Device Management

Learning objectives:

- Organization of I/O functions
- Key issues in design of OS support for I/O
- I/O buffering
- Understanding performance issues involved in mg meagnetic disks.
- Understanding RAID.

I/O devices

- Human readable
  - printers, terminals, keyboards, etc.
- Machine readable
  - disk drives, USB, sensors, controllers.
- Communication
  - Modems, routers, switches.

I/O devices differ in many categories.

- Data rate
- Applications
- Complexity of control
- Unit of transfer
- Data representation
- Error conditions.

There could be thousands of different types of devices that you can connect to a computer system. This makes it very hard to create an abstract way of handling each device.

Organization of I/O function

- Programmed I/O
- Interrupt driven I/O
- Direct memory Access (DMA)

Programmed I/O

Processor issues I/O command on behalf of the process to an I/O module; that process then busy waits for the operation to be completed before processing.

Processor waits and polls the I/O status register.

Interrupt driven I/O

This is an improvement for the programmed I/O. Processor issues I/O command on behalf of the process. But it no longer polls for the I/O status register. Instead processor will keep executing the process that issued the I/O command if it was non-blocking, otherwise, dispatcher will select some other process to run.

- Once the I/O device is ready it will send an interrupt to the processor, which in turn ~~swap~~ dispatch the original process that issued the I/O command.

DMA.

- This is a separate module that controls exchange of data between main memory and I/O module.
- Processor sends a request to the DMA module, then DMA module acts independently and transfer the data requested to a part of main memory.
- Once everything is done, interrupt is sent to the processor

Evolution of I/O functions.

1. Processor directly controls peripheral devices.
2. A controller I/O module is added
3. Same configuration as (2) but interrupt is added.
4. I/O module is given direct control of memory via DMA.
5. The I/O module is enhanced to become a separate processor, with specialized instruction set tailored for I/O
6. I/O module has local memory of its own, in fact it acts as a separate computer.

Design objectives of I/O functions

1. Efficiency
2. Generality.
   - ↳ Desirable to handle all input devices in a uniform manner.

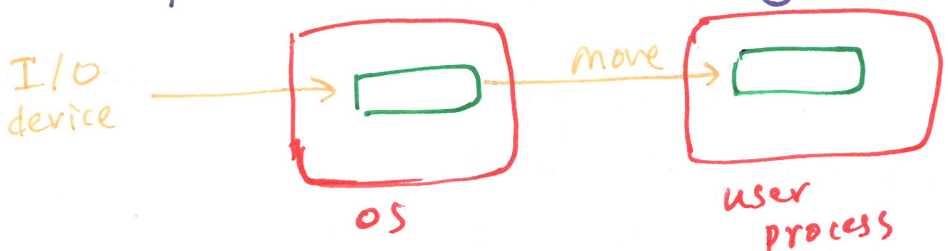
Buffering

It's convenient to have a buffer between Processor and I/O

No - Buffer
- OS directly accesses I/O device when needed.
- Inefficient.

Single buffer

- When user process issues I/O request, OS creates a buffer area in the system portion of the memory



- A real example of producer/consumer problem.

Advantages

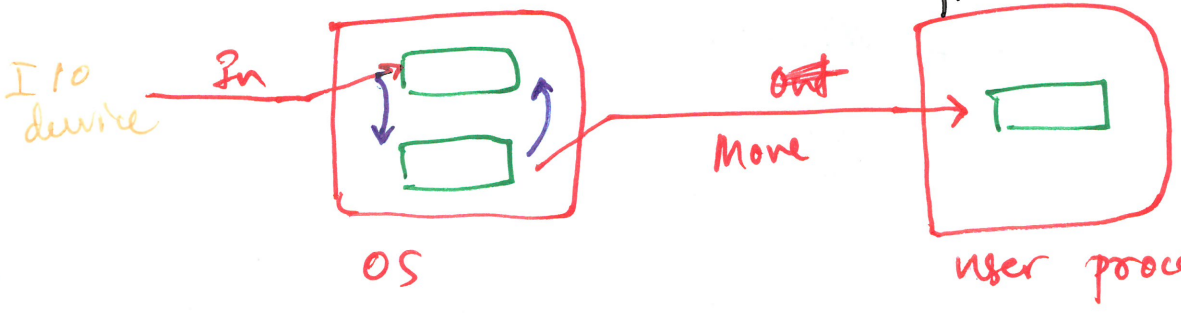
- The user process can process one block of data while the other block is being read in
- The OS is able to swap the process out because the input data that is being read in is stored in the system portion of the memory.

Disadvantages

- complicated logic
- swap logic is also affected.

Double Buffer.

- Two system buffers are used.
- A process now transfers data to or from one buffer while OS empties or fills the other buffer.
- Also known as "Buffer swapping"



- Basically I/O device or OS fills or empties one buffer while the process empties or fills the other buffer.

- when one buffer is empty (or full) it ~~waits~~ swaps the buffers when its ready.

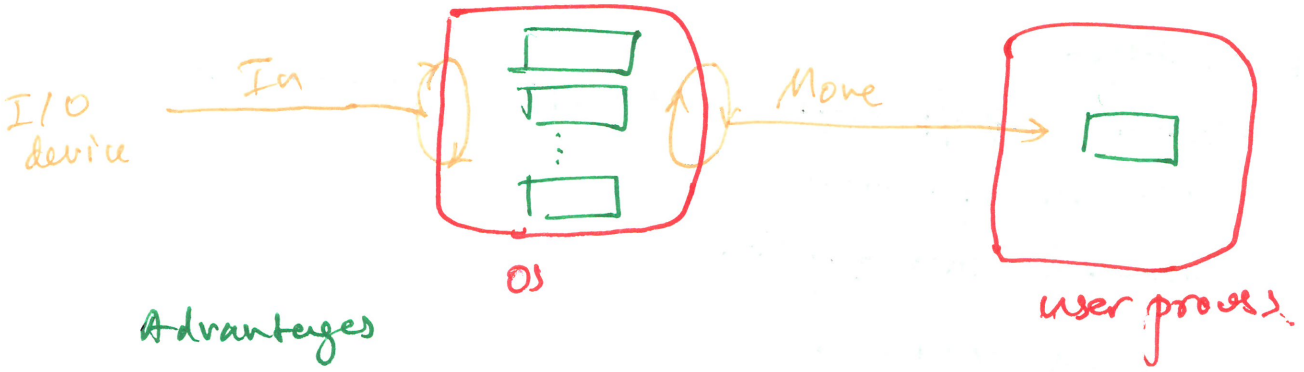
Advantages
- Allows CPU/process and I/O devices to work in parallel.
- Increased throughput.
- Reduced latency.

Disadvantages
- Memory overhead; Requires twice the amount of memory to be allocated to hold two buffers.
- Synchronization: careful coordination between the CPU & I/O device is necessary to ensure that buffers are used efficiently.

Example of double buffering:
1. Disk reads data into the 1st buffer.
2. When the 1st buffer is full the disk starts to read data into the 2nd buffer.
3. While disk is filling the 2nd buffer, the CPU process the data from the 1st buffer.
4. The cycle continues by swapping the buffers.

Circular - Buffer

- More than 2 buffers are used.
- Each individual buffer is one unit in circular buffer.
- smooths out flow of data between I/O device & process.



Advantages

- Efficient use of memory
  A single fixed size buffer (where elements/items are buffers) is used. Wrapping behaviour ensures that memory is used efficiently.
- continuous data transfer
  continuous buffer allows seamless and continuous data streaming. (good for audio processing, real-time sensor data collection, video streaming, etc)
- Reduced latency.

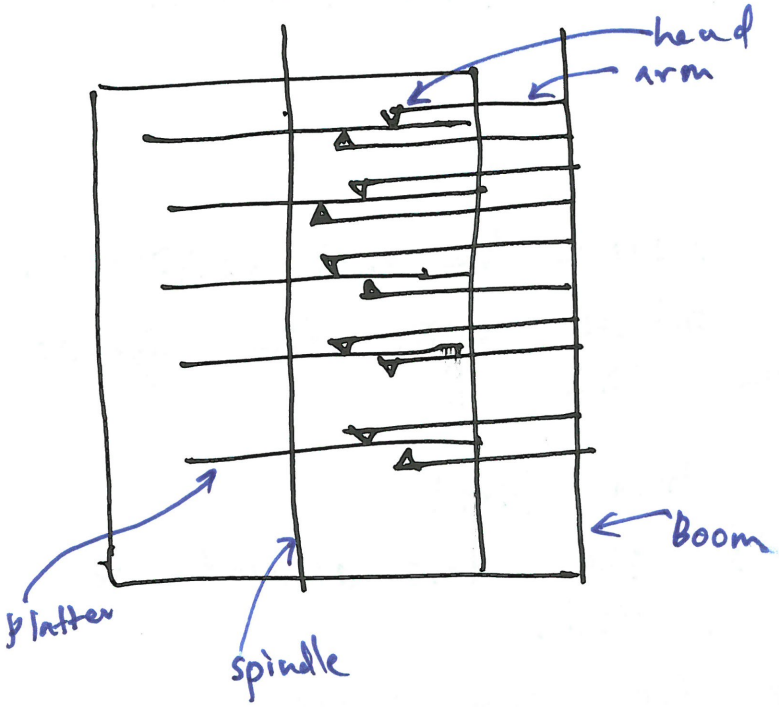
Disadvantages and challenges

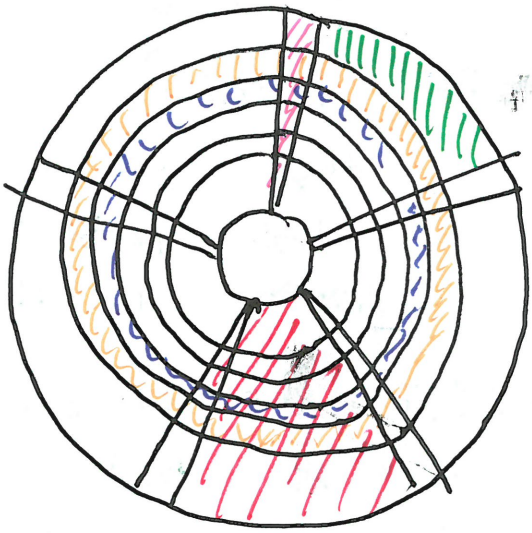
- Buffer overflow
  If write pointer catches up to read pointer.
- Buffer underflow
  If read pointer catches up to write pointer
- Synchronization issues.

Example of circular buffering ~~using~~: Network Packet Handling

1. Network interface writes incoming packets into the circular buffer at the position indicated by the write pointer.
2. As soon as packets are written into buffer, the CPU can read and process the packets from the position indicated by the read pointer
3. Write pointer advances as more packets arrive and read pointer advances as packets are processed.

Disk scheduling





- sector
- track
- Interstracle gap

- geometric sector
- Intersector gap

Seek time - Time taken to move the head to the track

rotational delay - Time taken to rotate to correct sector.

Access time - seek time + rotational delay.

Scheduling policies are needed to improve access time.

- FIFO (first-in-first-out)
- LIFO (Last-in-first-out)

  Serves the FIFO/LIFO requests from I/O request queue.

- PRI (Priority)

  does not try to improve the disk utilization rather let the process with the highest priority access the disk

- SSTF (shortest service time first)
  - request that needs the least amount of head movement is served.
  - starvation is possible for some processes.

- SCAN
  - Arm moves in one direction at a time.
  - serves all the requests in that direction
  - once the end is reached direction changes to opposite direction.

C-SCAN

- Just like scan, but direction is circular.

N-step-SCAN

- divides the queue into subques of size n.
- Then runs SCAN on subques.
- service is guaranteed

FSCAN

- two ques are used
- One will be served the other is used to queue new requests
- once one queue empties ques are swapped.

RAID

- Redundant Array of Independant Disks.
- framework for handling multiple disks parallely.
- 2 - configurations.

~~RAID - 0~~

- ~~Not 10~~

RAID

set of physical drives arranged as an array of disk viewed by the OS as one logical drive

Redundant disk capacity is used to store parity information, which guarantees data recoverability in case of a disk failure.

Data are distributed across the physical drives of an array in a scheme known as striping

RAID - 0

- Not considered as part of RAID
- disks are divided into strips
- Each strip of data is stored in one of the disks
- N disks are needed
- No redundant data is kept
- If one disk fails data is not recoverable.

RAID - 1

- Similar to RAID-0, but copy of N drives are kept
- 2N # of drives are needed.
- If one drive fails, new drive can be used and lost data can be reconstructed
- No write penalty as parity bits are not kept
- cost is very high.

RAID- 2

- uses parallel access
- Data striping is used
- Typical strip size is 1-byte
- Each bit of the strip is kept in a separate drive.
- parity bits based on hamming code is calculated & kept in the parity disks.
- N + m disks needed.
- overkill for most cases.

RAID - 3

- similar to RAID-2, but each disk keeps strip instead of a bit of strip.
- One parity disk is used.
- for each i^{th} bit in a strip of a stripe is used to calculate the parity & stored in the parity disk.
- single parity disk becomes a bottleneck.
- Still overkill for most cases.

RAID - 4

- uses independent access technique.

- block level stripping is used.
- Dedicated parity disk
- Good for applications that prefer block level transfer.
- Better performance for random access compared to RAID-3 because block level access is used.
- Dedicated parity disk becomes a bottleneck.
- Good for read heavy environments.
- N+1

RAID-5

- similar to RAID-4 but distributes parity bits across all disks.
- Less bottleneck since no dedicated parity disk is used.
- Good balance between read & write performance.
- write penalty due to parity calculations.
- N+1

RAID-6

- similar to RAID-5, Also distributes parity data
- Uses two parity calculations (Additional redundancy)
- N+2 drives.
- can tolerate 2 disk failures at the same time.
- Higher write performance penalty
- lower storage efficiency
- No dedicated parity bottleneck
- Ideal for large storage arrays where likelihood of multiple disk failures is higher (ex: data centers)