PERSISTENCE: I/O DEVICES

Shivaram Venkataraman
CS 537, Spring 2020
CHECK: 1,2,3

1. Can you hear me?

2. Can you see the slides and the annotations?

3. Can you write a question on the chat?
ADMINISTRIVIA

Videos will be available

Discussion: Sync ✓
Async → Video posted → OH → Individually or Groups of two

Project 4a: Out today! Due April 2nd, 10pm

Grades: Project 3 grades out!
Midterm grades end of the week (hopefully)

Attendance: 24 hours after end of the class
2:15 pm on Wed (Tue class)
Fri (Thu class)

Check Piazza → PDF with grades
AGENDA / LEARNING OUTCOMES

How does the OS interact with I/O devices?

What are the components of a hard disk drive?
RECAP
OPERATING SYSTEMS: THREE EASY PIECES

Three conceptual pieces

1. Virtualization
   Make each application believe it has each resource to itself: CPU and Memory

2. Concurrency
   Provide mutual exclusion, ordering

3. Persistence
   Devices store data
What good is a computer without any I/O devices?
keyboard, display, disks
movie, pen!

We want:
- **H/W** that will let us plug in different devices
- **OS** that can interact with different combinations
HARDWARE SUPPORT FOR I/O

CPU ➔ Memory

Memory Bus (proprietary)

General I/O Bus (e.g., PCI)

Graphics ➔ Peripheral I/O Bus (e.g., SCSI, SATA, USB)

4-core
60 GB/sec

PCI bus
8-10 GB/s

Farther
Hierarchy of devices connected
Slower

100-500 MB/s
CANONICAL DEVICE

OS reads writes to these

Device Registers

Status COMMAND DATA

Extra RAM Firmware Other special chips

Hidden registers

Hard disks Layout
Example Write Protocol

while (STATUS == BUSY)
    ; // spin
Write data to DATA register
Write command to COMMAND register
while (STATUS == BUSY)
    ; // spin

Microcontroller (CPU+RAM)
Extra RAM
Other special-purpose chips
while (STATUS == BUSY)  // 1
;
Write data to DATA register  // 2
Write command to COMMAND register  // 3
while (STATUS == BUSY)  // 4
;

CPU:  

Disk:  

Process A  

0. Shaded regions are wasteful as no useful work is done.
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!
interrupts vs. polling

Are interrupts always better than polling?

→ | μs

Fast device: Better to spin than take interrupt overhead
  → Device time unknown? Hybrid approach (spin then use interrupts)

Flood of interrupts arrive
  → Can lead to livelock (always handling interrupts)
  → Better to ignore interrupts while make some progress handling them

Other improvement
  → Interrupt coalescing (batch together several interrupts)
# Protocol Variants

Status checks: polling vs. interrupts

<table>
<thead>
<tr>
<th>Status</th>
<th>COMMAND</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller (CPU+RAM)</td>
<td>Extra RAM</td>
<td>Other special-purpose chips</td>
</tr>
</tbody>
</table>

- Status checks: polling vs. interrupts
**PROGRAMMED I/O VS. DIRECT MEMORY ACCESS**

**PIO (Programmed I/O):**
- CPU directly tells device what the data is

**DMA (Direct Memory Access):**
- CPU leaves data in memory
- Device reads data directly from memory
CPU can now do useful work while copy is in process.
while (STATUS == BUSY) // 1 
 ;
Write data to DATA register // 2
Write command to COMMAND register // 3
while (STATUS == BUSY) // 4 
 ;
## Protocol Variants

### Status Checks: polling vs. interrupts

- PIO vs DMA

### Table:

<table>
<thead>
<tr>
<th>Status</th>
<th>COMMAND</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller (CPU+RAM)</td>
<td>Extra RAM</td>
<td>Other special-purpose chips</td>
</tr>
</tbody>
</table>
while (STATUS == BUSY) // 1
{
    Write data to DATA register // 2
    Write command to COMMAND register // 3
    while (STATUS == BUSY) // 4
    {
    }
SPECIAL INSTRUCTIONS VS. MEM-MAPPED I/O

**Special instructions**
- each device has a port
- in/out instructions (x86) communicate with device

**Memory-Mapped I/O**
- H/W maps registers into address space
- loads/stores sent to device

Only OS can access

Doesn’t matter much (both are used)
# Protocol Variants

**Status checks**: polling vs. interrupts

- **PIO** vs **DMA**

- **Special instructions** vs. **Memory mapped I/O**

<table>
<thead>
<tr>
<th>Status</th>
<th>COMMAND</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller (CPU+RAM)</td>
<td>Extra RAM</td>
<td>Other special-purpose chips</td>
</tr>
</tbody>
</table>

1. Status of a device
2. Copy data
3. Copy command
4. Wait for device

- Port typical ranges of memory used for memory mapped devices
- Extra RAM - other special-purpose chips

Status checks: polling vs. interrupts

PIO vs DMA → 2

Special instructions vs. Memory mapped I/O → 3
DEVICE DRIVERS

Application

- POSIX API [open, read, write, close, etc.]
  - File System
  - Raw

Generic Block Interface [block read/write]

Generic Block Layer

Specific Block Interface [protocol-specific read/write]

Device Driver [SCSI, ATA, etc.]

Protocol for talking with devices

Ext 4
NTFS
FAT 32
VARIETY IS A CHALLENGE

Problem:

- many, many devices
- each has its own protocol

How can we avoid writing a slightly different OS for each H/W combination?

Write device driver for each device

Drivers are 70% of Linux source code

Installed a device driver

Modular / Pluggable

Don't typically use all!
If you have a fast non-volatile memory based storage device, which approach would work better?

- Polling avoids context switch

What part of a device protocol is improved by using DMA?

- Writing data to the device
HARD DISKS
HARD DISK INTERFACE

Disk has a sector-addressable address space
Appears as an array of sectors

Sectors are typically **512 bytes**

Main operations: reads + writes to sectors

Mechanical and slow (?)
Platter

media that can persist data
Both sides are used.

Spindle

Surface

Surface
Motor connected to spindle spins platters.

Rate of rotation: RPM

10,000 RPM → single rotation is 6 ms

\[
\frac{10,000 \text{ rotations}}{60 \text{ s}} = \frac{60,000 \text{ ms}}{1 \text{ rotation}}
\]
Surface is divided into rings: **tracks**

Stack of tracks (across platters): **cylinder**
Tracks are divided into numbered sectors

Project 4a released today
Survey: how did today's class go?

8 sectors in a track
Heads on a moving arm can read from each surface.
READING DATA FROM DISK

Rotational delay
READING DATA FROM DISK

Rotates this way

Seek Time

Remaining rotation

Seek

Spindle
TIME TO READ/WRITE

Three components:
Time = seek + rotation + transfer time
Seek, Rotate, Transfer

Seek cost: Function of cylinder distance
- Not purely linear cost
- Must accelerate, coast, decelerate, settle
- Settling alone can take 0.5 - 2 ms

Entire seeks often takes 4 - 10 ms
Average seek = 1/3 of max seek

Depends on rotations per minute (RPM)
- 7200 RPM is common, 15000 RPM is high end

Average rotation?

Pretty fast: depends on RPM and sector density.

100+ MB/s is typical for maximum transfer rate
What is the time for 4KB random read?

<table>
<thead>
<tr>
<th></th>
<th>Cheetah 15K.5</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>300 GB</td>
<td>1 TB</td>
</tr>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Average Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
<tr>
<td>Platters</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Cache</td>
<td>16 MB</td>
<td>16/32 MB</td>
</tr>
<tr>
<td>Connects via</td>
<td>SCSI</td>
<td>SATA</td>
</tr>
</tbody>
</table>
NEXT STEPS

Advanced disk features
Scheduling disk requests

Project 4a: Out tonight
Grades: Project 2b, 3, midterm by tomorrow!