Review: Memory Accesses
Impact of Program Structure on Memory Performance

- Consider an array named *data* with 128*128 elements
- Each row is stored in one page (of size 128 words)
Impact of Program Structure on Memory Performance

- Consider an array named data with 128*128 elements
- Each row is stored in one page (of size 128 words)
- **Program 1**
  ```c
  for (j = 0; j < 128; j++)
      for (i = 0; i < 128; i++)
          data[i][j] = 0;
  ```

128 x 128 = **16,384** page faults
Impact of Program Structure on Memory Performance

- Consider an array named `data` with 128*128 elements
- Each row is stored in one page (of size 128 words)

**Program 1**

```c
for (j = 0; j < 128; j++)
    for (i = 0; i < 128; i++)
        data[i][j] = 0;
```

128 x 128 = **16,384** page faults

**Program 2**

```c
for (i = 0; i < 128; i++)
    for (j = 0; j < 128; j++)
        data[i][j] = 0;
```

Only **128** page faults
I/O Devices
Why I/O?

- I/O == Input/Output

- What good is a computer without any I/O devices?
  - Keyboard, display, disks…
Why I/O?

- I/O == Input/Output

- What good is a computer without any I/O devices?
  - Keyboard, display, disks...

- We want
  - **Hardware**: which will provide direct physical interfaces
  - **OS**: which can interact with different combinations
Prototypical System Architecture

- CPU
- Memory

Memory Bus (proprietary)
Prototypical System Architecture

- CPU
- Memory
- Memory Bus (proprietary)
- General I/O Bus (e.g., PCI)
- Graphics
Prototypical System Architecture

- CPU
- Memory
- Memory Bus (proprietary)
- General I/O Bus (e.g., PCI)
- Graphics
- Peripheral I/O Bus (e.g., SCSI, SATA, USB)
Canonical I/O Device
Canonical I/O Device

OS reads from and writes to these

Registers  Status  Command  Data

Interface

Internals
Canonical I/O Device

OS reads from and writes to these

<table>
<thead>
<tr>
<th>Registers</th>
<th>Status</th>
<th>Command</th>
<th>Data</th>
</tr>
</thead>
</table>

Micro-controller (CPU)
Memory (DRAM or SRAM or both)
Other Hardware-specific Chips
Hard Disk Drive PCB Example
A Basic I/O Protocol

while (STATUS == BUSY)
    ; // spin
Write data to DATA register
Write command to COMMAND register
while (STATUS == BUSY)
    ; // spin
A Basic I/O Protocol

while (STATUS == BUSY) //1
   ; // spin
Write data to DATA register //2
Write command to COMMAND register //3
while (STATUS == BUSY) //4
   ; // spin
A Basic I/O Protocol

Process A wants to do I/O

CPU

A

Disk

C

while (STATUS == BUSY) //1
; // spin
Write data to DATA register //2
Write command to COMMAND register //3
while (STATUS == BUSY) //4
; // spin
A Basic I/O Protocol

while (STATUS == BUSY) //1
 ; // spin
Write data to DATA register //2
Write command to COMMAND register //3
while (STATUS == BUSY) //4
 ; // spin
A Basic I/O Protocol

while (STATUS == BUSY) //1
    ; // spin
Write data to DATA register //2
Write command to COMMAND register //3
while (STATUS == BUSY) //4
    ; // spin
A Basic I/O Protocol

while (STATUS == BUSY) //1
    ; // spin
Write data to DATA register //2
Write command to COMMAND register //3
while (STATUS == BUSY) //4
    ; // spin
A Basic I/O Protocol

while (STATUS == BUSY) //1
    ; // spin
Write data to DATA register //2
Write command to COMMAND register //3
while (STATUS == BUSY) //4
    ; // spin
Interrupts

while (STATUS == BUSY) //1
    wait for interrupt;
Write data to DATA register //2
Write command to COMMAND register //3
while (STATUS == BUSY) //4
    wait for interrupt;
Interrupts

while (STATUS == BUSY) //1
    wait for interrupt;

Write data to DATA register //2

Write command to COMMAND register //3

while (STATUS == BUSY) //4
    wait for interrupt;
Interrupts vs. Polling

- Any potential issues for interrupts?
Interrupts vs. Polling

- Any potential issues for interrupts?

- Interrupts can lead to **livelock**
  - E.g., flood of network packets
Interrupts vs. Polling

- Any potential issues for interrupts?

- Interrupts can lead to **livelock**
  - E.g., flood of network packets

- Techniques
  - Hybrid approach: polling + interrupts
  - Interrupt coalescing: batching a bunch interrupts in one go
Where else Can We Optimize?

while (STATUS == BUSY)  //1
    wait for interrupt;
Write data to DATA register  //2
Write command to COMMAND register  //3
while (STATUS == BUSY)  //4
    wait for interrupt;
while (STATUS == BUSY) //1
    wait for interrupt;

Write data to DATA register //2

Write command to COMMAND register //3

while (STATUS == BUSY) //4
    wait for interrupt;
Programmed I/O vs. Direct Memory Access

- PIO (Programmed I/O)
  - CPU directly tells device what data is
  - CPU involved in data transfer

- DMA (Direct Memory Access)
  - CPU leaves data in memory
  - DMA hardware does data copy
Revisit: PIO Data Flow

\[\text{CPU} \rightarrow \text{Memory} \rightarrow \text{Disk} \]
Revisit: DMA Data Flow
while (STATUS == BUSY) //1
    wait for interrupt;
Initiate DMA transfer //2a
Wait for interrupt //2b
Write command to COMMAND register //3
while (STATUS == BUSY) //4
    wait for interrupt;
Hard Disk Drives (HDDs)
Basic Interface

- A magnetic disk has a **sector-addressable** address space
  - You can think of a disk as an array of sectors
  - Each sector (logical block) is the smallest unit of transfer

- Sectors are typically 512 or 4096 bytes

- Main operations
  - Read from sectors (blocks)
  - Write to sectors (blocks)
Disk Structure

- The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially
  - Sector 0 is the first sector of the first track on the outermost cylinder
  - Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost
  - Logical to physical address should be easy
    - Except for bad sectors
Internals of Hard Disk Drive (HDD)
Internals of Hard Disk Drive (HDD)

Platter
Covered with a magnetic film
Internals of Hard Disk Drive (HDD)

A single track example
Internals of Hard Disk Drive (HDD)

Spindle in the center of the surface
Internals of Hard Disk Drive (HDD)

The track is divided into numbered sectors
Internals of Hard Disk Drive (HDD)

A single track + an arm + a head
HDD Mechanism (3D view)

- track $t$
- sector $s$
- cylinder $c$
- platter
- rotation
- spindle
- arm assembly
- read-write head
- arm
Let’s Read Sector 0

Rotates this way
Let’s Read Sector 0

1. Seek for right track
2. Rotate (sector 9 → 0)
3. Transfer data (sector 0)
Don’t Try This at Home!

https://www.youtube.com/watch?v=9eMWG3fwiEU&feature=youtu.be&t=30s
Disk Performance

- I/O latency of disks
  \[ L_{I/O} = L_{\text{seek}} + L_{\text{rotate}} + L_{\text{transfer}} \]

- Disk access latency at \text{millisecond} level
Seek, Rotate, Transfer

- Seek may take several milliseconds (ms)
- Settling along can take 0.5 - 2ms
- Entire seek often takes 4 - 10ms
Seek, Rotate, Transfer

- Rotation per second (RPM)
  - 7200 RPM is common nowadays
  - 15000 RPM is high end
  - Old computers may have 5400 RPM disks

- $\frac{1}{7200} \text{ RPM} = \frac{1 \text{ minute}}{7200 \text{ rotations}} = \frac{1 \text{ second}}{120 \text{ rotations}} = 8.3 \text{ ms} / \text{ rotation}$
Seek, **Rotate**, Transfer

- **Rotation per second (RPM)**
  - 7200 RPM is common nowadays
  - 15000 RPM is high end
  - Old computers may have 5400 RPM disks

- \[ \frac{1}{7200 \text{ RPM}} = \frac{1 \text{ minute}}{7200 \text{ rotations}} = \frac{1 \text{ second}}{120 \text{ rotations}} = 8.3 \text{ ms} / \text{rotation} \]

- So it may take 4.2 ms on average to rotate to target (0.5 * 8.3 ms)
Seek, Rotate, **Transfer**

- Relatively fast
  - Depends on RPM and sector density

- 100+ MB/s is typical for SATA I (1.5Gb/s max)
  - Up to **600MB/s** for SATA III (6.0Gb/s)

- 1s / 100MB = 10ms / MB = 4.9us/sector
  - Assuming 512-byte sector
Workloads

- Seeks and rotations are slow while transfer is relatively fast

- What kind of workload is best suited for disks?
Workloads

- Seeks and rotations are slow while transfer is relatively fast

- What kind of workload is best suited for disks?
  - **Sequential I/O**: access sectors in order (transfer dominated)

- **Random** workloads access sectors in a random order (seek+rotation dominated)
  - Typically slow on disks
  - Never do random I/O unless you must! E.g., Quicksort is a terrible algorithm for disk!
Disk Performance Calculation

- Seagate Enterprise SATA III HDD

<table>
<thead>
<tr>
<th>Metric</th>
<th>Perf</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>7200</td>
</tr>
<tr>
<td>Avg seek</td>
<td>4.16ms</td>
</tr>
<tr>
<td>Max transfer</td>
<td>500MB/s</td>
</tr>
</tbody>
</table>

- How long does an average 4KB read take?
Disk Performance Calculation

- Seagate Enterprise SATA III HDD

<table>
<thead>
<tr>
<th>Metric</th>
<th>Perf</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>7200</td>
</tr>
<tr>
<td>Avg seek</td>
<td>4.16ms</td>
</tr>
<tr>
<td>Max transfer</td>
<td>500MB/s</td>
</tr>
</tbody>
</table>

- How long does an average 4KB read take?

\[
\text{transfer} = \frac{1 \, \text{sec}}{500 \, \text{MB}} \times 4 \, \text{KB} \times \frac{1,000,000 \, \text{us}}{1 \, \text{sec}} = 8 \, \text{us}
\]
Disk Performance Calculation

- Seagate Enterprise SATA III HDD

<table>
<thead>
<tr>
<th>Metric</th>
<th>Perf</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>7200</td>
</tr>
<tr>
<td>Avg seek</td>
<td>4.16ms</td>
</tr>
<tr>
<td>Max transfer</td>
<td>500MB/s</td>
</tr>
</tbody>
</table>

- How long does an average 4KB read take?

\[
\text{transfer} = \frac{1\ sec}{500\ MB} \times 4\ KB \times \frac{1,000,000\ us}{1\ sec} = 8\ us
\]

Latency = 4.16 ms + 4.2 ms + 8 us = 8.368 ms
The First Commercial Disk Drive

- 1956 IBM RAMDAC computer
  - 5M (7-bit) characters
  - 50 x 24” platters
  - Access time <= 1 sec