Lecture 27
Storage + I/O

Disk Fault Tolerance with RAID

- Redundant Array of Inexpensive Disks
  - Several smaller disks play a role of one big disk
- Can improve performance
  - Data spread among multiple disks
  - Accesses to different disks go in parallel
- Can improve reliability
  - Data can be kept with some redundancy

RAID 0

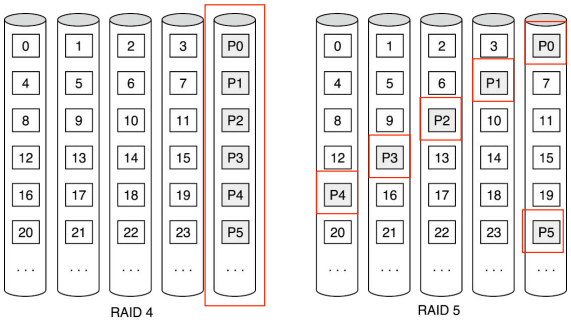
- Striping used to improve performance
  - Data stored on disks in array so that consecutive "stripes" of data are stored on different disks
  - Makes disks share the load, improving
    - Throughput: all disks can work in parallel
    - Latency: less queuing delay – a queue for each disk
- No Redundancy
  - Reliability actually lower than with single disk (if *any* disk in array fails, we have a problem)

RAID 1

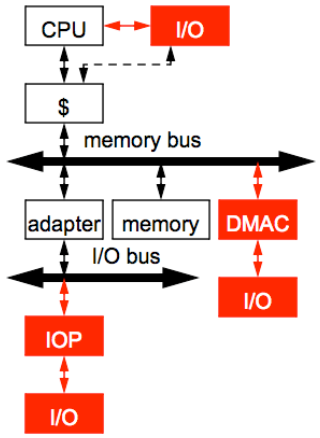
- Disk mirroring
  - Disks paired up, keep identical data
  - A write must update copies on both disks
  - A read can read any of the two copies
- Improved performance and reliability
  - Can do more reads per unit time
  - If one disk fails, its mirror still has the data
- If we have more than 2 disks (e.g. 8 disks)
  - "Striped mirrors" (RAID 1+0)
    - Pair disks for mirroring, striping across the 4 pairs
  - "Mirrored stripes" (RAID 0+1)
    - Do striping using 4 disks, then mirror that using the other 4

RAID 5

- Distributed block-interleaved parity
  - Like RAID 4, but parity blocks distributed to all disks
  - Read accesses only the data disk where the data is
  - A write must update the data block and its parity block
    - But now all disks share the parity update load



I/O System Architecture

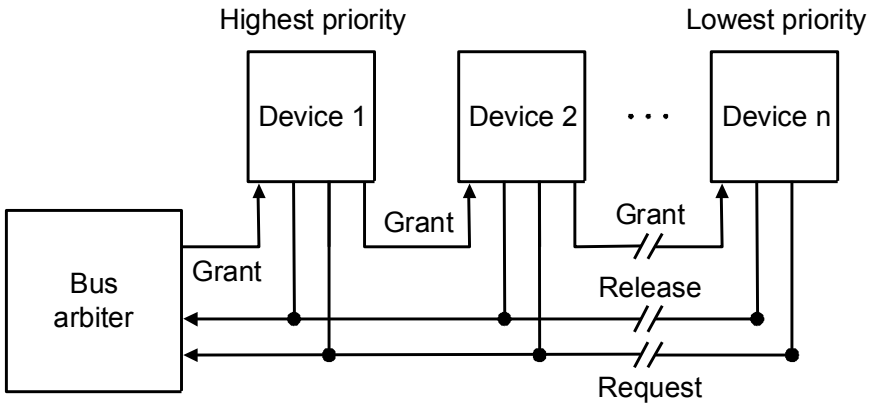


- buses
  - memory bus
  - I/O bus
- I/O processing
  - program controlled
  - DMA
  - I/O processors (IOPs)

Bus Issues (Memory & I/O Buses)

- *clocking*: is bus clocked?
  - synchronous: clocked, short bus ⇒ fast
  - asynchronous: no clock, use "handshaking" instead ⇒ slow
- *switching*: when is control of bus acquired and released?
  - atomic: bus held until request complete ⇒ slow
  - split-transaction (pipelined): bus free btwn request & reply ⇒ fast
- *arbitration*: how do we decide who gets the bus next?
  - overlap arbitration for next master with current transfer
  - daisy chain: closer devices have priority ⇒ slow
  - distributed: wired-OR, low-priority back-off ⇒ medium
- some other issues
  - split data/address lines, width, burst transfer

Arbitration: Daisy Chain



Simple but not fair and slow

Others

- Centralized Parallel Arbitration
  - Requires central arbiter
  - Each device has separate line
  - Central arbiter may become bottleneck
  - Used in PCI bus
- Distributed Arbitration by Self Selection
  - Each device sees all requestors
  - Priority scheme allows each to know if they get bus
  - Requires lots of request lines

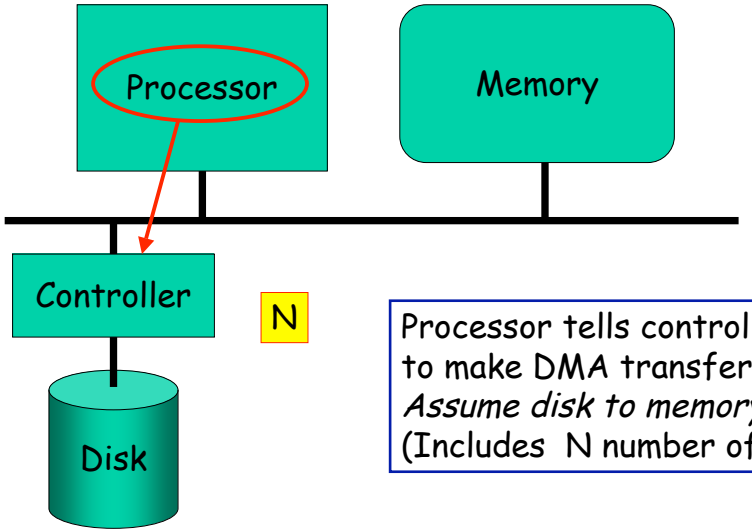
Others

- Distributed Arbitration by Collision Detection
  - Devices independently request bus
  - Devices have ability to detect simultaneous requests or collisions.
  - Upon collision a variety of schemes are used to select among requestors
  - Used by Ethernet

DMA

- Using sophisticated general-purpose processor for very specialized function
- Solution: Add enough processing power to device controller (and possibly bus controller) to allow direct transfer between device and memory.

DMA



Processor tells controller to make DMA transfer.
Assume disk to memory.
(Includes N number of bytes)