PERSISTENCE: I/O DEVICES

Shivaram Venkataraman
CS 537, Spring 2019
Project 4a: Out tonight, due on April 4th
Work in groups of up to two

Grades: Project 2b, 3, midterm by tomorrow!
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

2. Concurrency

3. Persistence
VIRTUALIZATION

Make each application believe it has each resource to itself (CPU and Memory).

Abstraction: Process API, Address spaces

Mechanism:
- Limited direct execution, CPU scheduling
- Address translation (segmentation, paging, TLB)

Policy: MLFQ, LRU etc.
CONCURRENCY

Events occur simultaneously and may interact with one another

Need to
  Hide concurrency from independent processes
  Manage concurrency with interacting processes

Provide abstractions (locks, semaphores, condition variables etc.)

Correctness: mutual exclusion, ordering

Performance: scaling data structures, fairness

Common Bugs!
OPERATING SYSTEMS: THREE EASY PIECES

Three conceptual pieces

1. Virtualization

2. Concurrency

3. Persistence
What good is a computer without any I/O devices?

keyboard, display, disks

We want:

- **H/W** that will let us plug in different devices
- **OS** that can interact with different combinations
Why use hierarchical buses?

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)

Peripheral Component

Interconnect

Data drives, USB component = 100s MB/s

60 GB/s on a 4-core system
**EXAMPLE WRITE PROTOCOL**

```
while (STATUS == BUSY)
    ; // spin

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

CPU:

Disk:
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;
Are interrupts always better than polling?

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

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

Status checks: polling vs. interrupts
DATA TRANSFER COSTS

CPU: 1 1 1 1 1 1 2 2 2 2 2 2 1 1

Disk: 1 1 1 1 1

PID 1

Copy data
CPU to disk
PID = 2 (interrupt)

CPU is not used effectively

Disk
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
while (STATUS == BUSY)  // 1 
;
Write data to DATA register  // 2
Write command to COMMAND register  // 3
while (STATUS == BUSY)  // 4 
;
PROTOCOL VARIANTS

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

Status checks: polling vs. interrupts

PIO vs DMA
while (STATUS == BUSY) // 1
;
Write data to DATA register // 2
Write command to COMMAND register // 3
while (STATUS == BUSY) // 4
;
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

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>

- Remove polling
- DMA
- In/out memory mapped
Application

POSIX API [open, read, write, close, etc.]

File System

Generic Block Interface [block read/write]

Raw

Specific Block Interface [protocol-specific read/write]

Device Driver [SCSI, ATA, etc.]

Protocol

Device Drivers

SSD

USB

SATA
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
BUNNY 10

https://tinyurl.com/cs537-sp19-bunny10
If you have a fast non-volatile memory based storage device, which approach would work better?

Polling.

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

Data register.
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
Motor connected to spindle spins platters

Rate of rotation: RPM

10000 RPM → single rotation is 6 ms

\[
\frac{10000}{60} \text{ rotations per min} \times \frac{60}{10000} \text{ rotations per sec} = 6 \text{ ms}
\]
Surface is divided into rings: tracks

Stack of tracks (across platters): cylinder
Tracks are divided into numbered sectors
Heads on a moving arm can read from each surface.
READING DATA FROM DISK

Rotational delay

Average wait for half a rotation before sector reaches head

12 sectors in track
READING DATA FROM DISK

Rotates this way

Seek Time

- Move the disk head right to track
TIME TO READ/WRITE

Three components:
Time = seek + rotation + transfer time

Diagram:
- Seek time
- Rotation time
- Bus bandwidth
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 take 4 - 10 ms
Average seek = 1/3 of max seek

Seek, Rotate, Transfer

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
BUNNY 11

https://tinyurl.com/cs537-sp19-bunny11
What is the time for 4KB random read?

\[ T_{\text{total}} = T_{\text{seek}} + T_{\text{rotation}} + T_{\text{transfer}} \]

For Cheetah:
- \( T_{\text{seek}} = 4 \text{ ms} \)
- \( T_{\text{rotation}} = 2 \text{ ms} \)

For Transfer:
\[ T_{\text{transfer}} = \frac{4 \text{ KB}}{125 \text{ MB/s}} = \frac{4 \times 10^3 \text{ B}}{125 \times 10^6 \text{ B/s}} \approx 30 \text{ ms} \]
NEXT STEPS

Advanced disk features
Scheduling disk requests

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