IO Devices

Communication with IO devices
- Processor needs to get info to/from IO device
  - Two ways:
    - In/out instructions
    - Read/write value to "io port"
    - Devices have specific port numbers
    - Memory mapped
    - Regions of physical addresses not actually in DRAM
      - But mapped to IO device
    - Stores to mapped addresses send info to device
    - Reads from mapped addresses get info from device

A view of the world
- Chip 0 requests read of 0x100100
- Request goes to all devices

IO: Interacting with the outside world
- Input and Output Devices
  - Video
  - Disk
  - Keyboard
  - Sound
  - ...

A view of the world
- Chip 0 requests read of 0x100100
- Request goes to all devices
A view of the world

• Chip 0 requests read of 0x100100
• Request goes to all devices, which check address ranges

Other address ranges may be for a particular device

Exploring Memory Mappings on Linux

• You can see what devices have what memory ranges on Linux with `lspci -v` (at least those on the PCI bus)

00:02.0 VGA compatible controller: Intel Corporation Core Processor Integrated Graphics Controller (rev 02)
  Subsystem: Lenovo Device 215a
  Flags: bus master, fast devsel, latency 0, IRQ 30
  Memory at f2000000 (64-bit, non-prefetchable) [size=4M]
  Memory at d0000000 (64-bit, prefetchable) [size=256M]
  I/O ports at 1800 [size=8]
  Capabilities: [90] Message Signalled Interrupts: Mask: 64bit: Queue=0/0
  Capabilities: [d0] Power Management version 2
  Capabilities: [a4] PCIe advanced features <?>
  Kernel driver in use: i915
  Kernel modules: i915

A simple “IO device” example

• Read (physical) address 0xFFFF1000 for “ready”
• If ready, read address 0xFFFF1004 for data value
• IO device will go to next value automatically on read
• Write a value to 0xFFFF1008 to output it

read_dev:
  la $t0, 0xFFFF1000
  loop:
    lw $t1, 0($t0)
    beqz $t1, loop
    lw $v0, 4($t0)
    jr $ra

Who can remind us what this is called (last lecture)?

A handful of questions…

• How do we use physical addresses?
  • Programs only know about virtual addresses right?

• What about caches?
  • Won’t the first lw bring the current value of 0xFFFF1000 into the cache?
  • And then subsequent requests just hit the cache?

• How do we use physical addresses?
  • Programs only know about virtual addresses right?
  • Only OS accesses IO devices:
    • OS knows about physical addresses, and can use them

• What about caches?
  • Won’t the first lw bring the current value of 0xFFFF1000 into the cache?
  • And then subsequent requests just hit the cache?
A handful of questions...

- How do we use physical addresses?
  - Programs only know about virtual addresses right?
  - Only OS accesses IO devices:
    - OS knows about physical addresses, and can use them

- What about caches?
  - Won’t the first \texttt{lw} bring the current value of 0xFFFF1000 into the cache?
  - And then subsequent requests just hit the cache?
  - Pages have attributes, including cacheability
    - IO mapped pages marked non-cacheable
    - Also, prevent speculative loads (e.g., out-of-order)
    - Remember: speculative only fine as long as nobody knows...

Hard disks

- Viewed from above:
  - Disks are circular platters of spinning metal
  - Multiple tracks (concentric rings)
  - Each track divided into sectors
  - Modern disks: addressed by "logical block"
    (Real disks are actually circular...)

Hard disks

- Read/written by "head"
  - Moves across tracks ("seek")
  - After seek completes, wait for proper sector to rotate under head.
  - Reads or writes magnetic medium by sensing/changing magnetic state (this takes time as the desired data ‘spins under’ the head)

Hard disks

- Want to read data on blue curve (imagine circular arc)
  - First step: seek—move head over right track
  - Takes time (\texttt{Tseek}), disk keeps spinning
  - Now head over right track... but data needs to move under head
  - Second step: wait (\texttt{Trotate})
Hard disks

- Want to read data on blue curve (imagine circular arc)
  - First step: seek—move head over right track
    - Takes time (Tseek), disk keeps spinning
  - Now head over right track... but data needs to move under head
  - Second step: wait (Trotate)
  - Third: as data comes under head, start reading

A few things about HDD performance

- Tseek:
  - Depends on how fast heads can move
  - And how far they have to go
  - OS may try to schedule IO requests to minimize Tseek
- Trotate:
  - Depends largely on how fast disk spins (RPM)
  - Also, how far around the data must spin, but usually assume avg
  - OS cannot keep track of position, nor schedule for better
- Tread:
  - Depends on RPM + how much data to read

Disk Drive Performance

- Suppose on average
  - Tseek = 10 ms
  - Trotate = 3.0 ms
  - Tread = 5 usec/ 512-byte sector
- What is the average time to read one 512-byte sector?
  - 10 ms + 3 ms + 0.005 ms = 13.05 ms
  - Reading 1 sector a a time: 512 byte/ 13.05 ms => ≈40KB/sec
- What is the average time to read 1MB of (contiguous) data?
  - 1MB = 2048 sectors
  - 10 + 3 + 0.005 * 2048 = 23.24 ms => ≈43MB/sec
**Disk Drive Performance**

- Suppose on average
  - Tseek = 10 ms
  - Trotate = 3 ms
  - Tread = 5 usec/512-byte sector

- What is the average time to read one 512-byte sector?
  - 10 ms + 3 ms + 0.005 ms = 13.005 ms
  - Reading 1 sector at a time: 512 byte/13.005 ms = ~40KB/sec

- What is the avg time to read 1MB of (contiguous) data?
  - 1MB = 2048 sectors
  - 10 + 3 + 0.005 * 2048 = 23.24 ms = ~43MB/sec

- Larger contiguous reads: approach **100MB/sec**

- Amortize Tseek + Trotate (key to good disk performance)

**Disk Performance**

- Hard disks have caches (spatial locality)
- OS will also buffer disk in memory
  - Ask to read 16 bytes from a file?
  - OS reads multiple KB, buffers in memory

- "Defragmenting" (Windows):
  - Improve locality by putting blocks for same files near each other

**Transferring the data to memory**

- OS asks disk to read data
  - Disk read takes a long time (15 ms => millions of cycles)
  - Does OS poll disk for 15M cycles looking for data?

- No—disk interrupts OS when data is ready.

- Ready: version 1
  - Disk has data, needs it transferred to memory
  - OS does "memory" like routine:
    - Read disk memory mapped I/O
    - Write appropriate location in main memory
    - Repeat
    - For many KB to a few MB
## DMA: Direct Memory Access
- Alternative: DMA
  - When OS requests disk read, sets up DMA
  - "Read this data from the disk, and put it in memory for me"
- DMA controller handles "memcpy"
- Ready (version 2.0): data is in memory
- Frees up CPU to do useful things

## Hard disk: reliability
- Hard disks fail relatively easily
  - Spinning piece of metal
  - With head hovering <1mm from platter
- Hard drive failures: major pain...
  - Anyone ever have one?

## Reliability
- Solution to functionality problem?
  - Level of indirection
- Solution to performance problem?
  - Add a cache
- Solution to a reliability problem?
  - ...?

## RAID: Reliability
- Redundant Array of In-expensive Disks (RAID)
  - Keep 2 hard-drives with identical copies of the data
  - One fails? Replace it, copy the other drive to it, resume
    - Can work from other drive while waiting for replacement
    - Performance?
**RAID: Reliability**

- Redundant Array of In-expensive Disks (RAID)
  - Keep 2 hard-drives with identical copies of the data
  - One fails? Replace it, copy the other drive to it, resume
  - Can work from other drive while waiting for replacement
- Performance?
  - Writes to both drives in parallel (no cost)
  - Reads from either drive
  - Improve performance: twice the bandwidth
- Downside?
  - Cost: need to buy 2x as many disks for 1x the space
  - Still: pretty popular (I have it on my home linux box)
  - Also very easy

**RAID: All sorts of things**

- Mirroring data (prev slides): "RAID 1"
- Tons of other RAID configurations:
  - RAID 0: striping—performance, not reliability
  - Parity schemes: reduce overhead for num disks > 2
  - Still give reliability and good performance
- Many covered in detail in your book
  - Good to know they exist, may be good solution to a problem one day
  - Don’t fret the obscure ones too much

**Other devices**

- Wide variety of IO devices
  - Most basically work the same way from high-level
  - Read/write proper physical memory location(s)
- Reality: each device has its own protocol
  - Requires device driver: Software module that handles protocol details of specific device
  - Which memory locations to read/write etc.
  - Example of?

**Next Up: Pipelines**

- Next week (last week of class)
- Pipelines
  - Slightly more realistic datapaths
  - Overlap instructions for higher performance
- Now: course evaluations