## Midterm results

- **Mean:** 41.7, **median:** 38

## Administrivia

- Recall we will have a resurrection final
  - As long as you took the midterm
  - Don’t panic if you didn’t do well on midterm
  - But make sure you understand all the answers
  - There may be questions on same topics on the final

- **Final grade based on rank and thresholds**
  - Rank based on Projects + max(Final, (Midterm + Final)/2) (Assuming you took the midterm)

## Outline

1. PC system architecture
2. Driver architecture
3. Disks
4. Disk scheduling
5. Flash

## Old school memory and I/O buses

- CPU accesses physical memory over a bus
- Devices access memory over I/O bus with DMA
- Devices can appear to be a region of memory

## Realistic ~2005 PC architecture

- Advanced Programmable Interrupt Controller
- Memory
- North Bridge
- Main memory
- South Bridge
- ISA bus
- USB
- PCI bus
- PCI IRQs
- Advanced Programmable Interrupt Controller bus
- CPU
- CPU

- 1880Mbps
- 1056Mbps
- I/O bus
Modern PC architecture (intel)

CPU now entirely subsumes IOH [intel]

What is memory?

- **SRAM – Static RAM**
  - Like two NOT gates circularly wired input-to-output
  - 4–6 transistors per bit, actively holds its value
  - Very fast, used to cache slower memory

- **DRAM – Dynamic RAM**
  - A capacitor + gate, holds charge to indicate bit value
  - 1 transistor per bit – extremely dense storage
  - Charge leaks – need slow comparator to decide if bit 1 or 0
  - Must re-write charge after reading, and periodically refresh

- **VRAM – “Video RAM”**
  - Dual ported DRAM, can write while someone else reads

What is I/O bus? E.g., PCI

Outline

1. PC system architecture
2. Driver architecture
3. Disks
4. Disk scheduling
5. Flash

Communicating with a device

- **Memory-mapped device registers**
  - Certain physical addresses correspond to device registers
  - Load/store gets status/sends instructions – not real memory

- **Device memory – device may have memory OS can write to directly on other side of I/O bus**

- **Special I/O instructions**
  - Some CPUs (e.g., x86) have special I/O instructions
  - Like load & store, but asserts special I/O pin on CPU
  - OS can allow user-mode access to I/O ports at byte granularity

- **DMA – place instructions to card in main memory**
  - Typically then need to “poke” card by writing to register
  - Overlaps unrelated computation with moving data over (typically slower than memory) I/O bus
x86 I/O instructions

static inline void IDEWait()
{
    asm volatile ("inb %w1, %b0" : "a" (data) : "Nd" (port));
}

IDE disk driver

void IDE_ReadSector(int disk, int off, void *buf)
{
    outb(0x1F6, disk == 0 ? 0xE0 : 0xF0); // Select Drive
    IDEWait();
    outb(0x1F2, 1); // Read length (1 sector = 512 B)
    outb(0x1F4, off >> 8); // LBA mid
    outb(0x1F3, off); // LBA low
    insw(0x1F0, buf, 256); // Read 256 words
}

Memory-mapped IO

- **in/out instructions slow and clunky**
  - Instruction format restricts what registers you can use
  - Only allows 2^16 different port numbers
  - Per-port access control turns out not to be useful (any port access allows you to disable all interrupts)

- **Devices can achieve same effect with physical addresses, e.g.**:
  
  volatile int32_t *device_control
    = (int32_t *)(0xc0100 + PHYS_BASE);
  *device_control = 0x80;
  int32_t status = *device_control;
  
  - OS must map physical to virtual addresses, ensure non-cachable

- **Assign physical addresses at boot to avoid conflicts. PCI:**
  - Slow/clunky way to access configuration registers on device
  - Use that to assign ranges of physical addresses to device

IDEE: parallel port (LPT1)

- **Simple hardware has three control registers:**

- Memory buffers

- **Idea:** only use CPU to transfer control requests, not data

- Include list of buffer locations in main memory
  - Device reads list and accesses buffers through DMA
  - Descriptions sometimes allow for scatter/gather I/O
Example: Network Interface Card

- Link interface talks to wire/fiber/antenna
  - Typically does framing, link-layer CRC
- FIFOs on card provide small amount of buffering
- Bus interface logic uses DMA to move packets to and from buffers in main memory

Driver architecture

- Device driver provides several entry points to kernel
  - Reset, ioctl, output, interrupt, read, write, strategy …
- How should driver synchronize with card?
  - E.g., Need to know when transmit buffers free or packets arrive
  - Need to know when disk request complete
- One approach: Polling
  - Sent a packet? Loop asking card when buffer is free
  - Waiting to receive? Keep asking card if it has packet
  - Disk I/O? Keep looping until disk ready bit set
- Disadvantages of polling?

Interrupt driven devices

- Instead, ask card to interrupt CPU on events
  - Interrupt handler runs at high priority
  - Asks card what happened (xmit buffer free, new packet)
  - This is what most general-purpose OSes do
- Bad under high network packet arrival rate
  - Packets can arrive faster than OS can process them
  - Interrupts are expensive
  - Interrupt handlers have high priority
  - In worst case, can spend 100% of time in interrupt handler and never make any progress – receive livelock
  - Best: Adaptive switching between interrupts and polling
- Very good for disk requests
- Rest of today: Disks (network devices in 3 lectures)

Outline

1. PC system architecture
2. Driver architecture
3. Disks
4. Disk scheduling
5. Flash
Anatomy of a disk [Ruemmler]

- Stack of magnetic platters
  - Rotate together on a central spindle @3,600-15,000 RPM
  - Drive speed drifts slowly over time
  - Can't predict rotational position after 100-200 revolutions

- Disk arm assembly
  - Arms rotate around pivot, all move together
  - Pivot offers some resistance to linear shocks
  - One disk head per recording surface (2× platters)
  - Sensitive to motion and vibration [Gregg] (demo on youtube)

Storage on a magnetic platter

- Platters divided into concentric tracks
- A stack of tracks of fixed radius is a cylinder
- Heads record and sense data along cylinders
  - Significant fractions of encoded stream for error correction

- Generally only one head active at a time
  - Disks usually have one set of read-write circuitry
  - Must worry about cross-talk between channels
  - Hard to keep multiple heads exactly aligned

Cylinders, tracks, & sectors
**Disk positioning system**

- Move head to specific track and keep it there
  - Resist physical shocks, imperfect tracks, etc.
- **A seek consists of up to four phases:**
  - **speedup**—accelerate arm to max speed or half way point
  - **coast**—at max speed (for long seeks)
  - **slowdown**—stops arm near destination
  - **settle**—adjusts head to actual desired track
- Very short seeks dominated by settle time (~1 ms)
- Short (200–400 cyl.) seeks dominated by speedup
  - Accelerations of 40g

**Seek details**

- Head switches comparable to short seeks
  - May also require head adjustment
  - Settles take longer for writes than for reads – Why?
- Disk keeps table of pivot motor power
  - Maps seek distance to power and time
  - Disk interpolates over entries in table
  - Table set by periodic “thermal recalibration”
  - But, e.g., ~500 ms recalibration every ~25 min bad for AV
- “Average seek time” quoted can be many things
  - Time to seek 1/3 disk, 1/3 time to seek whole disk

**Sectors**

- Disk interface presents linear array of sectors
  - Historically 512 B, but 4 KiB in “advanced format” disks
  - Written atomically (even if there is a power failure)
- Disk maps logical sector #s to physical sectors
  - **Zoning**—puts more sectors on longer tracks
  - **Track skewing**—sector 0 pos. varies by track (sequential access speed)
  - **Sparing**—flawed sectors remapped elsewhere
- OS doesn’t know logical to physical sector mapping
  - Larger logical sector # difference means longer seek time
  - Highly non-linear relationship (and depends on zone)
  - OS has no info on rotational positions
  - Can empirically build table to estimate times

**Disk interface**

- Controls hardware, mediates access
- Computer, disk often connected by bus (e.g., ATA, SCSI, SATA)
  - Multiple devices may contend for bus
- **Possible disk/interface features:**
  - Disconnect from bus during requests
  - Command queuing: Give disk multiple requests
    - Disk can schedule them using rotational information
  - Disk cache used for read-ahead
    - Otherwise, sequential reads would incur whole revolution
    - Cross track boundaries? Can’t stop a head-switch
- Some disks support write caching
  - But data not stable—not suitable for all requests
**SCSI overview [Schmidt]**

- **SCSI domain consists of devices and an SDS**
  - Devices: host adapters & SCSI controllers
  - Service Delivery Subsystem connects devices—e.g., SCSI bus
- **SCSI-2 bus (SDS) connects up to 8 devices**
  - Controllers can have >1 “logical units” (LUNs)
  - Typically, controller built into disk and 1 LUN/target, but “bridge controllers” can manage multiple physical devices
- **Each device can assume role of initiator or target**
  - Traditionally, host adapter was initiator, controller target
  - Now controllers act as initiators (e.g., COPY command)
  - Typical domain has 1 initiator, ≥1 targets

**SCSI requests**

- A request is a command from initiator to target
  - Once transmitted, target has control of bus
  - Target may disconnect from bus and later reconnect (very important for multiple targets or even multitasking)
- **Commands contain the following:**
  - Task identifier—initiator ID, target ID, LUN, tag
  - Command descriptor block—e.g., read 10 blocks at pos. N
  - Optional task attribute—SIMPLE, ORDERD, HEAD OF QUEUE
  - Optional: output/input buffer, sense data
  - Status byte—GOOD, CHECK CONDITION, INTERMEDIATE, . . .

**Executing SCSI commands**

- Each LUN maintains a queue of tasks
  - Each task is DORMANT, BLOCKED, ENABLED, OF ENDED
  - SIMPLE tasks are dormant until no ordered/head of queue
  - ORDERED tasks dormant until no HoQ/more recent ordered
  - HoQ tasks begin in enabled state
- Task management commands available to initiator
  - Abort/terminate task, Reset target, etc.
- **Linked commands**
  - Initiator can link commands, so no intervening tasks
  - E.g., could use to implement atomic read-modify-write
  - Intermediate commands return status byte INTERMEDIATE

**SCSI exceptions and errors**

- After error stop executing most SCSI commands
  - Target returns with CHECK CONDITION status
  - Initiator will eventually notice error
  - Must read specifics w. REQUEST SENSE
- Prevents unwanted commands from executing
  - E.g., initiator may not want to execute 2nd write if 1st fails
- **Simplifies device implementation**
  - Don’t need to remember more than one error condition
- **Same mechanism used to notify of media changes**
  - I.e., ejected tape, changed CD-ROM

**Disk performance**

- Placement & ordering of requests a huge issue
  - Sequential I/O much, much faster than random
  - Long seeks much slower than short ones
  - Power might fail any time, leaving inconsistent state
- **Must be careful about order for crashes**
  - More on this in next two lectures
- **Try to achieve contiguous accesses where possible**
  - E.g., make big chunks of individual files contiguous
- **Try to order requests to minimize seek times**
  - OS can only do this if it has a multiple requests to order
  - Requires disk I/O concurrency
  - High-performance apps try to maximize I/O concurrency
- Next: How to schedule concurrent requests

**Outline**

1. PC system architecture
2. Driver architecture
3. Disks
4. Disk scheduling
5. Flash
Scheduling: FCFS

- “First Come First Served”
  - Process disk requests in the order they are received

Advantages
- Easy to implement
- Good fairness

Disadvantages
- Cannot exploit request locality
- Increases average latency, decreasing throughput

FCFS example

```
queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
```

Shortest positioning time first (SPTF)

- Shortest positioning time first (SPTF)
  - Always pick request with shortest seek time
- Also called Shortest Seek Time First (SSTF)

Advantages
- Exploits locality of disk requests
- Higher throughput

Disadvantages
- Starvation
- Don’t always know what request will be fastest

Improvement?

\[ T_{\text{eff}} = T_{\text{pos}} - W \cdot T_{\text{wait}} \]
**SPTF example**

```
queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
```

**“Elevator” scheduling (SCAN)**

- Sweep across disk, servicing all requests passed
  - Like SPTF, but next seek must be in same direction
  - Switch directions only if no further requests
- Advantages
  - Takes advantage of locality
  - Bounded waiting
- Disadvantages
  - Cylinders in the middle get better service
  - Might miss locality SPTF could exploit
- CSCAN: Only sweep in one direction
  - Very commonly used algorithm in Unix
  - Also called LOOK/CLOOK in textbook
    - (Textbook uses [C]SCAN to mean scan entire disk uselessly)

**VSCAN(r)**

- Continuum between SPTF and SCAN
  - Like SPTF, but slightly changes “effective” positioning time
    - If request in same direction as previous seek: $T_{eff} = T_{pos}$
    - Otherwise: $T_{eff} = T_{pos} + r \cdot T_{max}$
  - when $r = 0$, get SPTF, when $r = 1$, get SCAN
  - E.g., $r = 0.2$ works well
- Advantages and disadvantages
  - Those of SPTF and SCAN, depending on how $r$ is set
- See [Worthington] for good description and evaluation of various disk scheduling algorithms

**Outline**

1. PC system architecture
2. Driver architecture
3. Disks
4. Disk scheduling
5. Flash
Flash memory

- Today, people increasingly using flash memory
- Completely solid state (no moving parts)
  - Remembers data by storing charge
  - Lower power consumption and heat
  - No mechanical seek times to worry about
- Limited # overwrites possible
  - Blocks wear out after 10,000 (MLC) – 100,000 (SLC) erases
  - Requires flash translation layer (FTL) to provide wear leveling, so repeated writes to logical block don’t wear out physical block
  - FTL can seriously impact performance
  - In particular, random writes very expensive [Birrell]
- Limited durability
  - Charge wears out over time
  - Turn off device for a year, you can potentially lose data

Types of flash memory

- NAND flash (most prevalent for storage)
  - Higher density (most used for storage)
  - Faster erase and write
  - More errors internally, so need error correction
- NOR flash
  - Faster reads in smaller data units
  - Can execute code straight out of NOR flash
  - Significantly slower erases
- Single-level cell (SLC) vs. Multi-level cell (MLC)
  - MLC encodes multiple bits in voltage level
  - MLC slower to write than SLC
  - MLC has lower durability (bits decay faster)

NAND Flash Overview

- Flash device has 2112-byte pages
  - 2048 bytes of data + 64 bytes metadata & ECC
- Blocks contain 64 (SLC) or 128 (MLC) pages
- Blocks divided into 2–4 planes
  - All planes contend for same package pins
  - But can access their blocks in parallel to overlap latencies
- Can read one page at a time
  - Takes 25 µsec + time to get data off chip
- Must erase whole block before programming
  - Erase sets all bits to 1—very expensive (2 msec)
  - Programming pre-erased block requires moving data to internal buffer, then 200 (SLC)–800 (MLC) µsec

Flash Characteristics [Caulfield’09]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>SLC</th>
<th>MLC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Density Per Die (GB)</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Page Size (Bytes)</td>
<td>2048+32</td>
<td>2048+64</td>
</tr>
<tr>
<td>Block Size (Pages)</td>
<td>64</td>
<td>128</td>
</tr>
<tr>
<td>Read Latency (µs)</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>Write Latency (µs)</td>
<td>200</td>
<td>800</td>
</tr>
<tr>
<td>Erase Latency (µs)</td>
<td>2000</td>
<td>2000</td>
</tr>
<tr>
<td>40MHz, 16-bit bus Read b/w (MB/s)</td>
<td>75.8</td>
<td>75.8</td>
</tr>
<tr>
<td>Program b/w (MB/s)</td>
<td>20.1</td>
<td>5.0</td>
</tr>
<tr>
<td>133MHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read b/w (MB/s)</td>
<td>126.4</td>
<td>126.4</td>
</tr>
<tr>
<td>Program b/w (MB/s)</td>
<td>20.1</td>
<td>5.0</td>
</tr>
</tbody>
</table>