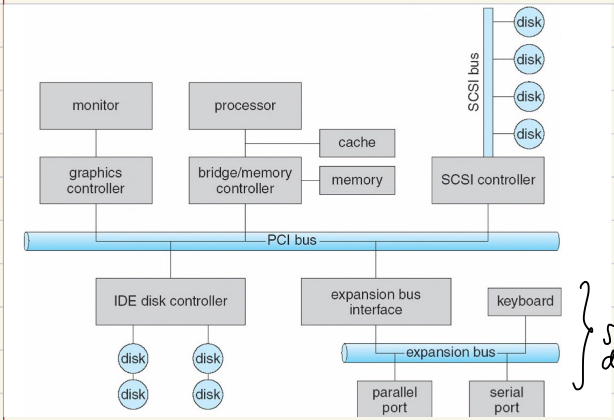
I/O Hardware



} slower devices

- each device has it's own controller (implemented as host adapter)
- CPU talks to controller
- PCI Bus : peripheral component interconnect
  ↓
  connects processor and memory subsystem to faster devices

- CPU issues instr. to device controller which does the work, after I/O, an interrupt is generated, it is handled by the CPU

- because there's so many I/O devices, OS needs to take care of all of them.
  ↓
- the devices can vary in many dimensions

aspect	variation	example
data-transfer mode	character (bytes) —from→ block —from→	terminal disk
access method	sequential ←uses— random ←uses—	modem CD-ROM
transfer schedule	synchronous asynchronous	tape (block device) keyboard (character device)
sharing (accessed one device at a time or not.)	dedicated ←is— sharable ←is—	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only ←is— write only ←is— read–write ←is—	CD-ROM graphics controller disk

- human readible / machine readible

I/O System Design Objectives

- Efficiency

→ ∵ I/O is much slower than memory access, I/O operation form the bottleneck.

and processor
→ CPU performance has bettered exponentially while disk has improved linearly over the years, so I/O performance is critical

→ possible solution: multiprogramming

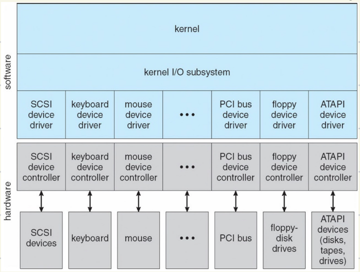
- General

→ want to be able to handle all devices in a uniform manner.

→ possible solution: layered architecture, modular approach to the design

details of device I/O are hidden in the lower level so that user processes and upper levels only see devices in terms of their functionality.

Kernel I/O Structure



- encapsulation

called using system calls

generic service

device driver: generates generic inst. to corr. inst. set for I/O device

device controller

- adding new device

⇒ add device driver

Kernel I/O subsystem

- provides generic services

- I/O scheduling : reorder I/O request to improve performance

↓

in multiprog systems, multiple process can issue I/O request for the same device

↳ I/O request queue

- Buffering : ① because diff devices have different speeds

↓

at any time, there is only 1 copy of the message

copy file from faster to slow device, the buffer will slow down the transfer from the faster device so that the slow one can cope

② devices have diff transfer sizes, so transfer files are segmented and reassembled later by the buffer.

- Caching : improves efficiency

↓

eg. open file table. or TLB

make a copy of the data in a faster device, so getting data becomes quicker.

↓

improves I/O of files that are written and reread rapidly

holds output or input for a device that can't have interleaved data streams.

- Spooling : special kind of buffer; used in devices that can only serve one request at a time

↓

temporarily stores data

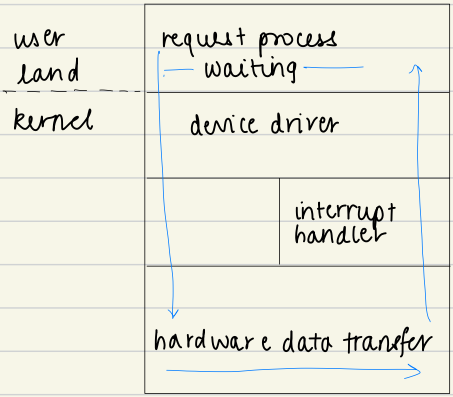
Performance Consideration

application of I/O interface : 2 options

handled by OS

system call

BLOCKING I/O

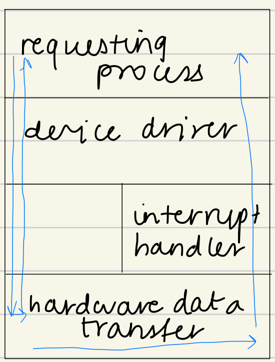


① A process makes an I/O request and the request is sent to the I/O device where it is processed

② Process is put in waiting state while I/O request is being processed

③ Process is put back to ready queue when the I/O request is completed.

ASYNCHRONOUS I/O



① Process makes an I/O request and the request is sent to the I/O device where it is processed

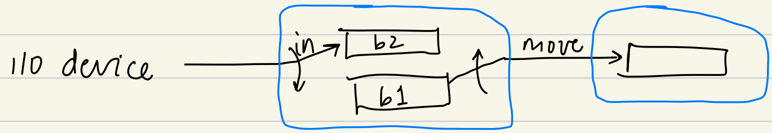
② After the system call, it returns as a the request is queued at the I/O device

③ So process is either in running or ready state.

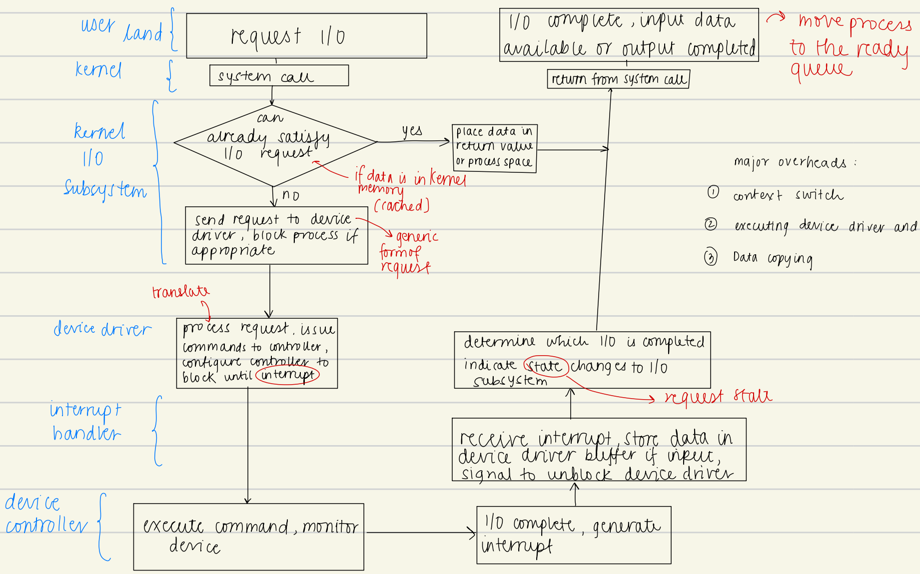
④ A flag or signal is set up to inform process when the I/O is complete when to use?

- If source file is a single source, process is waiting for one specific even, then blocking I/O is used.
- If the source is multiple (I/O request may come from multiple sources), use Asynchronous I/O
eg. server. window manager

ASYNCHRONOUS I/O WITH BUFFER



blocking I/O operations: + calculating overhead.



major overheads:

1. context switch
2. executing device driver and controller code
3. Data copying

Disk

Structure

cylinder = combination of tracks in the same position from different sectors.

- platter ↓ 2 surfaces → each surface has a head

covered in magnetic material

↓ divided into tracks

seek time : moving to track
10 ms

1K sectors ↓ divided into sectors (512 bytes) ← smallest storage on disk

rotational latency : moving head to sector
4 ms

block = sequence of sectors (1 or more)

Time to read/write on N sectors

- positioning time + data transfer time (N)

↓ seek + rot. latency

(N / no of sectors per track) x full rotation time

RANDOM REQUEST

seek + rotational + transfer time : $10 + 4 + (1/1024 \times 8) = 14$ ms

N bytes : $N/512 \times 14$ ms (N/512 = no of sectors) its so fast

we love it

SEQUENTIAL REQUEST : all required sectors are next to each other

locality is very important

- $\underbrace{\text{seek + rotational latency}}_{10+4}$ + $\underbrace{\text{transfer time}}_{(8/1024)\times \text{full rotational time (8)}}$

4KB : 8 sectors : 14 ms
128KB : 256 sectors : 16 ms

- positioning time ⟹ n times depending on n tracks

Scheduling

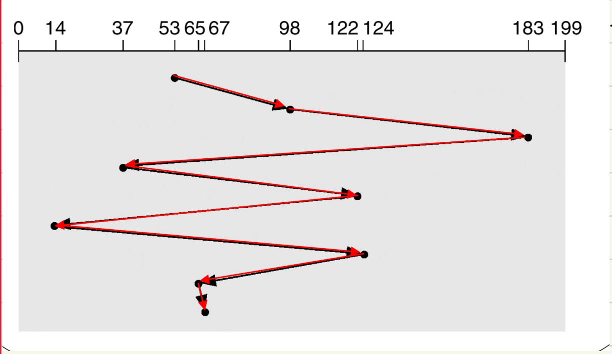
- for efficient hardware usage, disk drives should have a fast access time and high disk bandwidth
- minimise seek time → dependent on seek distance
- disk bandwidth : $\frac{\text{total number of bytes transferred}}{\text{time b/w 1st request \& completion of last transfer}}$
- minimise seek time by reducing distance travelled by disk head when servicing multiple random requests by managing the order in which these request are serviced

- to optimise such scheduling we have algorithms :

request queue : 98, 183, 37, 122, 14, 124, 65, 67

assume head is initially at cylinder 53

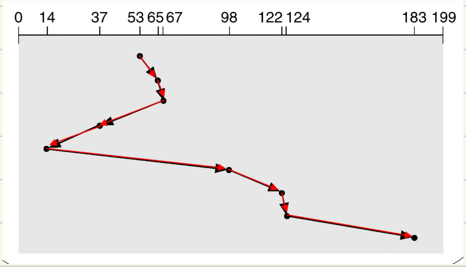
FCFS



total head movement = 640

SSTF : Shortest seek time first

- a form of SJF
- may have starvation under heavy load ∵ distant requests don't get serviced.

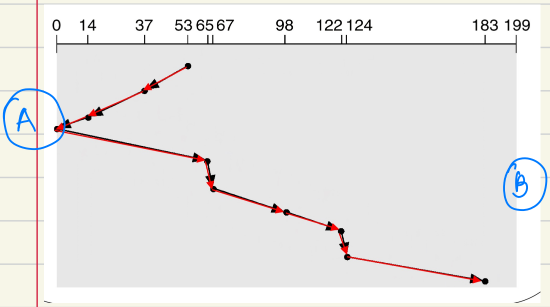


total head movement : 236

SCAN : or elevator algorithm

2 directional scanning

- arm starts at 1 end of the disk, moves in that direction, services all requests along the way, reaches the end and reverses.
- innermost cylinder : 0



total head movement : 236

when arm is at A, most requests are at B side.

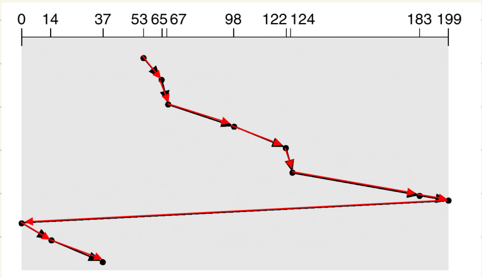
time 2t

CIRCULAR SCAN

1 directional scanning

head goes from one end to the other w/ servicing requests.
low to high → service
return to low → no service
continue

reduces maximum delay experienced by new requests



time t + s

total head movement : 382
(183 real)

CIRCULAR LOOK

- sometimes, max seek time is too much
- so instead of returning to the beginning, you go to the lowest cylinder # request

- algorithm can depend on file-allocation method.

↓

contiguous : close together ← less head movement
indexed / linked : wide, random ← more head

1. FCFS has no overhead
2. the rest do.

1. need a queue to test.

low load ⇒ close to FCFS

heavy load, multiprog degree is high, → performance improvement is greater than low load.

Disk Management

Low-Level / Physical Formatting

- dividing a disk into sectors that disk controller can read and write
- done by factor
- low-level formatting fills the device with a special data structure for each storage location.
- the ds of a sector has:
  1. header
  2. data area
  3. trailer

  controller uses

- to use a disk to hold files, OS needs to record its own data structures on it.
  ↓ does so by (eg. inode)

1. partition device into groups of cylinders
   - treat each partition as if it were its own device
   - partitions can be used to store different things
   - partition info is written in a fixed format at a fixed location on the storage device
   - its called "raw" if it does not hold a file system