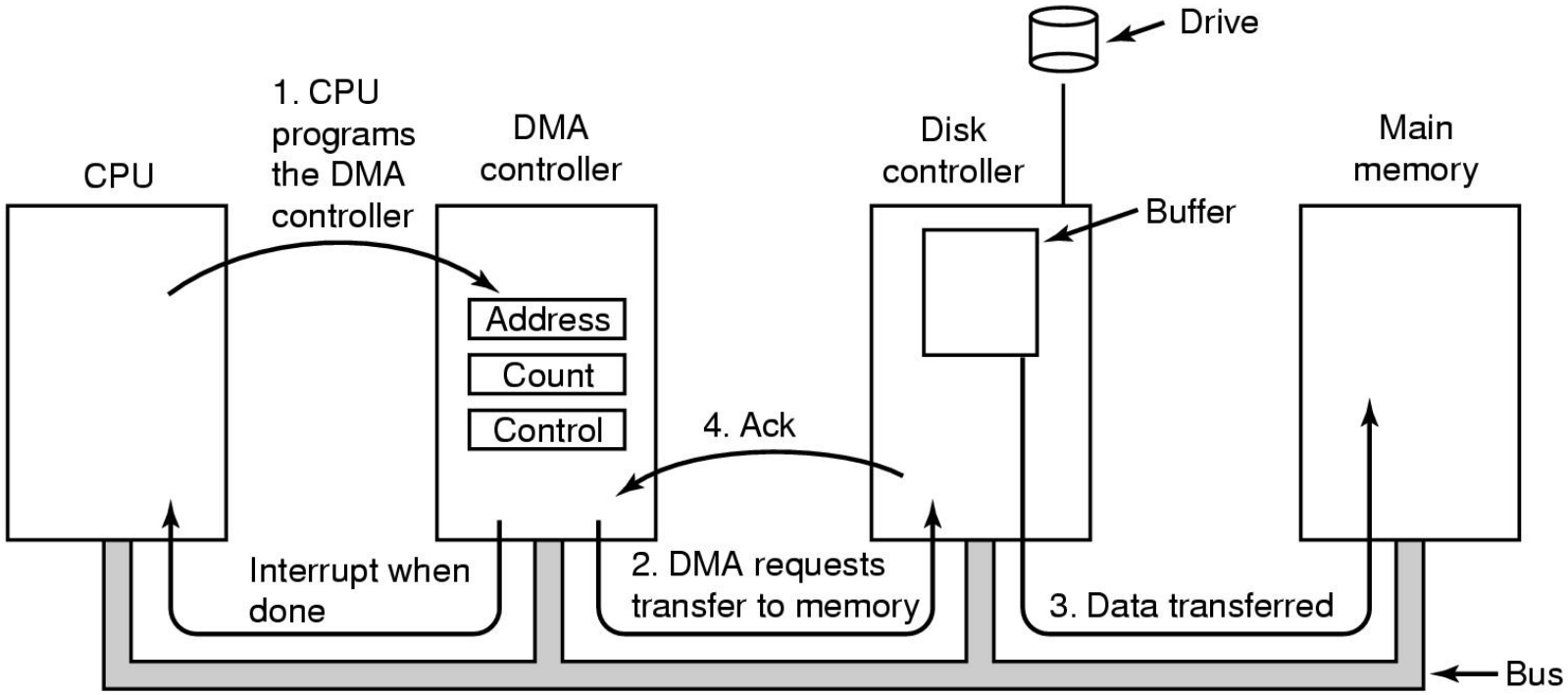
Direct Memory Access (DMA)

- Reading/writing one word at a time may waste CPU time
- A DMA controller supports “automatic” transfer between controllers and main memory
- A DMA controller can be associated with each device or can be one for all the devices
- The DMA controller
  - Has access to the device bus and to the memory
  - Has a memory address register, a count register, and one or more control register (I/O port to use, direction of transfer, etc.)

Reading with Direct Memory Access

- The CPU
  - Loads the correct values in the DMA controller
  - Sends a read operation to the device controller
- The DMA
  - Waits for the operation to complete
  - Sets the destination memory address on the bus
  - Sends a transfer request to the controller
- The controller
  - Transfers the data to memory
  - Sends an ACK signal when the operation is completed
- When the DMA has finished it sends an interrupt to the CPU

Direct Memory Access (DMA)



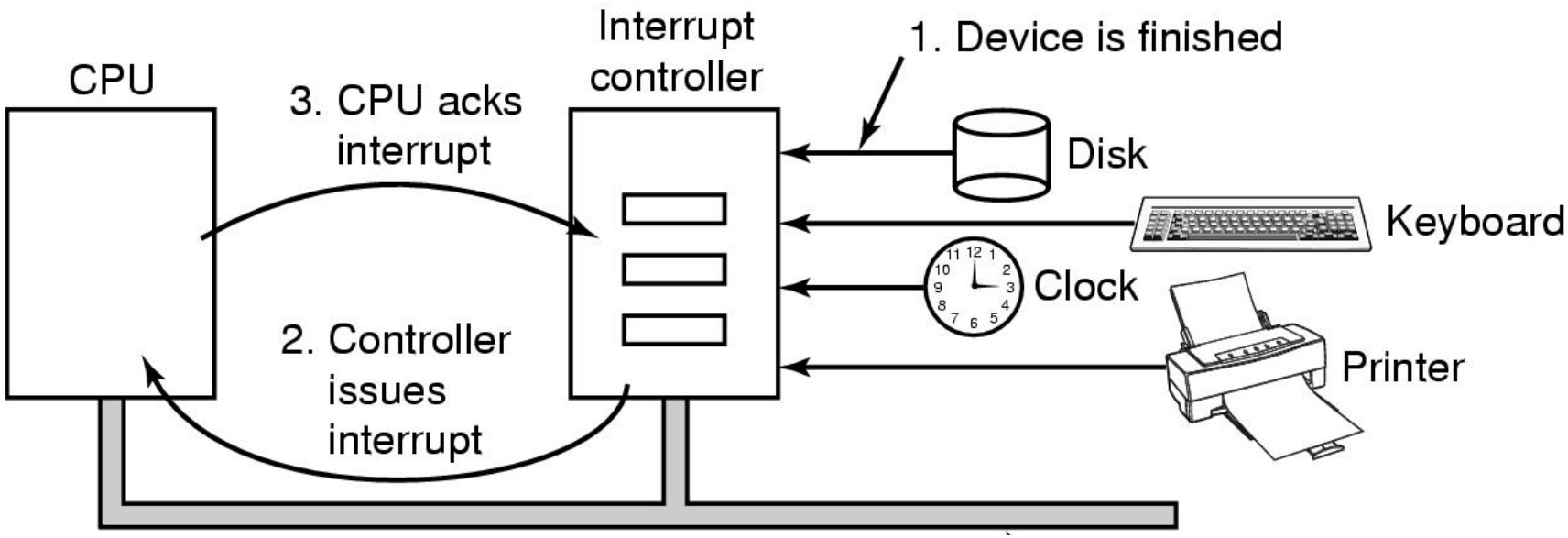
DMA Schema Variations

- Cycle stealing
  - DMA acquires the bus competing with the CPU for each word transfer
- Burst mode
  - DMA tells the controller to acquire the bus and issue a number of transfers
- The DMA may ask the controller to transfer data to a buffer on the DMA controller and then perform the actual transfer to memory
  - supports device-to-device direct transfer

Interrupts

- When a device has completed a task it sends out a signal
- The signal is detected by the interrupt controller
- The interrupt controller puts the device address on the bus and sends a signal to the CPU
- The CPU
  - Stops
  - Saves PC and PSW and uses the address on the bus to look up the *interrupt vector*
- The interrupt vector contains the address of the handling routine which is loaded in the program counter
- After the processing of the interrupt has started an ack is sent to the interrupt controller

Interrupts


CPU
Interrupt controller
1. Device is finished
3. CPU acks interrupt
Disk
Keyboard
Clock
2. Controller issues interrupt
Printer

Saving the CPU State

- The interrupt handler should save the current CPU state
- If registers are used, nested interrupts would overwrite the data and, therefore, the acknowledgment to the interrupt controller must be delayed
- If a stack is used, the information should be stored in a portion of memory that will not generate page faults

Restoring the CPU State

- Restoring is easier said then done when instructions may end up... half-baked (in case of pipelining)

- A *precise* interrupt leaves the machine in a well-defined state
  - The PC is saved in a known place
  - All instructions before the one pointed by the PC have been fully executed
  - No instruction beyond the one pointed by the PC has been executed
  - The execution state of the instruction pointed by the PC is known

- Restoring in case of imprecise interrupts requires a lot of information to be saved

Goals of I/O Software

- Device independence
  - Programs can access any I/O device without specifying device in advance (reading from floppy, hard drive, or CD-ROM should not be different)
- Uniform naming
  - Name of a file or device should not depending on the device
- Error handling
  - Errors should be handled as close to the hardware as possible
- Synchronous vs. asynchronous transfers
  - User program should see blocking operations even though the actual transfer is implemented asynchronously
- Buffering

I/O Software

- System call in user-space
- Data is copied from user space to kernel space
- I/O software can operate in several modes
  - Programmed I/O
    - Polling/Busy waiting for the device
  - Interrupt-Driven I/O
    - Operation is completed by interrupt routine
  - DMA-based I/O
    - Set up controller and let it deal with the transfer

Programmed I/O


```
copy_from_user(buffer, p, count);          /* p is the kernel bufer */
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( );
```

Interrupt-Driven I/O


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

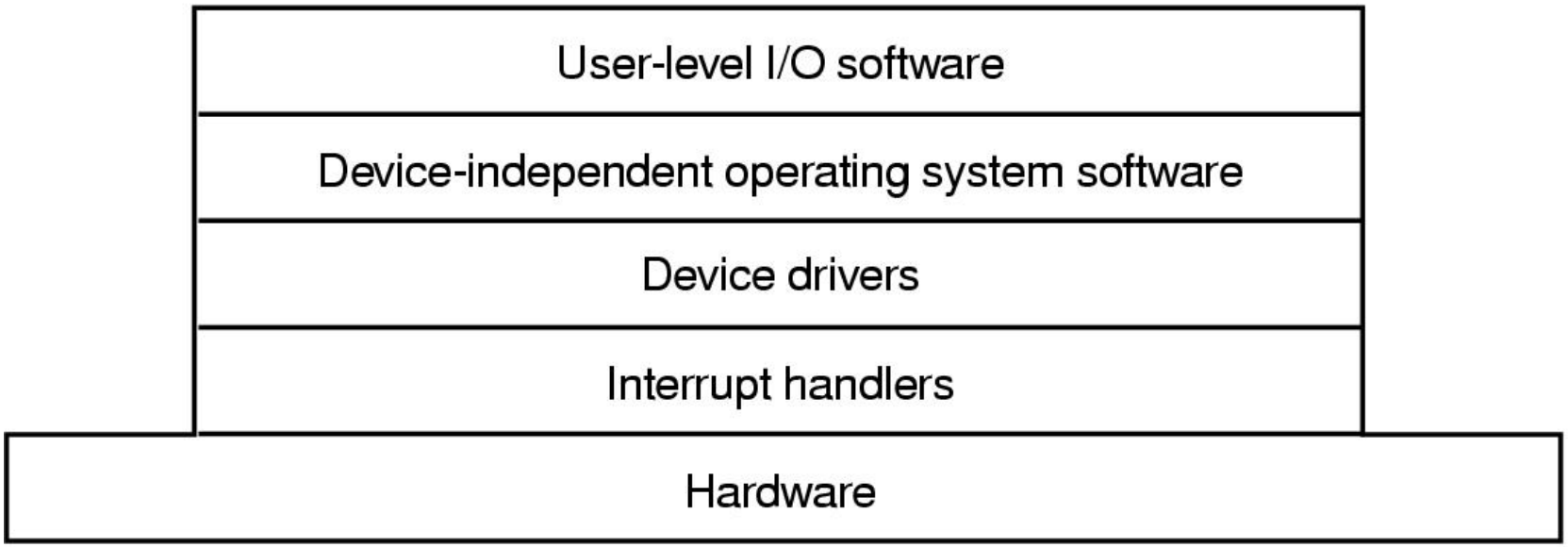
```
if (count == 0) {
    unblock_user( );
} else {
    *printer_data_register = p[i];
    count = count – 1;
     i = i + 1;
}
acknowledge_interrupt( );
return_from_interrupt( );
```

I/O Using DMA


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

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

I/O Software Layers



Interrupt Handlers

- Device driver starts I/O and then blocks (e.g., p->down)
- Interrupt handler does the actual work and then then unblocks driver that started it (e.g., p->up)
- Mechanism works best if device drivers are threads in the kernel

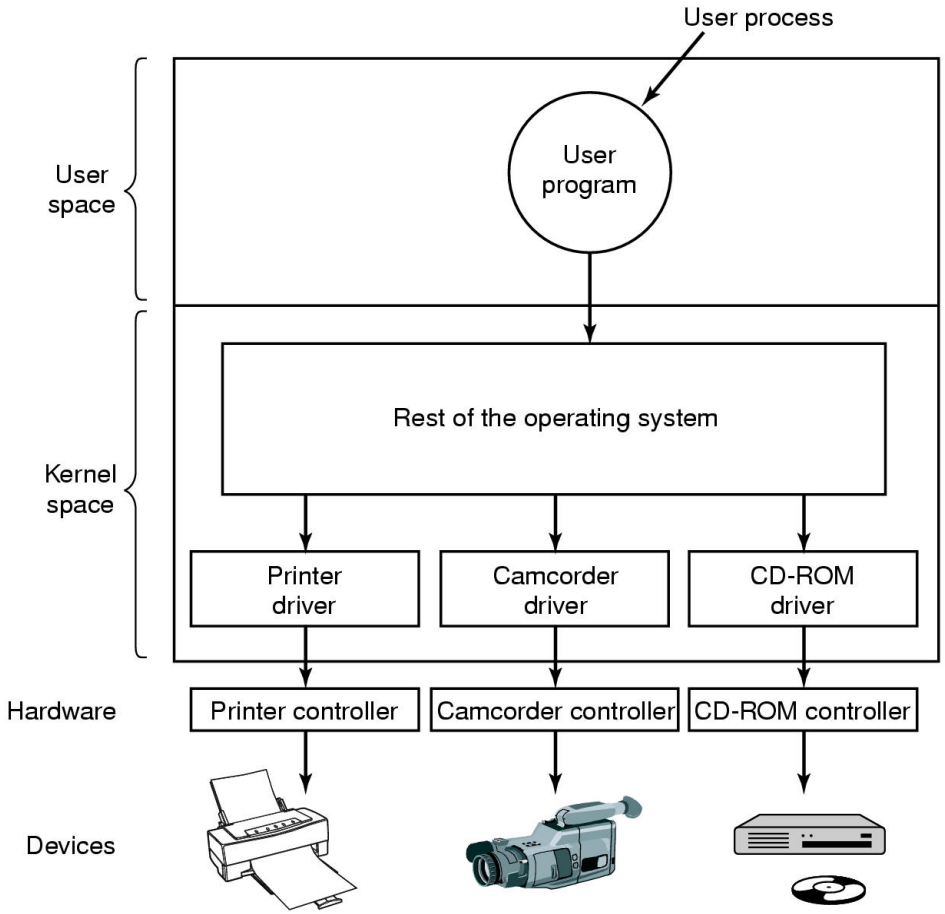
Interrupt Handlers

- Save registers not already saved by interrupt hardware
- Set up context for interrupt service procedure (TLB, MMU)
- Set up stack for interrupt service procedure
- Acknowledge interrupt controller, re-enable interrupts
- Copy registers from where saved to process table
- Run service procedure
- Decide which process to run next
- Set up MMU context for process to run next
- Load new process' registers
- Start running the new process

Device Drivers

- A device driver is a specific module that manages the interaction between the device controller and the OS
- Device drivers are usually provided by the device manufacturer (or by frustrated Linux users!)
- Device are usually part of the kernel
  - compiled and linked in
  - loadable modules
- Usually provide a standard API depending on the type of device
  - Character
  - Block
- Device drivers are usually the source of kernel problems

Device Drivers



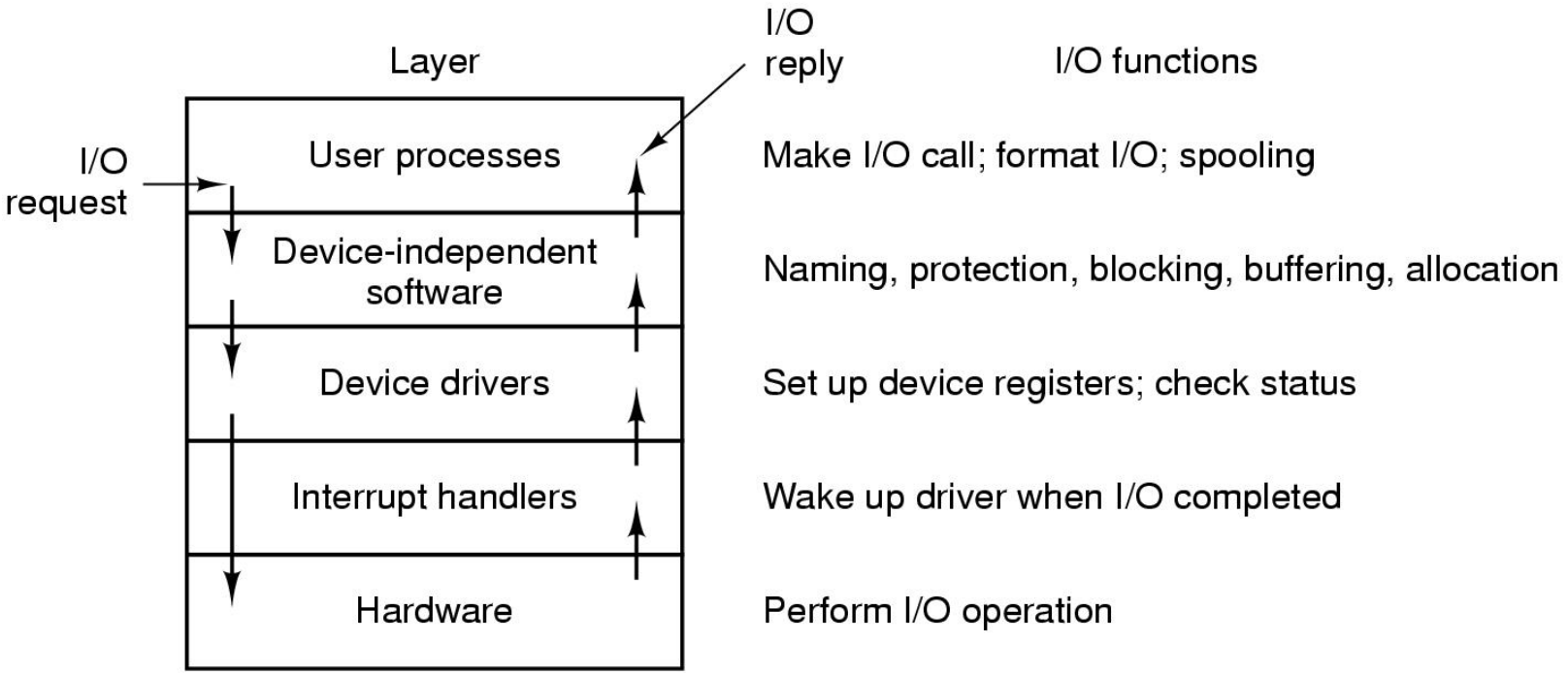
Device Driver's Tasks

- Device initialization
- Accept read-write request from the OS
- Check input parameters
- Start the device if necessary (e.g., start spinning the CD-ROM)
- Check if device is available: if not, wait
- Issue command(s)
- Wait for results
  - Busy wait (awakened by interrupt)
  - Block
- Check for possible errors
- Return results

Device-Independent I/O Software

- Some I/O related functionalities are independent of the particular device and may be carried out outside the device driver
  - Uniform interfacing for device drivers: make all the devices look more or less the same
    - Uniform API
    - Uniform naming
  - Buffering: maintain a copy of the data to read/write in the kernel and transfer to user-space only when needed
  - Error reporting
  - Allocating and releasing dedicate devices
  - Providing a device-independent block size: hide logical/physical differences

User-Space I/O Software



Disk

- Most important and commonly used device
- Used for secondary memory (swap space, file system)
- Different types:
  - Magnetic (floppy, hard disk)
  - Optical (CD-ROM, DVD)

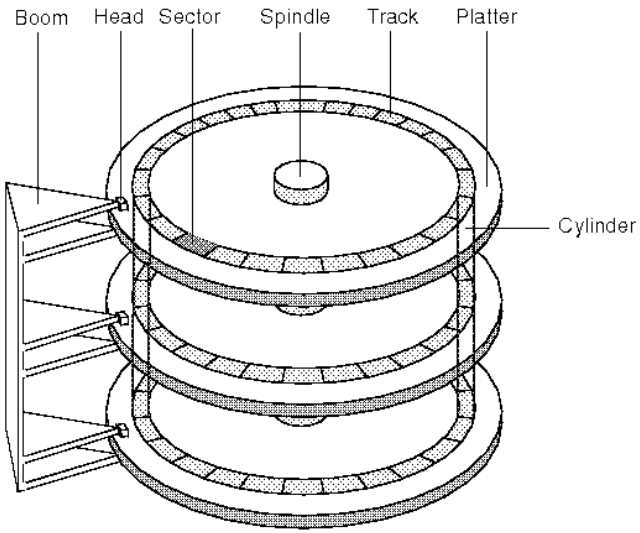
Magnetic Disks

- Disk “geometry” specified in terms of
  - Cylinders composed of tracks (one per head)
  - Tracks composed of sectors
  - Sectors composed of bytes

Parameter	IBM 360-KB floppy disk	WD 18300 hard disk
Number of cylinders	40	10601
Tracks per cylinder	2	12
Sectors per track	9	281 (avg)
Sectors per disk	720	35742000
Bytes per sector	512	512
Disk capacity	360 KB	18.3 GB
Seek time (adjacent cylinders)	6 msec	0.8 msec
Seek time (average case)	77 msec	6.9 msec
Rotation time	200 msec	8.33 msec
Motor stop/start time	250 msec	20 sec
Time to transfer 1 sector	22 msec	17 µsec

Disk Architecture

- Hard disk
  - several platters – disks (heads)
  - each platter has multiple tracks (start with 0)
  - each track has multiple sectors (start with 1)



Disk Architecture

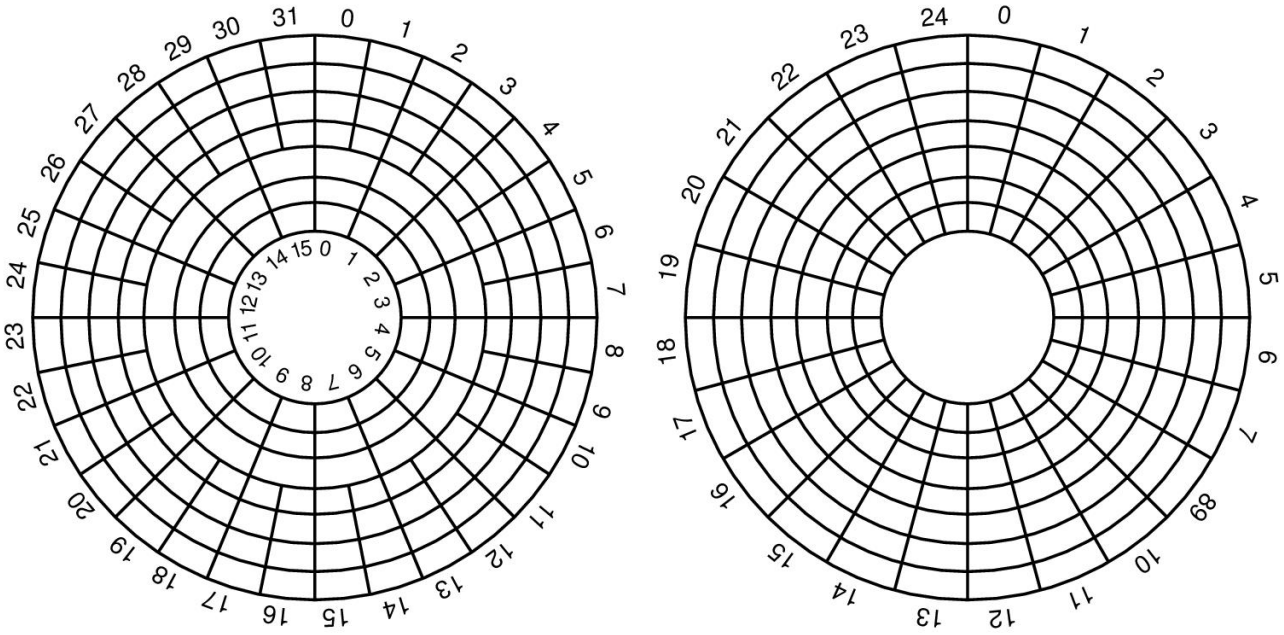
Addressing sectors (blocks)

- CHS (cylinder, head, sector) triple
  - old disks use 10 bits for cylinder, 8 bits for head, 6 for sector
  - limits maximum disk size to ~ 8.4 GB

- Logical block address (LBA)
  - decouples logical and physical location
  - specifies 48 bit logical block numbers
  - allows controller to mask corrupt blocks

Disk Geometry

- Physical geometry could be different from the "logical" geometry
  - Mapping between the two performed by the controller



Disk Architecture

Disk Interfaces

between controller (motherboard) and disk

- ATA (AT Attachment)
  - 28 bit addresses (~128 GB maximum size)
  - 40 pin cables, 16 bit parallel transfer (single-ended signaling)
  - 2 devices (master and slave) can be attached to connection cable
  - ATA-3 introduced security features (passwords)
- Serial ATA (SATA)
  - 8 pin cables
  - higher data transfer (differential signaling)

Disk Architecture

- Hidden protected area (HPA)
  - introduced with ATA-4
  - disk can be set to report to OS less blocks than actually available
  - remaining blocks can be used for data that is not formatted
    utilities and diagnostic tools, but also malicious code or illegal material

- Device configuration overlay (DCO)
  - introduced with ATA-6
  - additional space (blocks) after HPA
  - used by manufacturers to shrink different disks to appear with
    exactly the same size

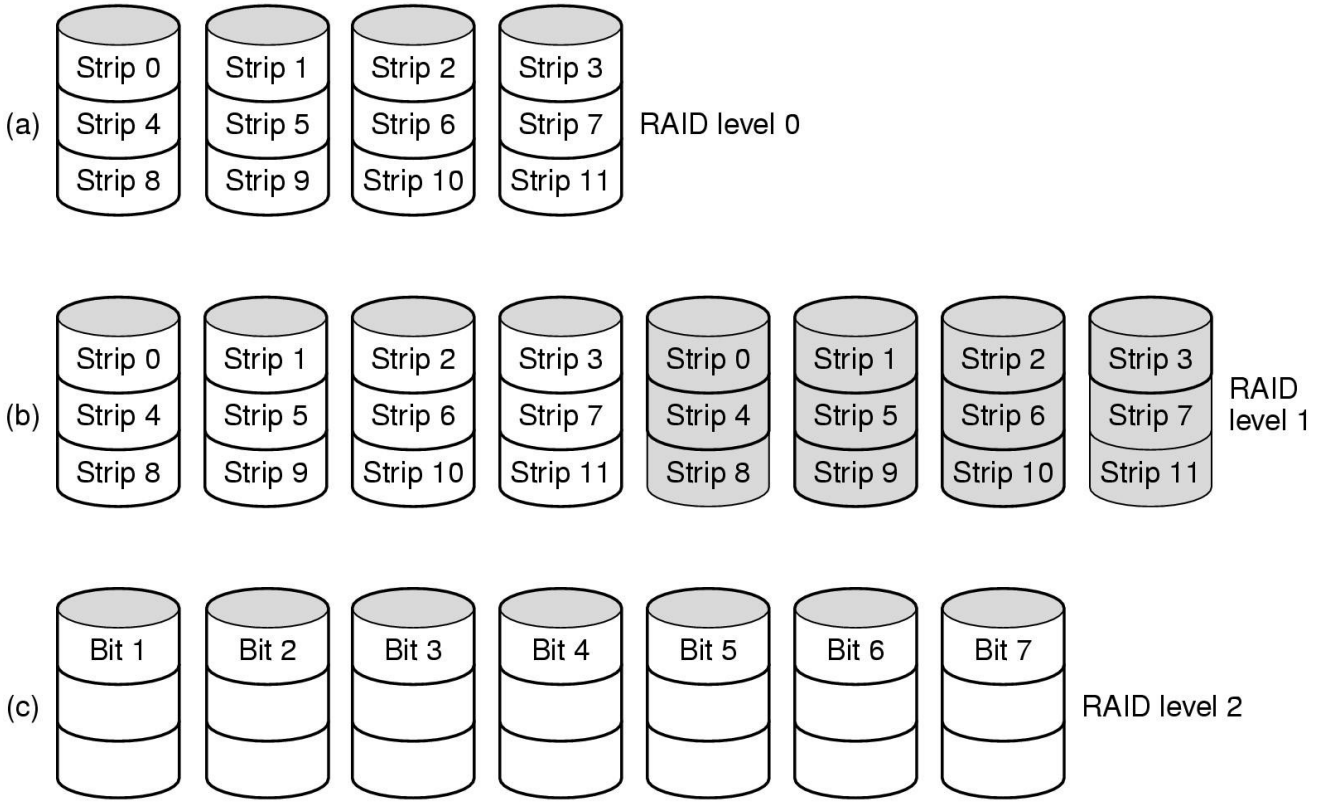
RAID

- Redundant Array of Inexpensive Disks vs.
  Single Large Expensive Disk (SLED)

- A set of disks is managed by a RAID controller
- Different RAID modes (called “levels”)

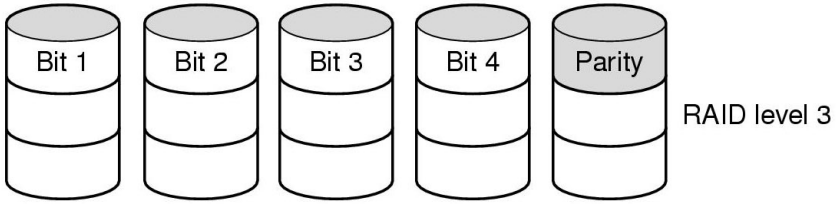
- RAID 0
  - Disks are divided into strips of k sector each
  - Strips are allocated to disks in a round-robin fashion
  - Request for consecutive strips can be carried out in parallel
- RAID 1
  - Striping + redundancy
- RAID 2
  - Striping at the word/byte level + ECC

RAID

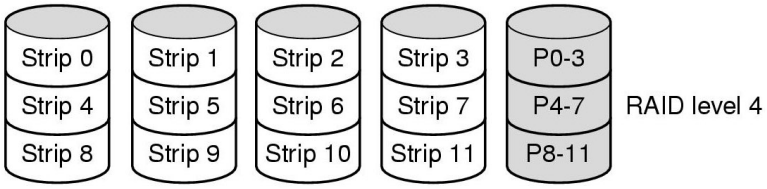

(a)
Strip 0
Strip 4
Strip 8
Strip 1
Strip 5
Strip 9
Strip 2
Strip 6
Strip 10
Strip 3
Strip 7
Strip 11
RAID level 0
(b)
Strip 0
Strip 4
Strip 8
Strip 1
Strip 5
Strip 9
Strip 2
Strip 6
Strip 10
Strip 3
Strip 7
Strip 11
Strip 0
Strip 4
Strip 8
Strip 1
Strip 5
Strip 9
Strip 2
Strip 6
Strip 10
Strip 3
Strip 7
Strip 11
RAID
level 1
(c)
Bit 1
Bit 2
Bit 3
Bit 4
Bit 5
Bit 6
Bit 7
RAID level 2

RAID

- RAID 3
  - Parity word kept on a separate drive

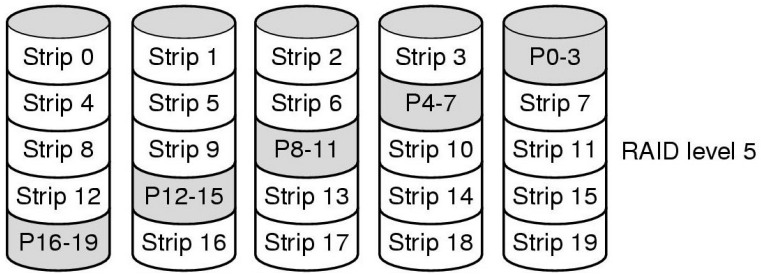


- RAID 4
  - Strip parity on extra drive (XOR of strip contents)



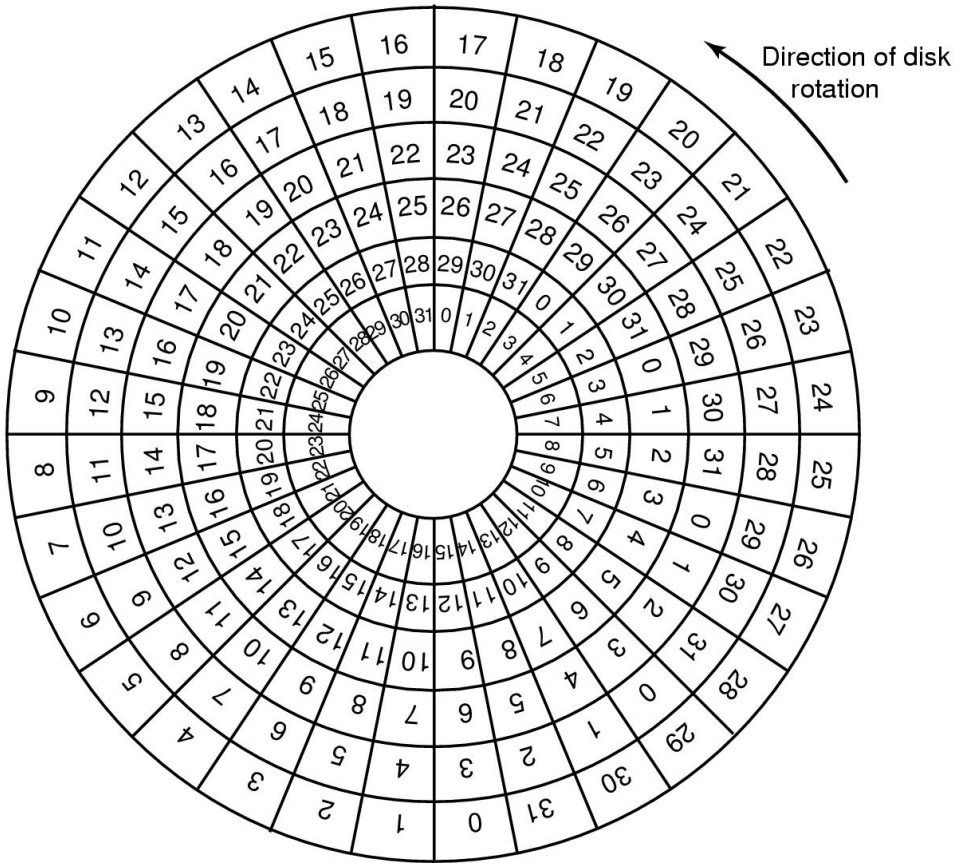
RAID

- RAID 5
  - Parity strips are distributed over the disks



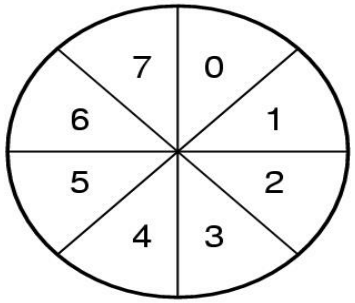
Cylinder Skew

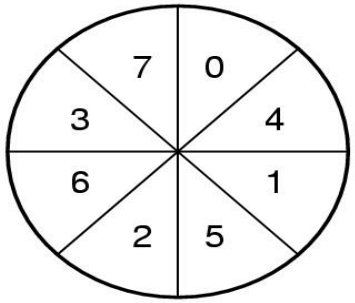
- The initial sector for each track is skewed with respect to the previous one
- This facilitates continuous reads across contiguous tracks by taking into account the rotation of the disk when the arm is moved
- 7,200 rpm with 360 sectors
- Cycle in 60/7,200 = 8,3msec
- Sector rate 8.3msec/360 = 23usec
- Moving from track to track = 900usec
- Skew ~ 40 sectors

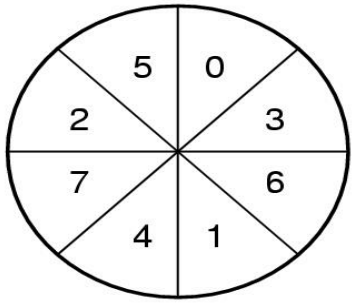


Interleaving

- A disk reads a sector and puts it in the controller's buffer
- While the sector is being transferred to memory the next sector will pass under the disk head
- Solution: Interleaving (single, double, etc)
- Solution: Buffer a whole track at a time





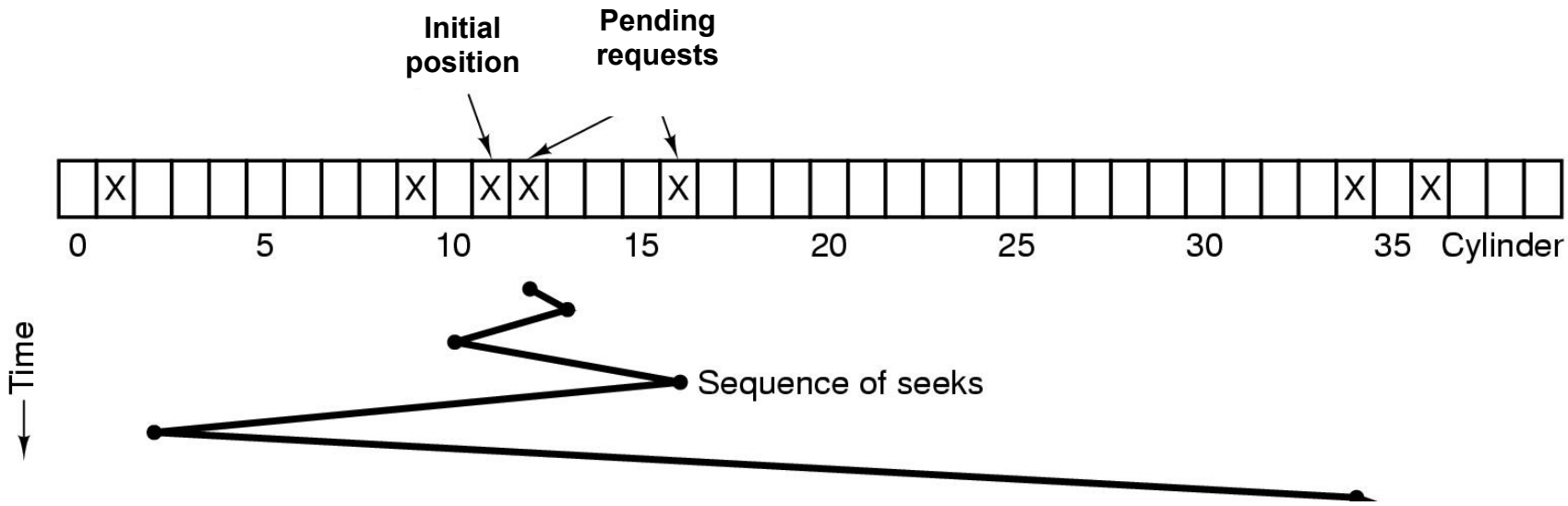


Disk Arm Scheduling Algorithms

- Time required to read or write a disk block determined by 3 factors
  - Seek time
  - Rotational delay
  - Actual transfer time
- Seek time is the most relevant and must be minimized
- Possible scheduling algorithms
  - First-Come First-Served: bad
  - Algorithms with request buffering in the driver
    - Shortest Seek First (SSF)
    - Elevator Algorithm
- Note that these algorithms imply that logical/physical geometry match or at least mapping is known

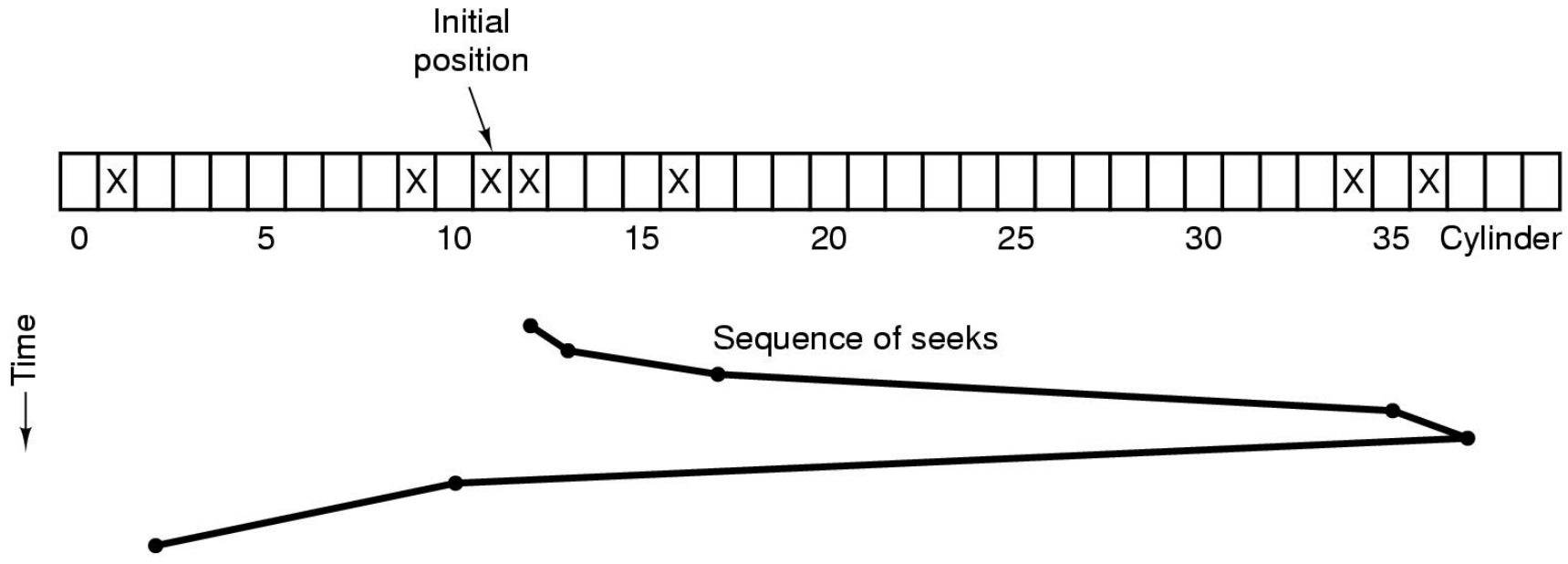
Shortest Seek First Algorithm

- SSF moves the arm towards the closest request
- If request are many the algorithm may be unfair towards request for sectors far from the arm's current position



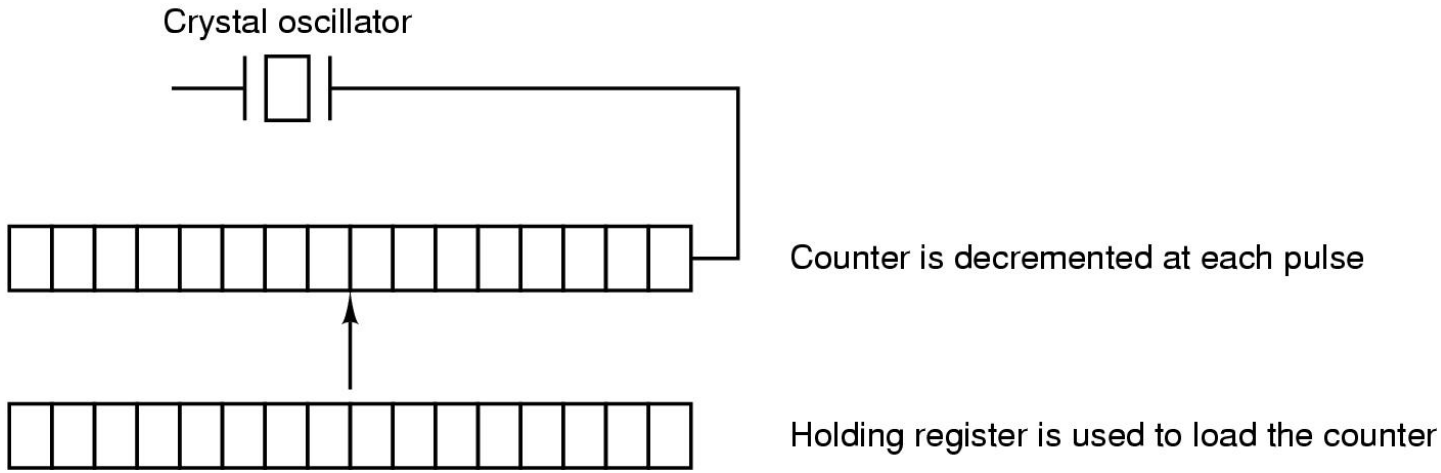
Elevator Algorithm

- The arms moves in one direction until there is no request left, then it changes direction



Clocks

- The clock is a fundamental device
- The counter is initialized with a OS-defined value
- The hardware decrements the counter with a certain frequency (e.g., 500 MHz)
- When the counter reaches 0 an interrupt is sent and the start value is restored



Clock Driver

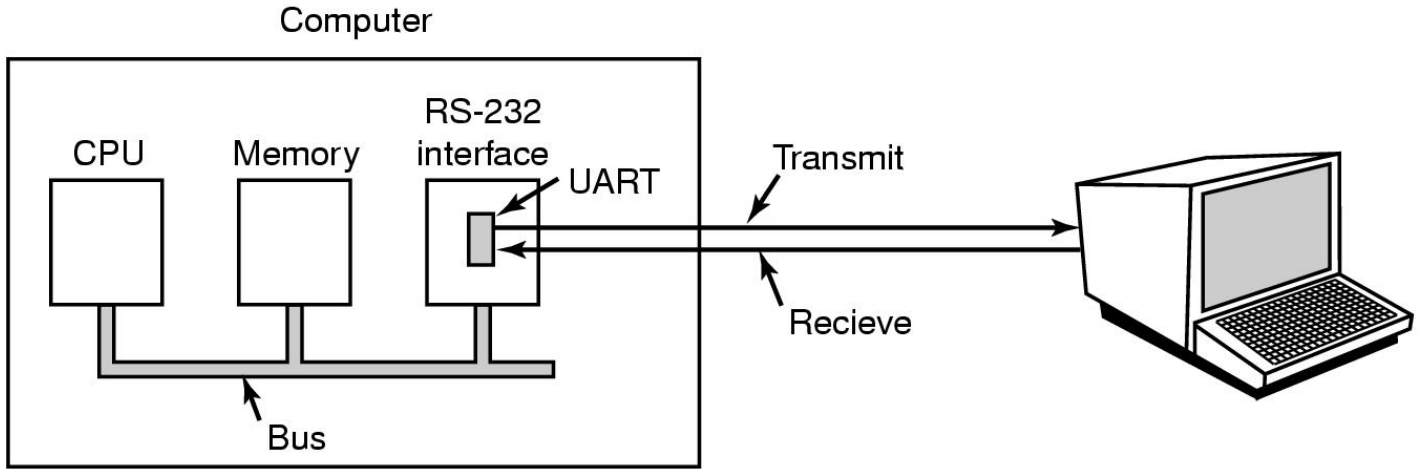
- Maintains the time of day
- Checks processes' CPU quantum usage
  - Calls the scheduler if quantum expired
- Does accounting of CPU usage and profiling of the system
- Handles alarms
  - Alarms are maintained in a list and fired whenever they expire

Character Oriented Terminals

- Simplest form of user-interaction
- A terminal is composed of a keyboard and a screen
- Characters typed from the keyboard are sent to the driver
- Characters sent by the driver are displayed on the screen
- Different modes of operation
  - Raw (non canonical): characters are passed by the driver to the user process as they are typed
  - Cooked, line-oriented (canonical): the drivers performs line-by-line processing before passing the line to the user process
- Drivers maintain buffered input/output and process special characters

RS-232 Terminal Hardware

- An RS-232 terminal communicates with computer 1 bit at a time (serial line)
- Bits are reassembled into characters by the UART (Universal Asynchronous Receiver/Transmitter)
- Windows uses COM1 and COM2 ports, UNIX uses /dev/ttyn
- Computer and terminal are completely independent



Special Input Characters

Character	POSIX name	Comment
CTRL-H	ERASE	Backspace one character
CTRL-U	KILL	Erase entire line being typed
CTRL-V	LNEXT	Interpret next character literally
CTRL-S	STOP	Stop output
CTRL-Q	START	Start output
DEL	INTR	Interrupt process (SIGINT)
CTRL-\	QUIT	Force core dump (SIGQUIT)
CTRL-D	EOF	End of file
CTRL-M	CR	Carriage return (unchangeable)
CTRL-J	NL	Linefeed (unchangeable)

Special Output Characters

- The ANSI escape sequences are accepted by terminal driver on output and converted in specific terminal commands through the termcap mapping

Escape sequence	Meaning
ESC [*n* A	Move up *n* lines
ESC [*n* B	Move down *n* lines
ESC [*n* C	Move right *n* spaces
ESC [*n* D	Move left *n* spaces
ESC [*m* ; *n* H	Move cursor to (*m*,*n*)
ESC [*s* J	Clear screen from cursor (0 to end, 1 from start, 2 all)
ESC [*s* K	Clear line from cursor (0 to end, 1 from start, 2 all)
ESC [*n* L	Insert *n* lines at cursor
ESC [*n* M	Delete *n* lines at cursor
ESC [*n* P	Delete *n* chars at cursor
ESC [*n* @	Insert *n* chars at cursor
ESC [*n* m	Enable rendition *n* (0=normal, 4=bold, 5=blinking, 7=reverse)
ESC M	Scroll the screen backward if the cursor is on the top line