CS 33

Signals part 3; Memory Hierarchy II
Interrupted System Calls

• What if a signal is handled before the system call completes?
  1) invoke handler, then resume system call
     • not clear if system call should be resumed or
  2) invoke handler, then return from system call prematurely
     • if one or more pieces were completed, return total number of bytes transferred
     • otherwise return “interrupted” error
Interrupted System Calls: Non-Lengthy Case

```c
while (read(fd, buffer, buf_size) == -1) {
    if (errno == EINTR) {
        /* interrupted system call — try again */
        continue;
    }
    /* the error is more serious */
    perror("big trouble");
    exit(1);
}
```
Quiz 1

```c
int ret;
char buf[128];

fillbuf(buf);

ret = write(1, buf, 128);
```

• The value of `ret` is:
  a) either -1 or 128
  b) either -1, 0, or 128
  c) any integer in the range [-1, 128]
Interrupted System Calls: Lengthy Case

```c
char buf[BSIZE];
fillbuf(buf);
long remaining = BSIZE;
char *bptr = buf;
for (; ; ) {
    long num_xfrd = write(fd, bptr, remaining);
    if (num_xfrd == -1) {
        if (errno == EINTR) {
            /* interrupted early */
            continue;
        }
        perror("big trouble");
        exit(1);
    }
    if (num_xfrd < remaining) {
        /* interrupted after the first step */
        remaining -= num_xfrd;
        bptr += num_xfrd;
        continue;
    }
    /* success! */
    break;
}
```
Asynchronous Signals (1)

```c
main( ) {
    void handler(int);
    signal(SIGINT, handler);

    /* long-running buggy code */
}

void handler(int sig) {
    /* clean up */
    exit(1);
}
```
Asynchronous Signals (2)

```c
computation_state_t state;

main() {
    void handler(int);

    signal(SIGINT, handler);
}

long_running_procedure() {
    while (a_long_time) {
        update_state(&state);
        compute_more();
    }
}

void handler(int sig) {
    display(&state);
}
```
Asynchronous Signals (3)

main() {
    void handler(int);
    signal(SIGINT, handler);
    ...
    /* complicated program */
    myput("important message\n");
    ...
    /* more program */
}

void handler(int sig) {
    ...
    /* deal with signal */
    myput("equally important "
        "message\n");
}
Asynchronous Signals (4)

```c
char buf[BSIZE];
int pos;
void myput(char *str) {
    int len = strlen(str);
    for (int i=0; i<len; i++, pos++) {
        buf[pos] = str[i];
        if ((buf[pos] == '\n') || (pos == BSIZE-1)) {
            write(1, buf, pos+1);
            pos = -1;
        }
    }
}
```
Async-Signal Safety

• Which library functions are safe to use within signal handlers?

- abort
- accept
- access
- aio_error
- aio_return
- aio_suspend
- alarm
- bind
- cfgetispeed
- cfgeto-speed
- cfsetispeed
- cfseto-speed
- chdir
- chmod
- chown
- clock_gettime
- close
- connect
- creat
- dup
- dup2
- execle
- execv
- execve
- _exit
- fchmod
- fchown
- fcntl
- fdatasync
- fork
- fpathconf
- fstat
- fsync
- lseek
- lstat
- mkdir
- mkfifo
- open
- pathconf
- pause
- pipe
- poll
- posix_trace_event
- pselect
- raise
- read
- readlink
- recv
- recvfrom
- recvmsg
- rename
- rmdir
- select
- sem_post
- send
- sendmsg
- sendto
- setgid
- setpgid
- setsid
- setsockopt
- setuid
- shutdown
- sigaction
- sigaddset
- sigdelset
- sigemptyset
- sigfillset
- sigismember
- signal
- sigpause
- sigpending
- sigprocmask
- sigqueue
- sigsuspend
- sleep
- socket
- socketpair
- stat
- symlink
- sysconf
- tcdrain
- tcflow
- tcflush
- tcgetattr
- tcgetpgrp
- tcgetpgrp
- tcgetpgrp
- tcpclose
- tcpsendbreak
- tcsetattr
- tcsetw
- tcsetpgrp
- time
- timer_getoverrun
- timer_getpgrp
- timer_getpgrp
- timer_gettime
- timer_settime
- times
- umask
- uname
- unlink
- utime
- wait
- waitpid
- write
What’s Inside A Disk Drive?

- Arm
- Spindle
- Platters
- Actuator
- Electronics (including a processor and memory!)
- SCSI connector

Image courtesy of Seagate Technology
Disk Geometry

- Disks consist of platters, each with two surfaces
- Each surface consists of concentric rings called tracks
- Each track consists of sectors separated by gaps
Disk Geometry (Multiple-Platter View)

- Aligned tracks form a cylinder
Disk Capacity

• **Capacity**: maximum number of bits that can be stored
  – capacity expressed in units of gigabytes (GB), where
    1 GB = $2^{30}$ Bytes ≈ $10^9$ Bytes

• Capacity is determined by these technology factors:
  – **recording density** (bits/in): number of bits that can be squeezed
    into a 1 inch segment of a track
  – **track density** (tracks/in): number of tracks that can be squeezed
    into a 1 inch radial segment
  – **areal density** (bits/in$^2$): product of recording and track density

• Modern disks partition tracks into disjoint subsets called
  **recording zones**
  – each track in a zone has the same number of sectors, determined
    by the circumference of innermost track
  – each zone has a different number of sectors/track
Computing Disk Capacity

Capacity = (# bytes/sector) x (avg. # sectors/track) x
(# tracks/surface) x (# surfaces/platter) x
(# platters/disk)

Example:
- 512 bytes/sector
- 600 sectors/track (on average)
- 40,000 tracks/surface
- 2 surfaces/platter
- 5 platters/disk

Capacity = 512 x 600 x 40000 x 2 x 5
= 122,880,000,000
= 113.88 GB
Disk Operation (Single-Platter View)

The disk surface spins at a fixed rotational rate by moving radially, the arm can position the read/write head over any track.

The read/write head is attached to the end of the arm and flies over the disk surface on a thin cushion of air.
Disk Operation (Multi-Platter View)

Read/write heads move in unison from cylinder to cylinder.

Arm

Spindle
Disk Structure: Top View of Single Platter

Surface organized into tracks

Tracks divided into sectors
Disk Access

Head in position above a track
Disk Access

Rotation is counter-clockwise
Disk Access – Read

About to read blue sector
Disk Access – Read

After BLUE read

After reading blue sector
Disk Access – Read

After BLUE read

Red request scheduled next
Disk Access – Seek

After **BLUE** read

Seek for **RED**

Seek to red’s track
Disk Access – Rotational Latency

After **BLUE** read

Seek for **RED**

Rotational latency

Wait for red sector to rotate around
Disk Access – Read

After BLUE read

Seek for RED

Rotational latency After RED read

Complete read of red
Disk Access – Service Time Components

After BLUE read
Data transfer

Seek for RED
Seek

Rotational latency After RED read
Rotational latency

Data transfer
Disk Access Time

• Average time to access some target sector approximated by:
  – $T_{\text{access}} = T_{\text{avg seek}} + T_{\text{avg rotation}} + T_{\text{avg transfer}}$

• Seek time ($T_{\text{avg seek}}$)
  – time to position heads over cylinder containing target sector
  – typical $T_{\text{avg seek}}$ is 3–9 ms

• Rotational latency ($T_{\text{avg rotation}}$)
  – time waiting for first bit of target sector to pass under r/w head
  – typical rotation speed $R = 7200$ RPM
  – $T_{\text{avg rotation}} = \frac{1}{2} \times \frac{1}{R} \times 60$ sec/1 min

• Transfer time ($T_{\text{avg transfer}}$)
  – time to read the bits in the target sector
  – $T_{\text{avg transfer}} = \frac{1}{R} \times \frac{1}{(\text{avg # sectors/track})} \times 60$ secs/1 min
Disk Access Time Example

• Given:
  – rotational rate = 7,200 RPM
  – average seek time = 9 ms
  – avg # sectors/track = 600

• Derived:
  – $T_{avg\ rotation} = \frac{1}{2} \times (60 \text{ secs/7200 RPM}) \times 1000 \text{ ms/sec} = 4 \text{ ms}$
  – $T_{avg\ transfer} = \frac{60}{7200} \text{ RPM} \times \frac{1}{600} \text{ secs/track} \times 1000 \text{ ms/sec} = 0.014 \text{ ms}$
  – $T_{access} = 9 \text{ ms} + 4 \text{ ms} + 0.014 \text{ ms}$

• Important points:
  – access time dominated by seek time and rotational latency
  – first bit in a sector is the most expensive, the rest are free
  – SRAM access time is about 4 ns/doubleword, DRAM about 60 ns
    » disk is about 40,000 times slower than SRAM
    » 2,500 times slower than DRAM
Logical Disk Blocks

• Modern disks present a simpler abstract view of the complex sector geometry:
  – the set of available sectors is modeled as a sequence of b-sized logical blocks (0, 1, 2, ...)

• Mapping between logical blocks and actual (physical) sectors
  – maintained by hardware/firmware device called disk controller
  – converts requests for logical blocks into (surface, track, sector) triples

• Allows controller to set aside spare cylinders for each zone
  – accounts for the difference in “formatted capacity” and “maximum capacity”
I/O Bus

- CPU chip
  - Register file
  - ALU
  - Bus interface

- System bus
- Memory bus
- Main memory

- I/O bus
- Expansion slots for other devices such as network adapters.

- USB controller
  - Mouse
  - Keyboard
- Graphics adapter
  - Monitor
- Disk controller
  - Disk
Reading a Disk Sector (1)

CPU initiates a disk read by writing a command, logical block number, and destination memory address to a port (address) associated with disk controller.
Disk controller reads the sector and performs a direct memory access (DMA) transfer into main memory.
When the DMA transfer completes, the disk controller notifies the CPU with an interrupt (i.e., asserts a special “interrupt” pin on the CPU).
Solid-State Disks (SSDs)

- Pages: 512KB to 4KB; blocks: 32 to 128 pages
- Data read/written in units of pages
- Page can be written only after its block has been erased
- A block wears out after 100,000 repeated writes
SSD Performance Characteristics

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential read tput</td>
<td>250 MB/s</td>
<td>Sequential write tput</td>
<td>170 MB/s</td>
</tr>
<tr>
<td>Random read tput</td>
<td>140 MB/s</td>
<td>Random write tput</td>
<td>14 MB/s</td>
</tr>
<tr>
<td>Random read access</td>
<td>30 us</td>
<td>Random write access</td>
<td>300 us</td>
</tr>
</tbody>
</table>

- Why are random writes so slow?
  - erasing a block is slow (around 1 ms)
  - modifying a page triggers a copy of all useful pages in the block
    » find a used block (new block) and erase it
    » write the page into the new block
    » copy other pages from old block to the new block
SSD Tradeoffs vs Rotating Disks

• Advantages
  – no moving parts → faster, less power, more rugged

• Disadvantages
  – have the potential to wear out
    » mitigated by “wear-leveling logic” in flash translation layer
    » e.g. Intel X25 guarantees 1 petabyte ($10^{15}$ bytes) of random writes before they wear out
  – in 2010, about 100 times more expensive per byte
  – in 2017, about 6 times more expensive per byte

• Applications
  – smart phones, laptops
  – Apple “Fusion” drives
Reading a File on a Rotating Disk

• Suppose the data of a file are stored on consecutive disk sectors on one track
  – this is the best possible scenario for reading data quickly
    » single seek required
    » single rotational delay
    » all sectors read in a single scan
Quiz 2

We have two files on the same (rotating) disk. The first file’s data resides in consecutive sectors on one track, the second in consecutive sectors on another track. It takes a total of $t$ seconds to read all of the first file then all of the second file.

Now suppose the files are read concurrently, perhaps a sector of the first, then a sector of the second, then the first, then the second, etc. Compared to reading them sequentially, this will take

a) less time
b) about the same amount of time
c) more time
Quiz 3

We have two files on the same solid-state disk. Each file’s data resides in consecutive blocks. It takes a total of $t$ seconds to read all of the first file then all of the second file.

Now suppose the files are read concurrently, perhaps a block of the first, then a block of the second, then the first, then the second, etc. Compared to reading them sequentially, this will take

a) less time
b) about the same amount of time
c) more time
### Storage Trends

**SRAM**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$/MB</td>
<td>2,900</td>
<td>320</td>
<td>256</td>
<td>100</td>
<td>75</td>
<td>60</td>
<td>25</td>
<td>116</td>
</tr>
<tr>
<td>access (ns)</td>
<td>150</td>
<td>35</td>
<td>15</td>
<td>3</td>
<td>2</td>
<td>1.5</td>
<td>1.3</td>
<td>115</td>
</tr>
</tbody>
</table>

**DRAM**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$/MB</td>
<td>880</td>
<td>100</td>
<td>30</td>
<td>1</td>
<td>0.1</td>
<td>0.06</td>
<td>0.02</td>
<td>44,000</td>
</tr>
<tr>
<td>access (ns)</td>
<td>200</td>
<td>100</td>
<td>70</td>
<td>60</td>
<td>50</td>
<td>40</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>typical size (MB)</td>
<td>0.256</td>
<td>4</td>
<td>16</td>
<td>64</td>
<td>2,000</td>
<td>8,000</td>
<td>16,000</td>
<td>62,500</td>
</tr>
</tbody>
</table>

**Disk**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$/GB</td>
<td>100,000</td>
<td>8,000</td>
<td>300</td>
<td>10</td>
<td>5</td>
<td>.3</td>
<td>0.03</td>
<td>3,333,333</td>
</tr>
<tr>
<td>access (ms)</td>
<td>75</td>
<td>28</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>25</td>
</tr>
<tr>
<td>typical size (GB)</td>
<td>.01</td>
<td>.16</td>
<td>1</td>
<td>20</td>
<td>160</td>
<td>1,500</td>
<td>3,000</td>
<td>300,000</td>
</tr>
</tbody>
</table>
## CPU Clock Rates

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>286</td>
<td>386</td>
<td>Pentium</td>
<td>P-III</td>
<td>P-4</td>
<td>Core 2</td>
<td>Core i7</td>
<td>---</td>
</tr>
<tr>
<td>Clock rate (MHz)</td>
<td>6</td>
<td>20</td>
<td>150</td>
<td>600</td>
<td>3300</td>
<td>2000</td>
<td>3000</td>
<td>500</td>
</tr>
<tr>
<td>Cycle time (ns)</td>
<td>166</td>
<td>50</td>
<td>6</td>
<td>1.6</td>
<td>0.3</td>
<td>0.50</td>
<td>0.33</td>
<td>500</td>
</tr>
<tr>
<td>Cores</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Effective cycle time (ns)</td>
<td>166</td>
<td>50</td>
<td>6</td>
<td>1.6</td>
<td>0.3</td>
<td>0.25</td>
<td>0.08</td>
<td>2075</td>
</tr>
</tbody>
</table>

Inflection point in computer history when designers hit the “Power Wall”
The CPU-Memory Gap

The gap widens between DRAM, disk, and CPU speeds

![Graph showing the gap between DRAM, disk, and CPU speeds over time]

- Disk seek time
- Flash SSD access time
- DRAM access time
- SRAM access time
- CPU cycle time
- Effective CPU cycle time

Light travels ~11 inches in a nanosecond!
Memory Hierarchies

• Some fundamental and enduring properties of hardware and software:
  – fast storage technologies cost more per byte, have less capacity, and require more power (heat!)
  – the gap between CPU and main memory speed is widening
  – well written programs tend to exhibit good locality

• These fundamental properties complement each other beautifully

• They suggest an approach for organizing memory and storage systems known as a memory hierarchy
An Example Memory Hierarchy

- **Registers**
  - CPU registers hold words retrieved from L1 cache
  - Larger, faster, costlier per byte

- **L1 cache (SRAM)**
  - L1 cache holds cache lines retrieved from L2 cache
  - Smaller, faster, costlier per byte

- **Main memory (DRAM)**
  - Main memory holds disk blocks retrieved from local disks
  - Larger, slower, cheaper per byte

- **Local secondary storage (local disks)**
  - Local disks hold files retrieved from disks on remote network servers

- **Remote secondary storage (distributed file systems, cloud storage)**
  - Remote secondary storage holds files retrieved from distributed file systems and cloud storage
Putting Things Into Perspective ...

• Reading from:
  – ... the L1 cache is like grabbing a piece of paper from your desk (3 seconds)
  – ... the L2 cache is picking up a book from a nearby shelf (14 seconds)
  – ... main system memory is taking a 4-minute walk down the hall to talk to a friend
  – ... a hard drive is like leaving the building to roam the earth for one year and three months
Disks Are Important

• **Cheap**
  – cost/byte much less than SSDs

• (fairly) **Reliable**
  – data written to a disk is likely to be there next year

• **Sometimes fast**
  – data in consecutive sectors on a track can be read quickly

• **Sometimes slow**
  – data in randomly scattered sectors takes a long time to read
Abstraction to the Rescue

• Programs don't deal with sectors, tracks, and cylinders
• Programs deal with files
  – maze.c rather than an ordered collection of sectors
  – OS provides the implementation
Implementation Problems

• Speed
  – use the hierarchy
    » copy files into RAM, copy back when done
  – optimize layout
    » put sectors of a file in consecutive locations
  – use parallelism
    » spread file over multiple disks
    » read multiple sectors at once
Implementation Problems

• Reliability
  – computer crashes
    » what you thought was safely written to the file never made it to the disk — it’s still in RAM, which is lost
    » worse yet, some parts made it back to disk, some didn’t
      • you don’t know which is which
      • on-disk data structures might be totally trashed
  – disk crashes
    » you had backed it up … yesterday
  – you screw up
    » you accidentally delete the entire directory containing your strings/performance solution
Implementation Problems

• Reliability solutions
  – computer crashes
    » transaction-oriented file systems
    » on-disk data structures always in well defined states
  – disk crashes
    » files stored redundantly on multiple disks
  – you screw up
    » file system automatically keeps "snapshots" of previous versions of files