CS 162: Operating Systems and Systems Programming

Lecture 12: I/O and Storage Devices

July 15, 2019
Instructor: Jack Kolb
https://cs162.eecs.berkeley.edu
Logistics

• Midterm Exam: 7/18, 5-7pm
  • No Lecture that Day
  • Review Session during tomorrow’s normal lecture slot

• Homework 2 and Project 2 Released
Recall: RPC

• Idea: Make communication look like an ordinary function call

• Wrapper library like for system calls
  • Called stubs

• Also wrappers at the receiving end
  • Read messages from socket, dispatch to actual function

• Look like local function calls
Recall: RPC Information Flow

Machine A

Client (caller)

Server (callee)

Network

Machine B

Client Stub

Packet Handle

Packet Handler

Server Stub

Packet Handle

bundle args

send

receive

unbundle ret vals

unbundle args

call

return

bundle args

send

receive

return

call

unbundle ret vals

Network
Recall: HTTP

• Application protocol for The Web
  • Retrieve a specific object, upload data, etc.

• Runs on top of TCP (sockets)

• Like any protocol, stipulates:
  • **Syntax**: Content sent over socket connection
  • **Semantics**: Meaning of a message
  • Valid replies and actions taken upon message receipt

• Arguably a form of RPC
Recall: DNS

- Distributed system

**Purpose:** Convert a human readable name (www.google.com) to an IP Address (172.217.6.78)

- Why?
  - Humans don't want to remember IP addresses
  - But IP routes traffic based on IP addresses

- Other benefits
  - Service can change hosts (IP Address) but keep name
  - Fancy things like sending Google users to different hosts
Recall: Life Cycle of An I/O Request

User Program
- request I/O
  - system call
    - can already satisfy request?
      - yes
      - user process
        - I/O completed, input data available, or output completed
        - return from system call
      - no
      - send request to device driver, block process if appropriate
        - kernel I/O subsystem
          - transfer data (if appropriate) to process, return completion or error code

Kernel I/O Subsystem
- process request, issue commands to controller, configure controller to block until interrupted
  - device driver
    - determine which I/O completed, indicate state change to I/O subsystem
      - receive interrupt, store data in device-driver buffer
        - if input, signal to unblock device driver

Device Driver Top Half
- device-controller commands
  - interrupt handler
    - I/O completed, generate interrupt

Device Driver Bottom Half
- monitor device, interrupt when I/O completed
  - device controller
    - time

Device Hardware
I/O In a Picture

Processor
- Core
  - Registers
  - LI Cache
  - L2 Cache
- Core
  - Registers
  - LI Cache
  - L2 Cache
  - L3 Cache (shared)

I/O Controllers
- Read / Write
- DMA transfer
- Wires
- Interrupts

Main Memory (DRAM)
- Secondary Storage (SSD)
- Secondary Storage (Disk)
I/O In a Picture

- **I/O Controllers** – hardware to support communication between processor and HW
- Processor accesses controller’s registers as if memory
  - Using memory bus or (sometimes) separate bus
- Controllers signal processor through **interrupts**
I/O In a Picture

- Controllers communicate results through *registers* or direct memory access to main memory.

- DMA transfer

- Wire connections

- Interrupts
I/O Requirements

• Without I/O, computers are useless!
• Thousands of devices, each slightly different
• Devices unreliable: media failures and transmission errors
• Devices unpredictable and/or slow
Operational Parameters for I/O

• Data Granularity: **byte** (e.g., keyboard) versus **block** (e.g., disks and network)

• Access pattern: **sequential** (e.g., tape) versus **random** (e.g., hard drive)

• Transfer Notification: **Polling** vs. **Interrupts**

• Transfer Mechanism: **Programmed I/O** versus **direct memory access**
Kernel Device Structure

The System Call Interface

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

- Architecture Dependent Code
- Virtual memory
- Files and dirs: the VFS
- Connectivity
- TTYs and device access

- File System Types

Sometimes a lot of (architecture-neutral) layers in front of actual device operations
Kernel Device Structure

The System Call Interface

- Process Management
- Memory Management
- Filesystems
- Device Control
- Networking
- Architecture Dependent Code
- Memory Manager
- Block Devices
- TTYs and device access
- Connectivity
- Network Subsystem
- IF drivers

Concurrency, multitasking

Other times just providing relatively direct access to devices
Sometimes primary “users” are parts of the OS – not user programs – but similar interface
Goal of the I/O Subsystem

- Provide **uniform interfaces** ("everything is a file")
- The following code works for many devices:
  ```c
  FILE *fh = fopen("/dev/something", "w");
  for (int i = 0; i < 10; i++) {
    fprintf(fd, "Count %d\n", i);
  }
  fclose(fh);
  ```
- Code that actually controls devices (**device drivers**) provides a standard interface to kernel I/O subsystem
- Kernel handles I/O syscalls by dispatching to proper driver
Options for User I/O Timing

• So far we’ve said read, write blocks calling thread

• Might want other choices:
  • Handle multiple I/O devices in one thread
  • Read data as it is available
  • Queue lots of work at once without using lots of buffering in the kernel
Non-Block I/O “Don’t Wait”

- **read**: Just return whatever data is available
- **write**: Just write whatever the kernel can buffer in its memory for now
- So read/write calls *may not read or write anything*
- Makes sense for network/terminal/etc.
- Questionable for regular files on disk
  - How much space to allocate for kernel buffers?
  - What starts a read or write?

- On POSIX: `fcntl(fd, F_SETFL, flags | O_NONBLOCK)`
I/O Multiplexing: select/poll

- POSIX way to wait for one of several files to have data available
- Supports event-loop style programming
- Indicates which file descriptors can read or write data without the need to block
- Mix with non-blocking I/O
Asynchronous I/O – “Tell Me Later”

• Invoke callback function with I/O op completes
• User makes a call to `start` I/O
  • Specifies buffer in userspace – source of a write or destination for a read
  • Program can run on CPU while I/O happens (once it is started)
• Notification might be signal, system call to `poll`, …
Types of Devices

• **Block Devices**: Hard Drives, Tape Drives, etc.
  • Access data in blocks (e.g. 4KB)
  • `open()`, `read()`, `write()`, `seek()`
  • Raw I/O or file-system access

• **Character Devices**: Keyboards, mice, serial ports, some USB devices
  • Read/write one character at a time
  • Commands include `get()`, `put()`
  • Libraries layered on top to allow line editing
Types of Devices

• **Network Devices**: Ethernet, Wireless, Bluetooth
  • Distinct enough from block/character devices to have its own interface
  • Unix and Windows include socket interface
    • Separates network protocol from network operation
    • Includes `select` (multiplexing) functionality
  • Usage: pipes, FIFOs, streams, queues, …
Chip-scale Features of 2015 x86

• Significant pieces:
  • Four CPU cores
  • Integrated GPU
  • System Agent (Memory and Fast I/O)
  • Shared L3 cache

• Integrated I/O
  • Integrated memory controller (IMC)
    • Two independent channels of DRAM
  • High-speed PCI-Express (for GPUs)
  • 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
  - Audio
  - BIOS support
  - More PCI Express
  - SATA (for Disks)

Sky Lake System Configuration
Modern I/O Systems
Flashback: Range of Timescales

L1 cache reference: 0.5 ns
Branch mispredict: 5 ns
L2 cache reference: 7 ns
Mutex lock/unlock: 25 ns
Main memory reference: 100 ns
Compress 1K bytes with Zippy: 3,000 ns
Send 2K bytes over 1 Gbps network: 20,000 ns
Read 1 MB sequentially from memory: 250,000 ns
Round trip within same datacenter: 500,000 ns
Disk seek: 10,000,000 ns
Read 1 MB sequentially from disk: 20,000,000 ns
Send packet CA->Netherlands->CA: 150,000,000 ns

Jeff Dean: "Numbers Everyone Should Know"
Challenge: Range of Device Data Transfer Rates

- Spans over 12 orders of magnitude
- OS Goal: Low overhead for fast and slow devices
How does the processor 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
How does the processor talk to the device?

- Processor accesses controller’s registers in two ways
  1. I/O-specific instructions: in/out
     1. Intel: out 0x21, AL
  2. Memory/mapped I/O: load or store
     - Registers appear in physical address space
Ex: Memory-Mapped Display Controller

- Hardware maps control registers and display memory into physical address space
- Addresses set at boot time (on modern OSs)
- Simply writing to display memory ("frame buffer") changes image on screen
  - Addr: 0x8000F000 — 0x8000FFFF
- Writing graphics description to cmd queue
  - Set of triangles describing some scene
  - Addr: 0x80010000 — 0x8001FFFF
- Writing to the command register may cause on-board graphics hardware to do something
  - Process current command queue
  - Addr: 0x0007F004
- Can protect with address translation
Transferring Data to/from Controller

- **Programmed I/O**: Processor reads/writes data from/to device registers
  - Pro: Simple hardware (control and data interfaces are the same)
  - Con: I/O can consume a lot of CPU time

- **Direct Memory Access (DMA)**: I/O controller reads/writes from/to RAM without CPU
  - OS specifies physical address range to use via device controller registers
  - Pro: CPU can now do other things during large I/O ops
Transferring Data: DMA

1. device driver is told to transfer disk data to buffer at address X
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X
3. CPU
4. disk controller sends each byte to DMA controller
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
6. when C = 0, DMA interrupts CPU to signal transfer completion
Transferring Data: DMA

1. Device driver is told to transfer disk data to buffer at address X
2. Device driver tells disk controller to transfer C bytes from disk to buffer at address X
3. Disk controller initiates DMA transfer
4. Disk controller sends each byte to DMA controller
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
6. When C = 0, DMA interrupts CPU to signal transfer completion

IDE disk controller

PCI bus

CPU memory bus

CPU

Cache

Memory

Buffer
I/O Devices: Notifying the OS

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

• Two options:
  • Interrupts
  • Polling
I/O Devices: Notifying the OS

- **Interrupt:** Device generates an interrupt whenever it needs CPU’s attention
  - Pro: Handles unpredictable events well
  - Con: Handling each interrupt relatively high overhead

- **Polling:** OS periodically checks a device-specific status register
  - Pro: Low overhead to run this check
  - Con: May waste cycles checking for infrequent events
I/O Devices: Notifying the OS

• Actual devices combine both interrupts and polling

• Example: High-bandwidth network adapter
  • Interrupt for first new incoming packet
  • OS polls for any subsequent packets until hardware queues are emptied
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

- **Startup or Overhead**: Time to initiate an operation

- Most I/O operations are roughly linear in $n$ bytes
  - $\text{Latency}(n) = \text{Overhead} + \frac{n}{\text{TransferCapacity}}$
Example: Fast Network

• Consider a 1 Gb/s link ($B = 125$ MB/s)
  • With a startup cost $S = 1$ ms
Example: At 10ms startup (More Like a Disk)
What determines peak bandwidth for I/O?

• **Bus Speed**
  - PCI-X: 1064 MB/s = 133MHz x 64 bit (per lane)
  - Ultra Wide SCSI = 40 MB/s
  - USB 3.0 = 5 GB/s
  - Thunderbolt 3 = 40 Gb/s

• **Device Transfer Bandwidth**
  - Rotational speed of spinning metal hard drive
  - Write/Read rate of NAND flash
  - Signaling rate of network link
Break
Popular Storage Devices

**Magnetic Disks**
- Rarely becomes corrupted
- Traditionally: large capacity at low cost
- Block level random access
- Slow performance for random access
- Better performance for sequential access

**Flash Memory**
- Rarely becomes corrupted
- Increasingly larger and cheaper
- Block level random access
- Good performance for reads, worse for random writes
- Have to erase data in large blocks
- Challenge: Wear Levelling
The Emergence of SSDs

In 2007

Source: Storage Newsletter
The Emergence of SSDs

Today

Source: Storage Newsletter
The Emergence of SSDs

- Faster
- Lower power
- No moving parts

But HDDs have their place
- Cheapest online storage per byte
- Application: Archival storage
Hard Disk Drives (HDDs)

IBM Personal Computer (1986)
30 MB hard disk - $500
30-40ms seek time
0.7-1 MB/s (est.)

Western Digital Drive
http://www.storagereview.com/guide/

Read/Write Head
Side View
Magnetic Disk

- Unit of Transfer: Sector
  - Ring of sectors form a track
  - Stack of tracks form a cylinder
- Read/Write heads position on cylinders
- Impressive Engineering
  - Track is ~1 um wide
  - Data Density > 250GB/in²
  - Head sits 15nm above platter surface
Reading and Writing

1. **Seek Time**: Move head to correct cylinder
   - Average of 5-10 ms
   - Faster if reads adjacent

2. **Rotational Latency**: Wait for right sector within cylinder to come under head
   - 4-8ms (3600-7200 RPM) 2-4ms (15000 RPM)
   - Faster if reads adjacent

3. **Transfer Time**: Time to actually read sectors
   - 50-100 MB/sec.
Latency = Queue Time + Controller Time + Seek Time + Rotational Latency + Transfer Time

To Achieve Best Bandwidth: Large Transfers of Physically Adjacent Sectors from one track
HDD Controllers

• Old Days: Device driver would address block of data by cylinder number, head (platter) number, and sector number

• Now: Hard drive is just an array of sectors
  • Sector number mapped internally to physical location
  • Numerically close sectors are probably physically close

• Lots of other intelligent features
  • Error Correcting Codes
  • Sector Sparing: Remap sector nums to avoid faulty regions of physical disk
Summary

• Diversity of I/O Devices
  • Many different speeds (0.1 bytes/sec. to GB/sec.)
  • Different access patterns (block, character, network)

• I/O Controllers: HW that controls actual device
  • Process access through specific instructions or load/store to specially mapped physical addresses
  • Report results either via polling or via interrupts

• Hard Disk Drives
  • Faster for sequential access than random access
  • Modern OS only sees sector numbers