SYSTEMS! The GRAND Illusion!

how computers *really* work!

HOW are we going to make this exciting?

- Work together?
  - Groups of 3?

- Ask crazy questions ALLLLLLLL the time…

- BUILD ON the knowledge and skills you already have!

- *INTRODUCTIONS*!!!!

WHAT?!?

- Languages?
- Homework?
- Projects?
- Mock Interviews?
- ???

Building a CAVE!

- Major Research Infrastructure
  Estimated Total Costs: $6,125,000.00
- Virtual Location VR/Film Studio (10,000 ft2) $4,400,000.00
  - 270°, 20-foot high, 1,000-screen LED Cyclorama, plus replacement screens and protective cases
  - Ceiling rigging system for lights and cameras
  - Stype motion-capture camera package
  - Rendering and editing nodes, sync generator
  - Brain Bar real-time editing studio
- Non-equipment Costs: $300,000.00
  - Shipping, Customs, Set-up Labour, Project Manager, On-Site Training
- Computer and Visualization Hardware $1,000,000.00
  - BCNet Advanced Network fibre-optic connection (available at VITP)
  - 1 main Server and 12 nDisplay servers with 1 PB storage on an infiniband network
  - Temperature-controlled, acoustically isolated server storage
  - Separate physical networks for camera tracking, display forwarding, remote control
  - Brain Interface/Monitoring EEG (electroencephalography).
- Spatial Audio Recording Studio (1,500 ft2): $300,000.00 (Microphones and recording equipment for 360° spatial audio)
  - RedNet Rl, Eigenmike, Genelec 8361A x 12, RedNet MP8R x 3
  - RME Digiface Dante, Laptop
  - Sound baffling and other acoustic renovations
- Set Fabrication Workshop I Staging, Rehearsal & Demo Space (3,000 ft2) Infrastructure: $100,000.00
  - Physical fabrication workshop, rehearsal and demo space, with 3D printers and other fabrication tools to create set pieces and modular built-environnent equipment
  - 2 x HTC Vive workstations for VR/XR demos and testing
  - Black box area for motion-capture rehearsals and small-scale green-screen work
- Office Space (500 ft2) Infrastructure and Renovation: $25,000.00
  - Acoustically isolated office space, with workstations, for Operations Manager, admin staff and technicians

The book! Easy to read, good exercises?

- ”*Standard*” out there!
- Dialogues?!
- Chapter 2: Intro!
  - Hopefully a bit of a review of 5008 or OS concepts from undergrad?
  - *Do* compile and run all code! ☺
- Chapter 4: Virtualization!
  - Homework by Monday in time for mock interviews?

Chapter 2: Introduction

- Sharing resources
  - *virtualization*
  - the OS takes a physical resource (such as the **processor**, or **memory**, or a **disk**) and provides an API to make an *easy-to-use virtual form* of itself
    - the operating system as a “virtual machine”

- *how*: what mechanisms and policies are implemented by the OS to attain virtualization?
  - How does the OS do so efficiently?
    - Lessons for ALLLLLLLLLL software systems!
  - What hardware support is needed?

The “API”

- APIs for *running* a program, or allocating memory, or accessing a file…
  - Where a file is also a program!
  - *System calls* are the API!

- Check out Linux system calls!
  - https://man7.org/linux/man-pages/man2/syscalls.2.html
  - Which ones are most interesting to you and why?

Code! Can anyone run this? How?

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/time.h>
4  #include <assert.h>
5  #include "common.h"
6
7  int
8  main(int argc, char *argv[])
9  {
10     if (argc != 2) {
11         fprintf(stderr, "usage: cpu <string>\n");
12         exit(1);
13     }
14     char *str = argv[1];
15     while (1) {
16         Spin(1);
17         printf("%s\n", str);
18     }
19     return 0;
20 }
```

Figure 2.1: **Simple Example: Code That Loops And Prints (cpu.c)**

```
prompt> gcc -o cpu cpu.c -Wall
prompt> ./cpu "A"
A
A
A
A
^C
prompt>
```

```
prompt> ./cpu A & ./cpu B & ./cpu C & ./cpu D &
[1] 7353
[2] 7354
[3] 7355
[4] 7356
A
B
D
C
A
B
D
C
A
...
```

Virtualizing the CPU

- the GRAND illusion!
  - the system has a *very large number* of virtual CPUs
  - allowing many programs to seemingly run at once
    - virtualizing the CPU, the focus of the first major part of this book

- if two programs want to run at a particular time, which should run?
  - **policies** are used in many different places within an OS to answer these types of questions
  - also the basic **mechanisms** that operating systems implement

Virtualizing Memory

```
5
6   int
7   main(int argc, char *argv[])
8   {
9       int *p = malloc(sizeof(int));                   // a1
10      assert(p != NULL);
11      printf("(%d) address pointed to by p: %p\n",
12             getpid(), p);                            // a2
13      *p = 0;                                         // a3
14      while (1) {
15          Spin(1);
16          *p = *p + 1;
17          printf("(%d) p: %d\n", getpid(), *p);       // a4
18      }
19      return 0;
20  }
```

Figure 2.3: **A Program That Accesses Memory (`mem.c`)**

```
prompt> ./mem
(2134) address pointed to by p: 0x200000
(2134) p: 1
(2134) p: 2
(2134) p: 3
(2134) p: 4
(2134) p: 5
^C
```

```
prompt> ./mem &; ./mem &
[1] 24113
[2] 24114
(24113) address pointed to by p: 0x200000
(24114) address pointed to by p: 0x200000
(24113) p: 1
(24114) p: 1
(24114) p: 2
(24113) p: 2
(24113) p: 3
(24114) p: 3
(24113) p: 4
(24114) p: 4
...
```

What?

- each process accesses its own **private** virtual address space
  - the OS maps it onto the physical memory of the machine
- LOOKS like a process has physical memory all to itself!
- **physical memory** is a shared resource, managed by the operating system
  - hardware makes this FAST!

Concurrency

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "common.h"
4  #include "common_threads.h"
5
6  volatile int counter = 0;
7  int loops;
8
9  void *worker(void *arg) {
10     int i;
11     for (i = 0; i < loops; i++) {
12         counter++;
13     }
14     return NULL;
15 }
16
17 int main(int argc, char *argv[]) {
18     if (argc != 2) {
19         fprintf(stderr, "usage: threads <value>\n");
20         exit(1);
21     }
22     loops = atoi(argv[1]);
23     pthread_t p1, p2;
24     printf("Initial value : %d\n", counter);
25
26     Pthread_create(&p1, NULL, worker, NULL);
27     Pthread_create(&p2, NULL, worker, NULL);
28     Pthread_join(p1, NULL);
29     Pthread_join(p2, NULL);
30     printf("Final value   : %d\n", counter);
31     return 0;
32 }
```

```
prompt> gcc -o thread thread.c -Wall -pthread
prompt> ./thread 1000
Initial value : 0
Final value   : 2000
```

```
prompt> ./thread 100000
Initial value : 0
Final value   : 143012    // huh??
prompt> ./thread 100000
Initial value : 0
Final value   : 137298    // what the??
```

What is “volatile”?

- prevents compiler optimizations that might render code incorrect in the presence of certain asynchronous events
  - not permitted to cache it in a register -- a common optimization that would be disastrous if that variable were shared among multiple threads!
- **volatile: The Multithreaded Programmer's Best Friend (2001!)**
  - https://www.drdobbs.com/cpp/volatile-the-multithreaded-programmers-b/184403766

Persistence

- input/output or I/O device; in modern systems, a hard drive is a common repository for longlived information, although solid-state drives (SSDs) are now more common!

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <assert.h>
4  #include <fcntl.h>
5  #include <sys/types.h>
6
7  int main(int argc, char *argv[]) {
8      int fd = open("/tmp/file", O_WRONLY|O_CREAT|O_TRUNC,
9                    S_IRWXU);
10     assert(fd > -1);
11     int rc = write(fd, "hello world\n", 13);
12     assert(rc == 13);
13     close(fd);
14     return 0;
15 }
```

Figure 2.6: **A Program That Does I/O (`io.c`)**

File System's "API"

- System calls
  - open(), opens the file and creates it
  - write(), writes some data to the file
  - close(), simply closes the file thus indicating the program won't be writing any more data to it

- file system handles the requests
  - and returns some kind of error code to the user

QUESTIONS?

- Syllabus
  - Sept 30th? Nov 11th? Nov 25th?
- Grades?
- Reading/presenting a research paper?