CSE 506: Operating Systems

What Hardware Provides to the OS
What Hardware Provides to the OS

- Memory
- Instructions
- Interrupts
- Video & Keyboard
- DMA
- PCI
- Disk
- Network
(Physical) Memory

• Machine includes some amount of physical memory
  – Provides blocks of sequential addresses
  – Sometimes has “holes” in it
    • BIOS can provide a list of available physical addresses
    • On x86, called e820
      • Usually determined by boot loader and passed as arg. to kernel

• Non-Uniform Memory Access (NUMA)
  – Some memory can be slower than others
    • Example: 2 CPUs (nodes), each with its own 32GB of RAM
  – Can be leveraged to avoid performance loss
    • OS can allocate local memory to avoid remote access
    • Or BIOS can interleave memory across nodes
      – All accesses are penalized roughly equally
(Virtual) Memory

• Hardware can manage virtual address spaces
  – Dictated by the OS through page tables
  – Page tables map from virtual to physical addresses
    • On x86, 4KB is common, 2MB is sometimes used, 1GB is possible

• Page tables dictate translation
  – Stored in memory
  – Indexed by virtual page index
    • Includes physical page index and permission bits
  – Uses hierarchical structure to reduce storage
Translation Overview

• From Intel 80386 Reference Programmer’s Manual
  – x86-64 is similar, but with 4 levels
Example

Page Dir Offset (Top 10 addr bits: 0xf10 >> 2)

Page Table Offset (Next 10 addr bits)

Physical Page Offset (Low 12 addr bits)

Entry at cr3+0x3b4 * sizeof(PTE)

Entry at 0x84 * sizeof(PTE)

Data we want at offset 0x150
Page Table Entries

- 12 bits not needed (4K pages have 12 bit offset)
  - Convenient place for flags

<table>
<thead>
<tr>
<th>Upper Physical Address Bits (40 bits)</th>
<th>Flags (12 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00384</td>
<td>PTE_W</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0x28370</td>
<td>PTE_W</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Page Table Flags

• 3 for OS to use however it likes
• 4 reserved by Intel, just in case
• 3 for OS to CPU metadata
  – User/vs kernel page,
  – Write permission,
  – Present bit (so we can swap out pages)
• 2 for CPU to OS metadata
  – Dirty (page was written), Accessed (page was read)
TLB Entries

• The CPU caches address translations in the TLB
  – *Translation Look-aside Buffer*

• Not coherent with memory
  – If you change a PTE, you need to invalidate TLB entry

Page Traversal is *Slow*  
Table Lookup is *Fast*
No execute (NX) bit

• Many security holes arise from bad input
  – Tricks program to jump to unintended address
  – That happens to be on heap or stack
  – And contains bits that form malware
• Idea: execute protection can catch these
• Bit 63 in 64-bit page tables
What Hardware Provides to the OS

- Memory
- Instructions
- Interrupts
- Video & Keyboard
- DMA
- PCI
- Disk
- Network
Instructions

• Bytes encoding an operation
• Basic instructions manipulate registers and/or memory
  – Can execute at any time (e.g., in ring 3 on x86)
  – Trigger system calls (traps or software interrupts)
• Special instructions manipulate other hardware
  – Usually only execute in privileged mode (ring 0)
  – Output and input to ports (special I/O space, going away slowly)
  – Model-Specific Registers (rdmsr, wrmsr) for hardware config
  – CRx registers
    • CR2 provides address of faulting instruction
    • CR3 sets base address of top-level page table
  – LIDT sets location of Interrupt Descriptor Table (IDT)
What Hardware Provides to the OS

• Memory
• Instructions
• Interrupts
• Video & Keyboard
• DMA
• PCI
• Disk
• Network
Interrupts

- Umbrella term (used to mean many things)
  - **Interrupts**: spontaneous hardware events (e.g., keypress)
  - **Exceptions**: events in response to insn (e.g., div/0, gpf)
  - **Traps**: events explicitly requested by insn (**swi**)

- Handling new interrupts can be suspended (**cli**)

- Hardware jumps to handler routine
  - Event number selects index into table
    - Index contains pointer and some flags
    - Flags may signal automatic “**cli**” on handler entry
  - If event occurs in ring 3, also switches stack (**%rsp**)
    - Stack pointer taken from TSS via GDT
Hardware Interrupts

• PIC: “old” style (slightly easier)
  – 8 wires, each wire goes to different device
  – Can be programmed to “mask” interrupts
  – Needs to receive ack before triggering new interrupt
  – Cascaded for more interrupts
    • Two 8-input PICs chained together, one master one slave
    • If interrupt from slave, must ack both master and slave
  – Programmable with “offset” to add to IRQ number
    • Must be programmed before use
      – Usually map master to 32, slave to 40

• APIC: “new” style (more complex)
  – One LAPIC (Local APIC) per core, one IOAPIC per chip
  – Supports Inter-Processor Interrupts (IPI), needed for MP
What Hardware Provides to the OS

• Memory
• Instructions
• Interrupts
• Video & Keyboard
• DMA
• PCI
• Disk
• Network
Video and Keyboard

- x86 text console
  - Memory mapped at 0xB8000
  - 80*50 bytes (80 cols, 25 rows, 2 bytes per location)
    - One byte for color, one byte for glyph

- PS/2 keyboard
  - Wired to IRQ#1
  - Needs read of port 0x60 to ack interrupt
  - Read byte is a scan code, not a letter
    - Letters on key caps are just a convention
  - Always get two interrupts per keypress
    - One for press (high bit clear) one for release (high bit set)
Keyboard via SSH and Curses

• We are working in an emulated environment
  – Which isn’t even on our local machines

• Must keep in mind keys such as Shift, Ctrl, and Alt
  – On a local keyboard, these behave like all other keys
  – In an ssh session, scan codes are not sent to destination
    • Local OS interprets scan codes, characters are sent via ssh

• Qemu in *curses* mode fakes keyboard scan codes
  – Qemu translates characters to scan codes
    • User presses Shift+a
    • Character “A” is transmitted over the ssh session
    • Qemu fakes “shift down, a down, a up, shift up” sequence
What Hardware Provides to the OS

- Memory
- Instructions
- Interrupts
- Video & Keyboard
- DMA
- PCI
- Disk
- Network
Direct Memory Access (DMA)

- I/O devices need to xfer data to/from memory
  - CPU looping for every word (read/write) is expensive
  - CPU can delegate xfer and get IRQ when xfer is done

- Originally, done with special DMA controller
  - CPU programmed controller
  - Controller performed xfer device ↔ memory
  - Legacy setup, *don’t accidentally try to implement it*
    - ISA – Industry Standard Architecture

- PCI / PCIe are dominant today
  - Each device has its own DMA controller
  - Called *bus mastering*
Typical DMA Operation

• CPU creates a *ring buffer*

• To receive data
  – Physical addresses given to device
  – When device has data (e.g., received network packet)
    • Device copies data directly into memory
    • Raises hardware IRQ
  – CPU replaces used buffer with new empty one

• To send data:
  – CPU prepares data in buffer (can take buffer out of ring)
  – Physical address given to device and xfer triggered
  – When xfer done, device raises IRQ (CPU reclaims buffer)
What Hardware Provides to the OS

- Memory
- Instructions
- Interrupts
- Video & Keyboard
- DMA
- PCI
- Disk
- Network
Peripheral Component Interconnect

- Slowly getting replaced by **PCIe (PCI Express)**
  - Same general idea, higher performance
- Replaces previous buses (e.g., ISA)
  - PCI is software-configurable
  - Opposite of ISA, which was hard-wired (e.g., with jumpers)
- OS should walk PCI **configuration space**
  - Enumerates all PCI devices on all PCI buses
  - Each device reports at least Vendor ID and Device ID
    - Used to select driver to load
    - Driver can configure up to 6 **BARs (Base Address Registers)**
      - BARs dictate the device’s memory-mapped I/O addresses
What Hardware Provides to the OS

- Memory
- Instructions
- Interrupts
- Video & Keyboard
- DMA
- PCI
- Disk
- Network
Disks

• Organized into *sectors*
  – C/H/S (cylinder, head sector) - mostly outdated notion
  – Modern disks addressed by *logical block address (LBA)*
  – Typically sectors were 512 bytes, newer disks use 4KB

• A sector is the minimum unit to read/write
  – Most OSes typically read/write entire memory page (4K)

• Can be grouped together into logical units
  – *Redundant Array of Independent Disks (RAID)*
    • RAID 1 (disk mirror): tolerate N-1 disk failures, improve read perf.
    • RAID 5 (striped w/parity): tolerate 1 failure
    • Hot spare: provides unused disk until failure (tolerate +1 failure)
Disk Controllers

• Disks connected via controller
  – IDE: just a bridge from ISA bus to on-disk controller
    • Evolved into ATA-x, renamed Parallel ATA (PATA), Serial ATA (SATA)
  – SCSI: actual “smart” disk controller
    • Many revisions, most recently to Serial Attached SCSI (SAS)

• SATA uses IDE protocol, SAS uses SCSI protocol

• Provides commands such as
  – Read/Write, Start/Stop, Inquiry

• Modern controllers provide tagged commands
  – Each read/write request has a tag
  – Allows for multiple operations to be outstanding
What Hardware Provides to the OS

- Memory
- Instructions
- Interrupts
- Video & Keyboard
- DMA
- PCI
- Disk
- Network
Network Cards

• “Physically” connects machine to network
• Typically support only low-level operations
  – Receive data from wire, put data into ring buffer
  – Take buffer out of ring, send contents on the wire
  – Raise interrupt when ready
• Modern NICs have many add-on features
  – Interrupt coalescing: don’t raise interrupt for each packet
  – TCP Offload Engine (TOE): do some TCP work on the NIC
    • Checksum Offload: compute checksum in hardware
    • TCP Segmentation Offload (TOS): split large buffers into packets
    • ...