Recall: Thrashing

• If a process does not have "enough" pages, the page-fault rate is very high. This leads to:
  – low CPU utilization
  – operating system spends most of its time swapping to disk
• Thrashing ≡ a process is busy swapping pages in and out
• Questions:
  – How do we detect Thrashing?
  – What is best response to Thrashing?

Locality In A Memory-Reference Pattern

• Program Memory Access Patterns have temporal and spatial locality
  – Group of Pages accessed along a given time slice called the "Working Set"
  – Working Set defines minimum number of pages needed for process to behave well
• Not enough memory for Working Set ⇒ Thrashing
  – Better to swap out process?

Working-Set Model

• $\Delta \equiv$ working-set window ≡ fixed number of page references
  – Example: 10,000 instructions
• $WS_i$ (working set of Process $P_i$) ≡ total set of pages referenced in the most recent $\Delta$ (varies in time)
  – if $\Delta$ too small will not encompass entire locality
  – if $\Delta$ too large will encompass several localities
  – if $\Delta = \infty \Rightarrow$ will encompass entire program
• $D = \sum |WS_i|$ ≡ total demand frames
• if $D > m \Rightarrow$ Thrashing
  – Policy: if $D > m$, then suspend/swap out processes
  – This can improve overall system behavior by a lot!
What about Compulsory Misses?

- Recall that compulsory misses are misses that occur the first time that a page is seen
  - Pages that are touched for the first time
  - Pages that are touched after process is swapped out/swapped back in

- Clustering:
  - On a page-fault, bring in multiple pages “around” the faulting page
  - Since efficiency of disk reads increases with sequential reads, makes sense to read several sequential pages

- Working Set Tracking:
  - Use algorithm to try to track working set of application
  - When swapping process back in, swap in working set

Reverse Page Mapping (Sometimes called “Coremap”)

- Physical page frames often shared by many different address spaces/page tables
  - All children forked from given process
  - Shared memory pages between processes

- Whatever reverse mapping mechanism that is in place must be very fast
  - Must hunt down all page tables pointing at given page frame when freeing a page
  - Must hunt down all PTEs when seeing if pages “active”

- Implementation options:
  - For every page descriptor, keep linked list of page table entries that point to it
    » Management nightmare – expensive
  - Linux 2.6: Object-based reverse mapping
    » Link together memory region descriptors instead (much coarser granularity)

Linux Memory Details?

- Memory management in Linux considerably more complex than the previous indications
- Memory Zones: physical memory categories
  - ZONE_DMA: < 16MB memory, DMAable on ISA bus
  - ZONE_NORMAL: 16MB → 896MB (mapped at 0xC0000000)
  - ZONE_HIGHMEM: Everything else (> 896MB)
- Each zone has 1 freelist, 2 LRU lists (Active/Inactive)
- Many different types of allocation
  - SLAB allocators, per-page allocators, mapped/unmapped
- Many different types of allocated memory:
  - Anonymous memory (not backed by a file, heap/stack)
  - Mapped memory (backed by a file)
- Allocation priorities
  - Is blocking allowed/etc

Recall: Linux Virtual memory map

- 32-Bit Virtual Address Space
  - 0xFF000000 - 0xFFFF00000000
  - 4GB Total

- 64-Bit Virtual Address Space
  - 0x0000000000000000 - 0xFFFFFFFFFFFFFFFF
  - 128TiB

- Physical Addresses
  - 0xC0000000 - 0xFFFFFFFF
  - 128TiB

- Kernel Addresses
  - 0xFFFF800000000000 - 0xFFFFFFFFFFFFFFFF
  - 64 TiB

- Empty Space
  - “Canonical Hole”
The Requirements of I/O

• So far in this course:
  – We have learned how to manage CPU and memory

• What about I/O?
  – Without I/O, computers are useless (disembodied brains?)
  – But… thousands of devices, each slightly different
    » How can we standardize the interfaces to these devices?
  – Devices unreliable: media failures and transmission errors
    » How can we make them reliable??!
  – Devices unpredictable and/or slow
    » How can we manage them if we don’t know what they will do or how they will perform?

In a Picture

• I/O devices you recognize are supported by I/O Controllers
• Processors accesses them by reading and writing IO registers as if they were memory
  – Write commands and arguments, read status and results

Operational Parameters for I/O

• Data granularity: Byte vs. Block
  – Some devices provide single byte at a time (e.g., keyboard)
  – Others provide whole blocks (e.g., disks, networks, etc.)

• Access pattern: Sequential vs. Random
  – Some devices must be accessed sequentially (e.g., tape)
  – Others can be accessed “randomly” (e.g., disk, cd, etc.)
    » Fixed overhead to start transfers
  – Some devices require continual monitoring
  – Others generate interrupts when they need service

• Transfer Mechanism: Programmed IO and DMA
Kernel Device Structure

The System Call Interface

- Process Management
- Memory Management
- Filesystems
- Device Control
- Networking

Concurrency, multitasking
Virtual memory
Files and dirs: the VFS
TTYS and device access
Connectivity
Network Subsystem
Block Devices
IF drivers

Want Standard Interfaces to Devices

- **Block Devices**: e.g. disk drives, tape drives, DVD-ROM
  - Access blocks of data
  - Commands include `open()`, `read()`, `write()`, `seek()`
  - Raw I/O or file-system access
  - Memory-mapped file access possible

- **Character Devices**: e.g. keyboards, mice, serial ports, some USB devices
  - Single characters at a time
  - Commands include `get()`, `put()`
  - Libraries layered on top allow line editing

- **Network Devices**: e.g. Ethernet, Wireless, Bluetooth
  - Different enough from block/character to have own interface
  - Unix and Windows include `socket` interface
  » Separates network protocol from network operation
  » Includes `select()` functionality
  - Usage: pipes, FIFOs, streams, queues, mailboxes

The Goal of the I/O Subsystem

- Provide Uniform Interfaces, Despite Wide Range of Different Devices
  - This code works on many different devices:
    ```cpp
    FILE fd = fopen("/dev/something", "rw");
    for (int i = 0; i < 10; i++) {
      fprintf(fd, "Count %d\n", i);
    }
    close(fd);
    ```
  - Why? Because code that controls devices (“device driver”) implements standard interface

- We will try to get a flavor for what is involved in actually controlling devices in rest of lecture
  - Can only scratch surface!

How Does User Deal with Timing?

- **Blocking Interface**: “Wait”
  - When request data (e.g. `read()` system call), put process to sleep until data is ready
  - When write data (e.g. `write()` system call), put process to sleep until device is ready for data

- **Non-blocking Interface**: “Don’t Wait”
  - Returns quickly from read or write request with count of bytes successfully transferred
  - Read may return nothing, write may write nothing

- **Asynchronous Interface**: “Tell Me Later”
  - When request data, take pointer to user’s buffer, return immediately; later kernel fills buffer and notifies user
  - When send data, take pointer to user’s buffer, return immediately; later kernel takes data and notifies user
Chip-scale Features of 2015 x86 (Sky Lake)

- Significant pieces:
  - Four OOO cores with deeper buffers
    » New Intel MPX (Memory Protection Extensions)
    » New Intel SGX (Software Guard Extensions)
    » Issue up to 6 µ-ops/cycle
  - Integrated GPU, System Agent (Mem, Fast I/O)
    » Large shared L3 cache with on-chip ring bus
    » 2 MB/core instead of 1.5 MB/core
    » High-BW access to L3 Cache
  - Integrated I/O
    » Integrated memory controller (IMC)
      » Two independent channels of DDR3L/DDR4 DRAM
      » High-speed PCI-Express (for Graphics cards)
      » Direct Media Interface (DMI) Connection to PCH (Platform Control Hub)

Sky Lake I/O: PCH

- Platform Controller Hub
  » Connected to processor with proprietary bus
    » Direct Media Interface
- Types of I/O on PCH:
  » USB, Ethernet
  » Thunderbolt 3
  » Audio, BIOS support
  » More PCI Express (lower speed than on Processor)
  » SATA (for Disks)

Modern I/O Systems

Example: PCI Architecture
Example Device-Transfer Rates in Mb/s (Sun Enterprise 6000)

- Device Rates vary over 12 orders of magnitude !!!
  - System better be able to handle this wide range
  - Better not have high overhead/byte for fast devices!
  - Better not waste time waiting for slow devices

How does the processor actually talk to the device?

- CPU interacts with a Controller
  - Contains a set of registers that can be read and written
  - May contain memory for request queues or bit-mapped images
- Regardless of the complexity of the connections and buses, processor accesses registers in two ways:
  - I/O instructions: in/out instructions
    » Example from the Intel architecture: out 0x21, AL
  - Memory mapped I/O: load/store instructions
    » Registers/memory appear in physical address space
    » I/O accomplished with load and store instructions

Example: Memory-Mapped Display Controller

- Memory-Mapped:
  - Hardware maps control registers and display memory into physical address space
    » Addresses set by HW jumpers or at boot time
  - Simply writing to display memory (also called the “frame buffer”) changes image on screen
    » Addr: 0x8000F000 – 0x8000FFFF
  - Writing graphics description to cmd queue
    » Say enter a set of triangles describing some scene
    » Addr: 0x80010000 – 0x8001FFFF
  - Writing to the command register may cause on-board graphics hardware to do something
    » Say render the above scene
    » Addr: 0x0007F004
- Can protect with address translation

Administrivia

- Midterm 2 coming up on **THURSDAY 3/22 8-10:00PM**
  - All topics up to and including Lecture 16
    » Focus will be on Lectures 10 – 16 and associated readings
    » Projects 1 and 2
    » Homework 0 – 2
  - Closed book
  - 1 page hand-written notes both sides
  - Room assignments posted on Piazza
    » 20 / 126 / 170 Barrows, 155 Kroeber, 101 Moffitt, 105 North Gate
Transferring Data To/From Controller

- **Programmed I/O:**
  - Each byte transferred via processor in/out or load/store
  - Pro: Simple hardware, easy to program
  - Con: Consumes processor cycles proportional to data size

- **Direct Memory Access:**
  - Give controller access to memory bus
  - Ask it to transfer data blocks to/from memory directly

- Sample interaction with DMA controller (from OSC book):

---

I/O Device Notifying the OS

- The OS needs to know when:
  - The I/O device has completed an operation
  - The I/O operation has encountered an error

- **I/O Interrupt:**
  - Device generates an interrupt whenever it needs service
  - Pro: handles unpredictable events well
  - Con: interrupts relatively high overhead

- **Polling:**
  - OS periodically checks a device-specific status register
    - I/O device puts completion information in status register
    - Pro: low overhead
    - Con: may waste many cycles on polling if infrequent or unpredictable I/O operations

- Actual devices combine both polling and interrupts
  - For instance – High-bandwidth network adapter:
    - Interrupt for first incoming packet
    - Poll for following packets until hardware queues are empty
Device Drivers

- **Device Driver**: Device-specific code in the kernel that interacts directly with the device hardware
  - Supports a standard, internal interface
  - Same kernel I/O system can interact easily with different device drivers
  - Special device-specific configuration supported with the `ioctl()` system call

- Device Drivers typically divided into two pieces:
  - **Top half**: accessed in call path from system calls
    - Implements a set of standard, cross-device calls like `open()`, `close()`, `read()`, `write()`, `ioctl()`, `strategy()`
    - This is the kernel's interface to the device driver
    - Top half will start I/O to device, may put thread to sleep until finished
  - **Bottom half**: run as interrupt routine
    - Gets input or transfers next block of output
    - May wake sleeping threads if I/O now complete

---

Basic Performance Concepts

- **Response Time or Latency**: Time to perform an operation(s)

- **Bandwidth or Throughput**: Rate at which operations are performed (op/s)
  - Files: MB/s, Networks: Mb/s, Arithmetic: GFLOP/s

- **Start up or “Overhead”**: time to initiate an operation

- Most I/O operations are roughly linear in n bytes
  - Latency(n) = Overhead + n/TransferCapacity

---

Life Cycle of An I/O Request

1. User Program
2. Kernel I/O Subsystem
3. Device Driver Top Half
4. Device Driver Bottom Half
5. Device Hardware

---

Example (Fast Network)

- Consider a 1 Gb/s link ($B = 125$ MB/s)
  - With a startup cost $S = 1$ ms

- **Performance of gbps link with 1 ms startup**

- Latency(n) = $S + n/B$
- Bandwidth = $n/(S + n/B) = B \times n/(B \times S + n) = B/(B \times S/n + 1)$
**Example (Fast Network)**

- Consider a 1 Gb/s link ($B = 125$ MB/s)
  - With a startup cost $S = 1$ ms

  - Bandwidth = $B/(B*S/n + 1)$
  - half-power point occurs at $n=S*B \Rightarrow$ Bandwidth = $B/2$

**Example: at 10 ms startup (like Disk)**

- Performance of gbps link with 10 ms startup

  - $n = 1,250,000$ bytes!

**What Determines Peak BW for I/O ?**

- **Bus Speed**
  - PCI-X: 1064 MB/s = 133 MHz x 64 bit (per lane)
  - ULTRA WIDE SCSI: 40 MB/s
  - Serial Attached SCSI & Serial ATA & IEEE 1394 (firewire): 1.6 Gb/s full duplex (200 MB/s)
  - USB 3.0 – 5 Gb/s
  - Thunderbolt 3 – 40 Gb/s

- **Device Transfer Bandwidth**
  - Rotational speed of disk
  - Write / Read rate of NAND flash
  - Signaling rate of network link

- Whatever is the bottleneck in the path...

**Storage Devices**

- **Magnetic disks**
  - Storage that rarely becomes corrupted
  - Large capacity at low cost
  - Block level random access (except for SMR – later!)
  - Slow performance for random access
  - Better performance for sequential access

- **Flash memory**
  - Storage that rarely becomes corrupted
  - Capacity at intermediate cost (5-20x disk)
  - Block level random access
  - Good performance for reads; worse for random writes
  - Erasure requirement in large blocks
  - Wear patterns issue
The Amazing Magnetic Disk

- Unit of Transfer: Sector
  - Ring of sectors form a track
  - Stack of tracks form a cylinder
  - Heads position on cylinders

- Disk Tracks ~ 1 µm (micron) wide
  - Wavelength of light is ~ 0.5 µm
  - Resolution of human eye: 50 µm
  - 100K tracks on a typical 2.5” disk

- Separated by unused guard regions
  - Reduces likelihood neighboring tracks are corrupted during writes (still a small non-zero chance)

Review: Magnetic Disks

- Cylinders: all the tracks under the head at a given point on all surface

- Read/write data is a three-stage process:
  - Seek time: position the head/arm over the proper track
  - Rotational latency: wait for desired sector to rotate under r/w head
  - Transfer time: transfer a block of bits (sector) under r/w head

Disk Performance Example

- Assumptions:
  - Ignoring queuing and controller times for now
  - Avg seek time of 5ms,
  - 7200RPM ⇒ Time for rotation: 60000 (ms/minute) / 7200(rev/min) = 8 ms
  - Transfer rate of 4MByte/s, sector size of 1 Kbyte ⇒
    1024 bytes/4 × 10^6 (bytes/s) = 256 × 10^-6 sec ≈ .26 ms

- Read sector from random place on disk:
  - Seek (5ms) + Rot. Delay (4ms) + Transfer (0.26ms)
  - Approx 10 ms to fetch/put data: 100 KByte/sec

- Read sector from random place in same cylinder:
  - Rot. Delay (4ms) + Transfer (0.26ms)
  - Approx 5 ms to fetch/put data: 200 KByte/sec

- Read next sector on same track:
  - Transfer (0.26ms): 4 MByte/sec

- Key to using disk effectively (especially for file systems) is to minimize seek and rotational delays
Typical Numbers for Magnetic Disk

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Info / Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space/Density</td>
<td>Space: 8TB (Seagate), 10TB (Hitachi) in 3½ inch form factor! ( \text{Areal Density: 1 Terabit/square inch! (SMR, Helium, …)} )</td>
</tr>
<tr>
<td>Average seek time</td>
<td>Typically 5-10 milliseconds. Depending on reference locality, actual cost may be 25-33% of this number.</td>
</tr>
<tr>
<td>Average rotational latency</td>
<td>Most laptop/desktop disks rotate at 3600-7200 RPM (16-8 ms/rotation). Server disks up to 15,000 RPM. Average latency is halfway around disk so 8-4 milliseconds</td>
</tr>
<tr>
<td>Controller time</td>
<td>Depends on controller hardware.</td>
</tr>
<tr>
<td>Transfer time</td>
<td>Typically 50 to 100 MB/s. Depends on: - Transfer size (usually a sector): 512B – 1KB per sector - Rotation speed: 3600 RPM to 15000 RPM - Recording density: bits per inch on a track - Diameter: ranges from 1 in to 5.25 in</td>
</tr>
<tr>
<td>Cost</td>
<td>Used to drop by a factor of two every 1.5 years (or even faster); now slowing down</td>
</tr>
</tbody>
</table>

(Lots of) Intelligence in the Controller

- Sectors contain sophisticated error correcting codes
  - Disk head magnet has a field wider than track
  - Hide corruptions due to neighboring track writes
- Sector sparing
  - Remap bad sectors transparently to spare sectors on the same surface
- Slip sparing
  - Remap all sectors (when there is a bad sector) to preserve sequential behavior
- Track skewing
  - Sector numbers offset from one track to the next, to allow for disk head movement for sequential ops

Seagate Enterprise

10 TB (2016)
- 7 platters, 14 heads
- 7200 RPMs
- 6 Gbps SATA /12Gbps SAS interface
- 220MB/s transfer rate, cache size: 256MB
- Helium filled: reduce friction and power usage
- Price: $500 ($0.05/GB)

IBM Personal Computer/AT (1986)
- 30 MB hard disk
- 30-40ms seek time
- 0.7-1 MB/s (est.)
- Price: $500 ($17K/GB, 340,000x more expensive !!)

HDD vs SSD Comparison

SSD prices drop much faster than HDD
### Largest SSDs

- 60TB (2016)
- Dual port: 16Gbs
- Seq reads: 1.5GB/s
- Seq writes: 1GB/s
- Random Read Ops (IOPS): 150K
- Price: ~ $20K ($0.33/GB)

### Summary

- **I/O Devices Types:**
  - Many different speeds (0.1 bytes/sec to GBytes/sec)
  - Different Access Patterns:
    - Block Devices, Character Devices, Network Devices
  - Different Access Timing:
    - Blocking, Non-blocking, Asynchronous
- **I/O Controllers:** Hardware that controls actual device
  - Processor Accesses through I/O instructions, load/store to special physical memory
- **Notification mechanisms**
  - Interrupts
  - Polling: Report results through status register that processor looks at periodically
- **Device drivers interface to I/O devices**
  - Provide clean Read/Write interface to OS above
  - Manipulate devices through PIO, DMA & interrupt handling
  - Three types: block, character, and network