Challenge

• How do we interface to other devices
  – Keyboard
  – Mouse
  – Disk
  – Network
  – Printer
  – …
Communication Interface

- **Programmed I/O**
  - CPU has dedicated, special instructions
  - CPU has additional output signals (I/O bus)
  - Instruction specifies device and operation
  - Protection: I/O instructions are privileged

- **Memory-mapped I/O**
  - Device communication goes over the memory bus
  - Reads/Writes to special addresses are converted into I/O operations
  - Each device appears as if it is part of the memory address space
  - Protection: device protected by the MMU

Memory-Mapped I/O

![Memory-Mapped I/O Diagram](image)
Memory-Mapped I/O

- Basic Idea – Make control registers and I/O device memory appear to be part of the system’s main memory
  - Reads and writes to that region of the memory are translated by OS into device accesses
  - Easy to support variable numbers/types of devices
    - Managing new devices is now memory allocation
    - Example: accessing memory on a PCMCIA card
      - Once card memory mapped into address space, just hand out pointers like conventional memory

Processor Memory Maps

- Some OSes use fixed memory maps

- Others use variable mappings
  - Unix/Linux has virtual memory system, maps I/O devices onto memory only when requested. This allows more flexibility for a system to accommodate variable number I/O devices
I/O Controllers

- We could place every device on the memory bus
  - We would have to replace devices as we improve the memory bus

- We could decouple them from the memory bus
  - Via an “I/O controller”
  - Need to replace only the I/O controller when the CPU/Memory interface changes
Buses

• A bus is a shared collection of wires with multiple senders/receivers
  – Has an associated protocol, obeyed by senders and receivers, for sending and receiving data
  – Simple broadcast mechanism: all devices can see all transactions
  – Pros: cost-effective
  – Cons: communication bottleneck

• Standards to ensure compatibility of devices
  – USB, Firewire, etc.
• “The nice thing about standards is that there are so many to choose from” – Andy Tannenbaum

Bus Parameters

• Bus width
  – Number of wires, separate control/data?
• Data width
  – Number of bits per transfer
• Transfer size
  – Number of words per bus transaction
• Synchronous (clock), asynchronous (wider variety of devices)
  – Handshaking protocol
    • Sender/receiver proceed only if in agreement
Communication Method

- **Polling**
  - OS code executing on the CPU periodically checks the device for new activity

- **Interrupts**
  - Device sends interrupt
  - OS responds by communicating with the device
  - Uses exception handling hardware
  - Interrupt priority levels

Interrupt-Driven I/O

- **Interrupt enable bit** has to be on

- **Priority levels**
  - Slower devices have lower priority interrupts

- Don’t want processor involved with each byte
DMA: Direct Memory Access

Non-DMA transfer: I/O device $\leftrightarrow$ CPU $\leftrightarrow$ RAM
- `for (i = 1 .. n)`
  - CPU sends transfer request to device
  - I/O writes data to bus, CPU reads into registers
  - CPU writes data to registers to memory

DMA transfer: I/O device $\leftrightarrow$ RAM
- CPU sets up DMA request on device
- `for (i = 1 .. n)`
  - I/O device writes data to bus, RAM reads data

Based on lecture from Kevin Walsh

How to do DMA?

- DMA implemented with special controller
  - Transfers data between I/O device and memory
- Processor sets up DMA
  - Which device, operation, address, # of bytes
  - DMA
    - gets bus
    - Transfers data (maybe multiple requests)
  - DMA interrupts processor to signal end of I/O operation
DMA Issues (1): Addressing

Problem: device operates on bus or physical addresses, programs operate with virtual addresses

Solution:
– translate to physical addresses before DMA
– Introduces new problem: Multi-page transfers?
  ▪ Have to be contiguous in physical memory which is not realistic

DMA Issues (1): Addressing

Problem: device operates on bus or physical addresses, programs operate with virtual addresses

Alternate solution:
– add translation hardware to dma controllers
– a software-controlled, miniature TLB.
DMA Issues (2): Virtual Mem

Problem: DMA destination page may not be in memory (i.e. swapped to disk by virtual memory system)

Solution:
– *pin* the page before initiating dma transfer

Alternate solution:
– dma to a pinned kernel page, then memcopy elsewhere

DMA Issues (3): Legacy Cruft

DMA controller (or bus) can't support full size addresses
– many legacy DMA devices can use only 16-bit addresses
– Solution: DMA to dedicated low-addressed physical pages, then memcopy elsewhere
DMA Issues (4): Caches

What if L1 or L2 caches DMA-related data?
- DMA write to device from RAM: device gets stale data if cache is dirty
- DMA read from device to RAM: cache will have stale data

Solution: (software enforced coherence)
- flush entire cache (or part of cache) before DMA begins
- don't touch pages during DMA or mark pages as uncacheable in VM page tables (and update the TLB entries for those pages)

Alternate solution: (hardware cache coherence, aka snooping)
- cache controller listens on bus, and conspires with RAM
- DMA write: cache services request if it has the data, otherwise RAM services
- DMA read: cache invalidates data seen on the bus or, cache updates itself when it sees data on the bus

Kavita Bala, Computer Science, Cornell University
Memory Hierarchy

- **16 KB** registers/L1: 2 ns, random access
- **512 KB** L2: 5 ns, random access
- **2 GB** DRAM: 20-80 ns, random access
- **300 GB** Disk: 2-8 ms, random access
- **1 TB** Tape: 100s, sequential access
### Tapes

- Same basic principle for 8-tracks, cassettes, VHS, atari tape drive, tape storage

### Disks & CDs

- Disks use same magnetic medium as tapes
  - concentric rings
    (not a spiral)

- CDs & DVDs use optics, and a single spiral track

  - Non-volatile
Disk Physics

Typical parameters:
- 1 spindle
- 1 arm assembly
- 1-4 platters
- 1-2 sides/platter
- 1 head per side (but only 1 active head at a time)

4,200 – 15,000 RPM

Disk Accesses

Accessing a disk requires:
- specify sector: C (cylinder), H (head), and S (sector)
- specify size: number of sectors to read or write
- specify memory address: bus address to DMA to

Performance:
- seek time: move the arm assembly to track
- Rotational delay: wait for sector to come around
- transfer time: get the bits off the disk
Example

- Average time to read/write 512-byte sector
  - Disk rotation at 10,000 RPM
  - Seek time: 6ms
  - Transfer rate: 50 MB/sec
  - Controller overhead: 0.2 ms

- Average time:
  - Seek time + rotational delay + transfer time + controller overhead
  - 6ms + 0.5 rotation/(10,000 RPM) + 0.5KB/(50 MB/sec) + 0.2ms
  - 6.0 + 3.0 + 0.01 + 0.2 = 9.2ms

Disk Scheduling

- Goal: minimize seek time
  - secondary goal: minimize rotational latency

- FCFS (First come first served)

- Shortest seek time

- SCAN/Elevator
  - First service all requests in one direction
  - Then reverse and serve in opposite direction

- Circular SCAN
  - Go off the edge and come to the beginning and start all over again
Disk Geometry: LBA

New machines use *logical block addressing* instead of CHS
- machine presents illusion of an array of blocks, numbered 0 to N

Modern disks…
- have varying number of sectors per track
  - roughly constant data density over disk
  - varying throughput over disk
- remap and reorder blocks (to avoid defects)
- completely obscure their actual physical geometry

Modern Disk Geometry

- Not to scale: actual disk has
  - dozens or hundreds of sectors per track
  - tens of thousands of tracks