Operating Systems CMPSCI 377 Spring 2020

Mark Corner
UMass Amherst

Clicker Question

From the reading, what is the primary API that an Operating System gives to user programs?

(A) System Calls

(B) Disk Drive

(C) Linux

(D) Assembly Instructions

Answer on Next Slide

Today's Class

- Comments and Reflections
- Introduction to Operating Systems
- Virtualization
- Virtualization the CPU
- Virtualizing Memory
- Virtualizing Execution with Concurrency
- Virtualizing Storage with Persistent I/O

Discussion Section #1

The topics for the upcoming discussion (F/M) section include:

- TA introduction
- Programming environment setup and questions
- Discussion work/homework

IMPORTANT: Bring a laptop where you have **already** run the setup instructions for Docker. Have docker installed and have run a container from the mcorner/os377 image. It takes a while to download it. See Development Environment doc.

Please give the teaching assistant (TA) leading your discussion all the attention and respect they deserve. This is a learning experience for them and they are doing their best to help you in this course.

Comments and Reflections

Do not forget to read the syllabus. Important.

Be respectful of who is around you.

We promote a respectful learning environment, if you need to talk - leave.

If you want to play WoW - leave.

If you are watching Friends (even with the subtitles), you should leave (and find better things to watch).

IOW - If you are doing anything but being present - leave.

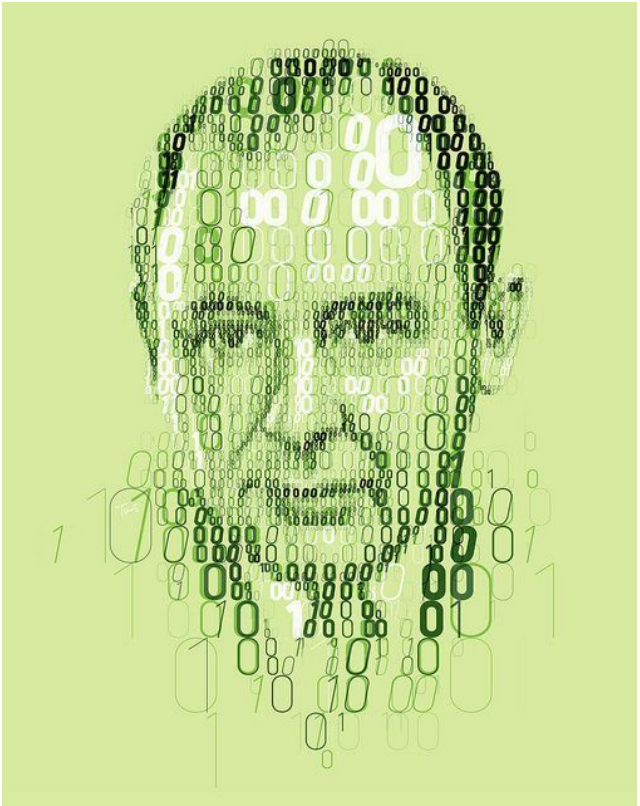
Introduction to Operating Systems

What happens when a program runs?

Introduction to Operating Systems

What happens when a program runs?

Well, a running program does one very simple thing: **it executes instructions**. Many millions (and these days, even billions) of times every second, the processor fetches an instruction from memory, decodes it (i.e., figures out which instruction this is), and executes it (i.e., it does the thing that it is supposed to do, like add two numbers together, access memory, check a condition, jump to a function, and so forth). After it is done with this instruction, the processor moves on to the next instruction, and so on, and so on, until the program finally completes.



Introduction to Operating Systems

Thus, we have just described the Von Neumann model of computing.

Sounds simple, right?

But in this class, we will be learning that while a program runs, a lot of other wild things are going on with the primary goal of making the system easy to use.

Introduction to Operating Systems

There is a body of software that is responsible for making it easy to run programs.

Introduction to Operating Systems

There is a body of software that is responsible for making it easy to run programs.

Allowing programs to share memory…

Introduction to Operating Systems

There is a body of software that is responsible for making it easy to run programs.

Allowing programs to share memory…

Enabling programs to interact with devices…

Introduction to Operating Systems

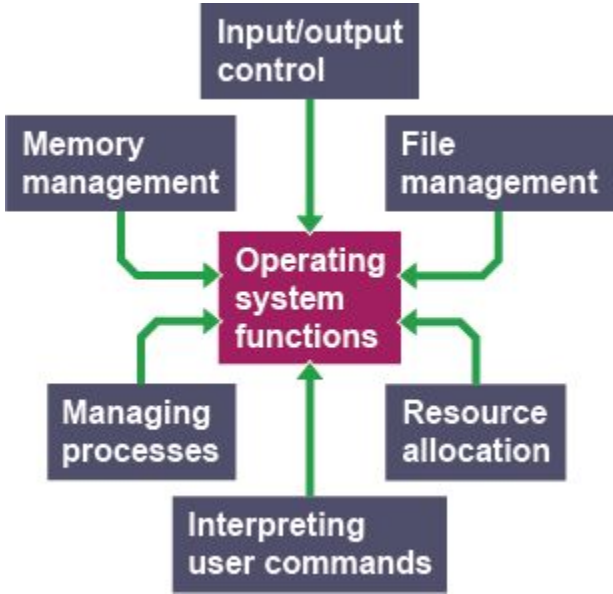
There is a body of software that is responsible for making it easy to run programs.

Allowing programs to share memory…

Enabling programs to interact with devices…

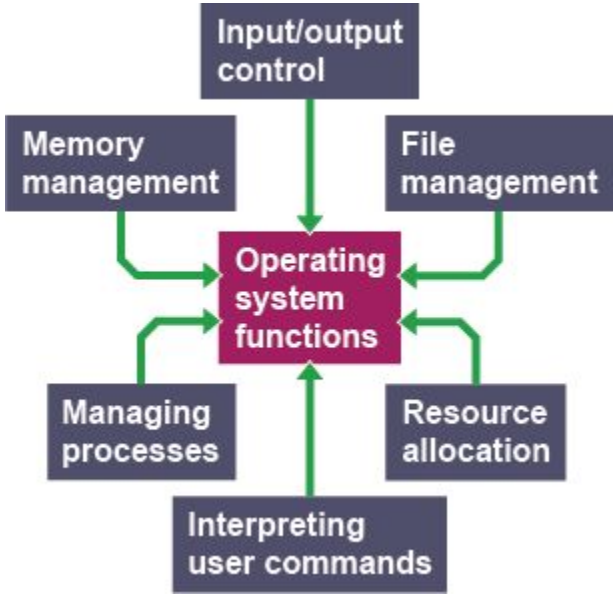
… and other fun stuff like that!

This body of software is called
the **operating system** (OS).



Virtualization

The primary mechanism the OS uses to accomplish this is **virtualization**.

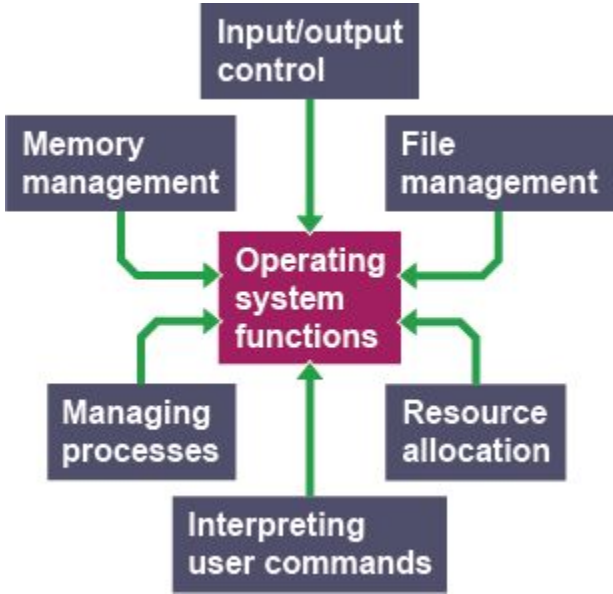


Virtualization

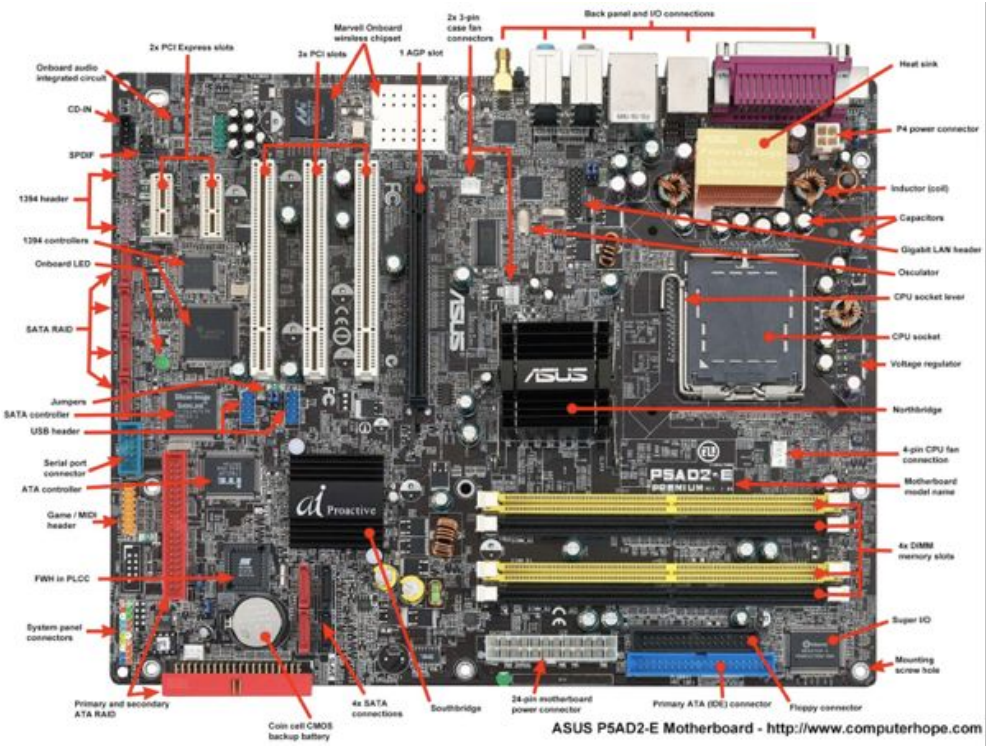
The primary mechanism the OS uses to accomplish this is **virtualization**.

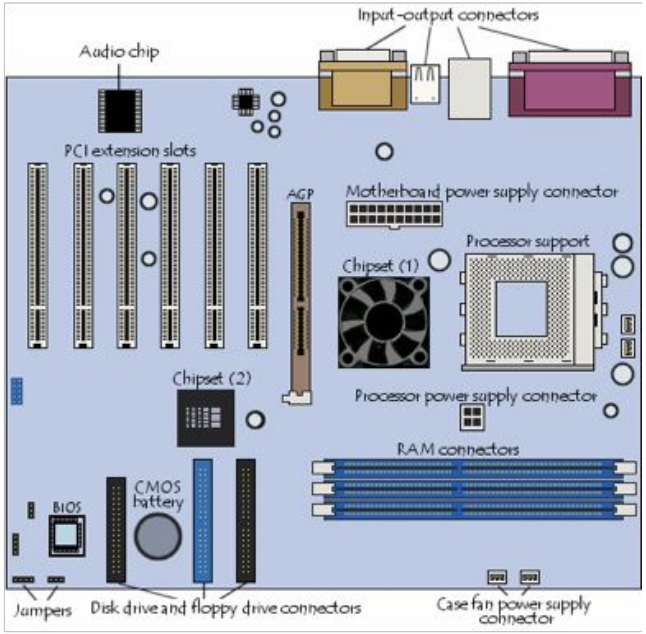
The OS takes a physical resource (e.g., processor, memory, disk) and transforms it into a more general, powerful, and easy-to-use **virtual** form of itself.

Thus, we sometimes refer to the operating system as a *virtual machine*.



What are we Virtualizing?





Generic Architecture

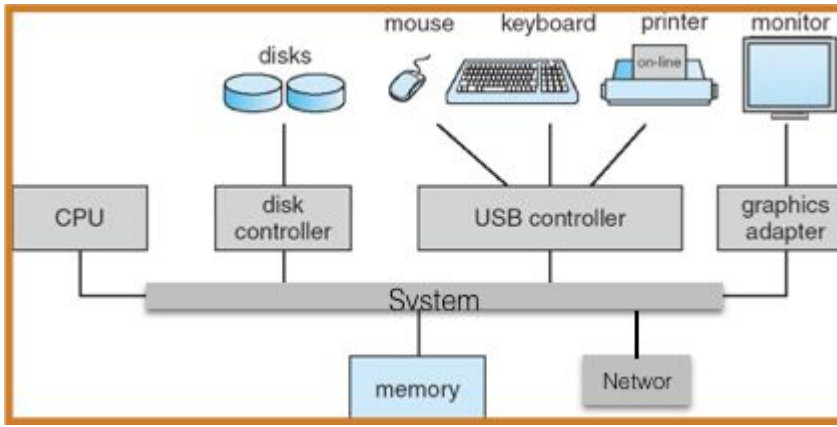
CPU: performs the actual computation

Multiple “cores” common

I/O devices: terminal, disks, video board, network, printer, etc.

Memory: RAM containing data and programs used by CPU

System bus: communication CPU, memory, peripherals



Virtualizing the CPU

```
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  }
```

Code that Loops and Prints (cpu.c)

It doesn't do much.

In fact, all it does is call `Spin()` and prints the program argument in a loop - forever.

`Spin()` repeatedly checks the time and returns after a second has passed.

Clicker Question

If a process A loads a value into a spot in memory and then another process B runs on the CPU, what value does B see in that memory location?

(A) Same

(B) Different

Answer on Next Slide

Virtualizing Memory

```
1   #include <unistd.h>
2   #include <stdio.h>
3   #include <stdlib.h>
4   #include "common.h"
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  }
```

Code that Loops and Prints (mem.c)

It doesn't do much.

In fact, all it does is print the address of a program variable, enter a loop and call `Spin()` and print the value of that variable - forever.

`Spin()` repeatedly checks the time and returns after a second has passed.

Clicker Question

If a machine's addresses go from 0 to FFFFFFFF, what kind of machine is this?

(A) 8 bit machine

(B) 16 bit machine

(C) 32 bit machine

(D) 64 bit machine

Answer on Next Slide

Clicker Question

If a machine's addresses go from 0 to FFFFFFFF, what kind of machine is this?

(A) 8 bit machine

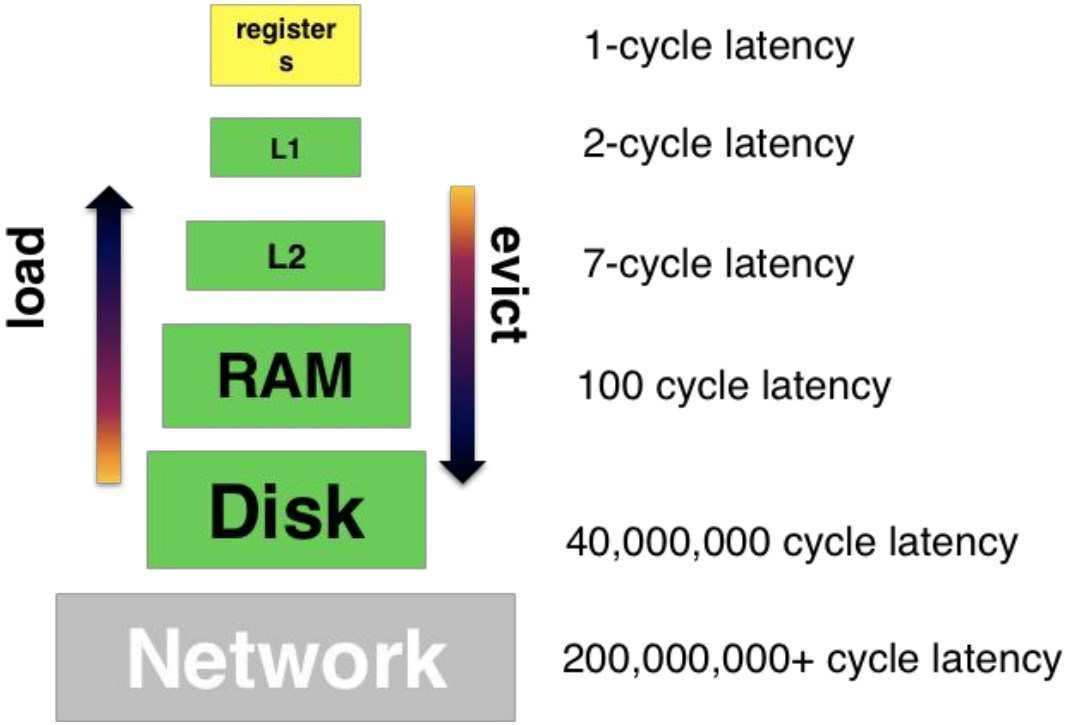
(B) 16 bit machine

(C) 32 bit machine

(D) 64 bit machine

$$\sum_{i=0}^{63} 2^i = 2^{64} - 1$$

Memory Hierarchy



Clicker Question

Listed from fastest to slowest, which is correct?

(A) Registers, L2 Cache, L1 Cache, RAM, Disk, Network

(B) Registers, L1 Cache, L2 Cache, RAM, Disk, Network

(C) Registers, RAM, L2 Cache, L1 Cache, Network, Disk,

(D) Registers, RAM, L2 Cache, L1 Cache, Disk, Network

Answer on Next Slide

Clicker Question

If a thread in A loads a value into a spot in memory and then another thread in process A runs on the CPU, what value does B see in that memory location?

(A) Same

(B) Different

Answer on Next Slide

Virtualizing Execution with Concurrency

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include "common.h"
4
5   volatile int counter = 0;
6   int loops;
7
8   void *worker(void *arg) {
9       int i;
10      for (i = 0; i < loops; i++) {
11          counter++;
12      }
13      return NULL;
14  }
15
```

A Multi-threaded program (threads.c)

This program does something interesting…

It creates two concurrently executing **threads** in a single process.

Both threads share a global variable (counter) and both threads update that shared variable in two independent threads of execution.

But, problems can arise here…

This makes multi-threaded programs hard.

Virtualizing Execution with Concurrency

```
16  int
17  main(int argc, char *argv[])
18  {
19      if (argc != 2) {
20          fprintf(stderr, "usage: threads <value>\n");
21          exit(1);
22      }
23      loops = atoi(argv[1]);
24      pthread_t p1, p2;
25      printf("Initial value : %d\n", counter);
26
27      Pthread_create(&p1, NULL, worker, NULL);
28      Pthread_create(&p2, NULL, worker, NULL);
29      Pthread_join(p1, NULL);
30      Pthread_join(p2, NULL);
31      printf("Final value   : %d\n", counter);
32      return 0;
33  }
```

A Multi-threaded program (threads.c)

This program does something interesting…

It creates two concurrently executing **threads** in a single process.

Both threads share a global variable (counter) and both threads update that shared variable in two independent threads of execution.

But, problems can arise here…

This makes multi-threaded programs hard.

Virtualizing Storage with Persistent I/O

```
1   #include <stdio.h>
2   #include <unistd.h>
3   #include <assert.h>
4   #include <fcntl.h>
5   #include <sys/types.h>
6
7   int
8   main(int argc, char *argv[])
9   {
10      int fd = open("/tmp/file", O_WRONLY | O_CREAT | O_TRUNC, S_IRWXU);
11      assert(fd > -1);
12      int rc = write(fd, "hello world\n", 13);
13      assert(rc == 13);
14      close(fd);
15      return 0;
16  }
```

A Program that does I/O
(io.c)

This program writes bytes to a file.

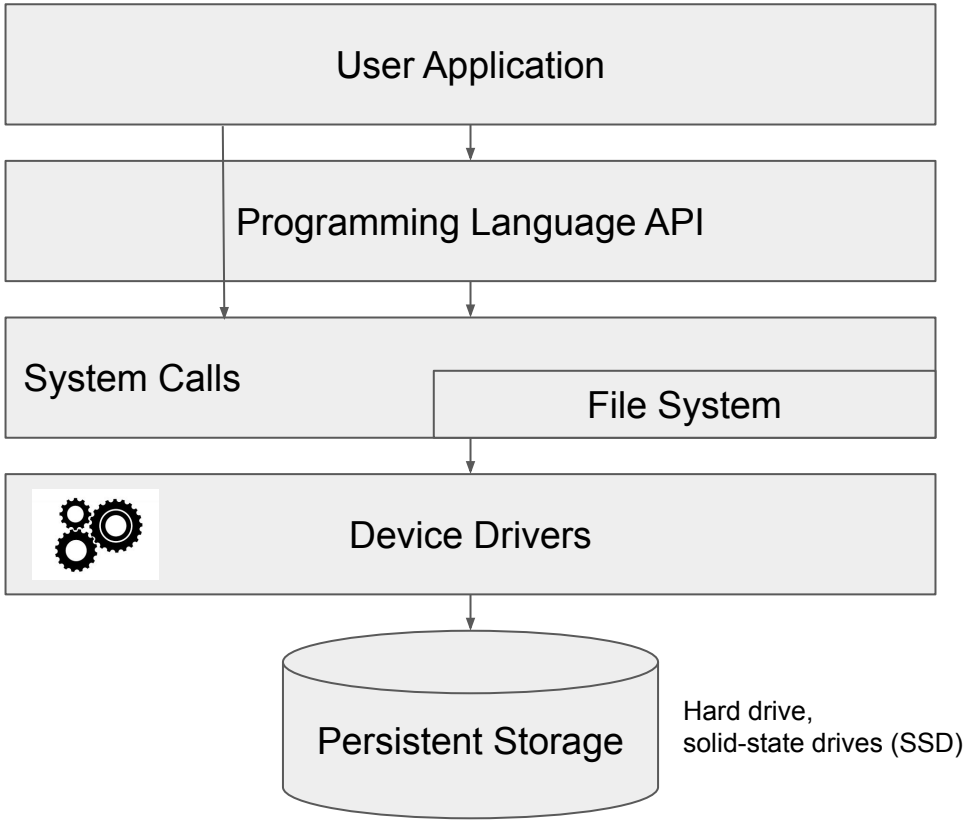
It uses the open() system call to create a new file.

It writes 13 bytes to that file.

In system memory, data can be easily lost, as devices such as DRAM store values in a volatile manner.

We need hardware and software to be able to store data persistently; such storage is thus critical to any system as users care a great deal about their data.

Virtualizing Storage with Persistent I/O



A Program that does I/O
(io.c)

The user application codes the program logic.

The PL API provides useful libraries and abstractions for working with "files" and "directories".

System calls work with a component of the OS called the **file system**.

Device drivers communicate directly to hardware that reads/writes bytes on a persistent storage medium.

Bytes are stored until overwritten.

OS Design Goals




Virtualize physical resources.

Handle tricky issues related to concurrency.

Provide useful (easy to use) abstractions - very fundamental, very important!

Provide high performance, minimize overheads.

Protection and isolation.

Others: energy efficiency, security, mobility, etc.

Different devices lead to different goals and optimizations.

Next Time...

Look at the schedule for reading (find link to schedule in pinned note in Piazza)

If you haven’t signed up for Piazza/Gradescope - see Moodle for instructions.

It is always best if you read the material (even if you don’t understand it) before class and then re-read it again afterwards. +1

Make sure you setup your laptop for discussion section