Operating Systems

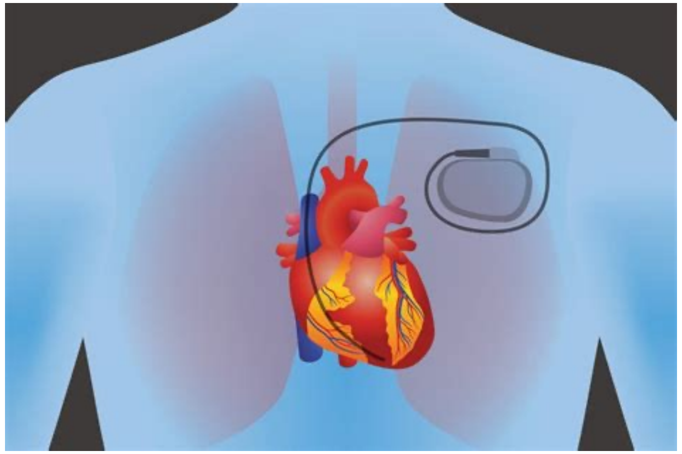
RUTGERS
UNIVERSITY | NEW BRUNSWICK

Computers everywhere Software everywhere







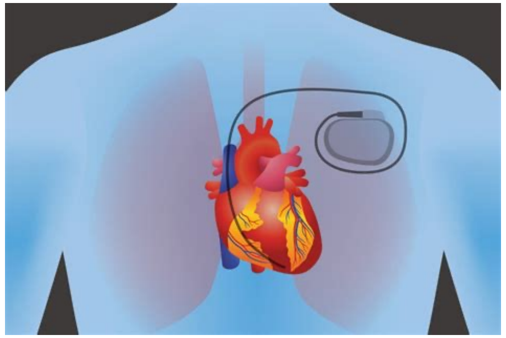


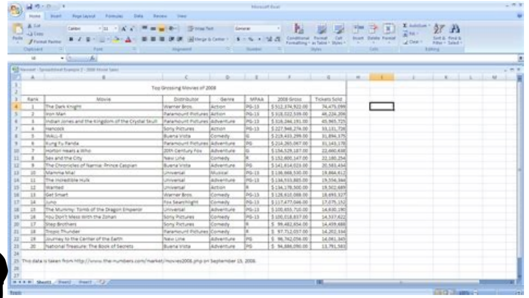
Making software useful means making it…

- Interact with human users

- Interact with the physical world

Hardware access





- Process data

Abstractions

- Easy to develop software

- Easy to run software

Resource isolation; performance

This Course: Operating Systems

Abstractions: How exactly does modern software use hardware?

Resource Management: How to isolate resources?

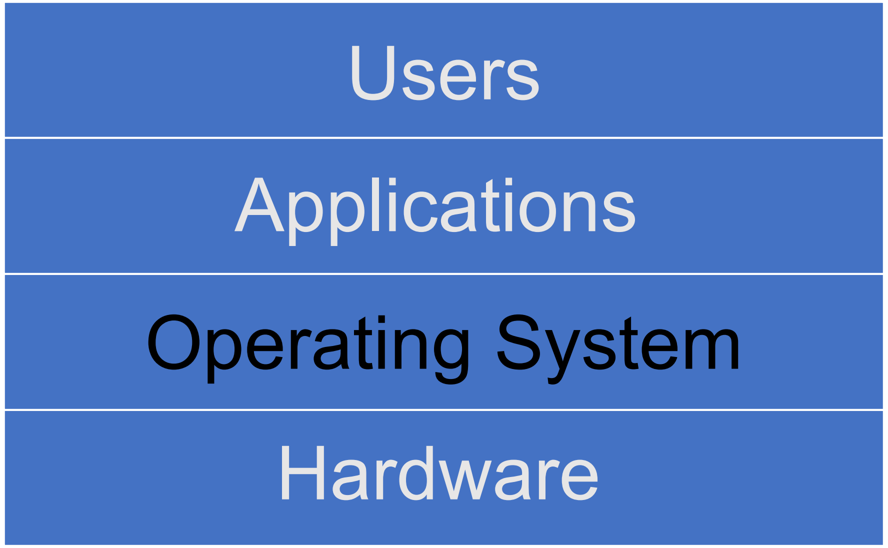
How to achieve the above correctly and with high performance?

Operating systems form the foundation of modern computing.

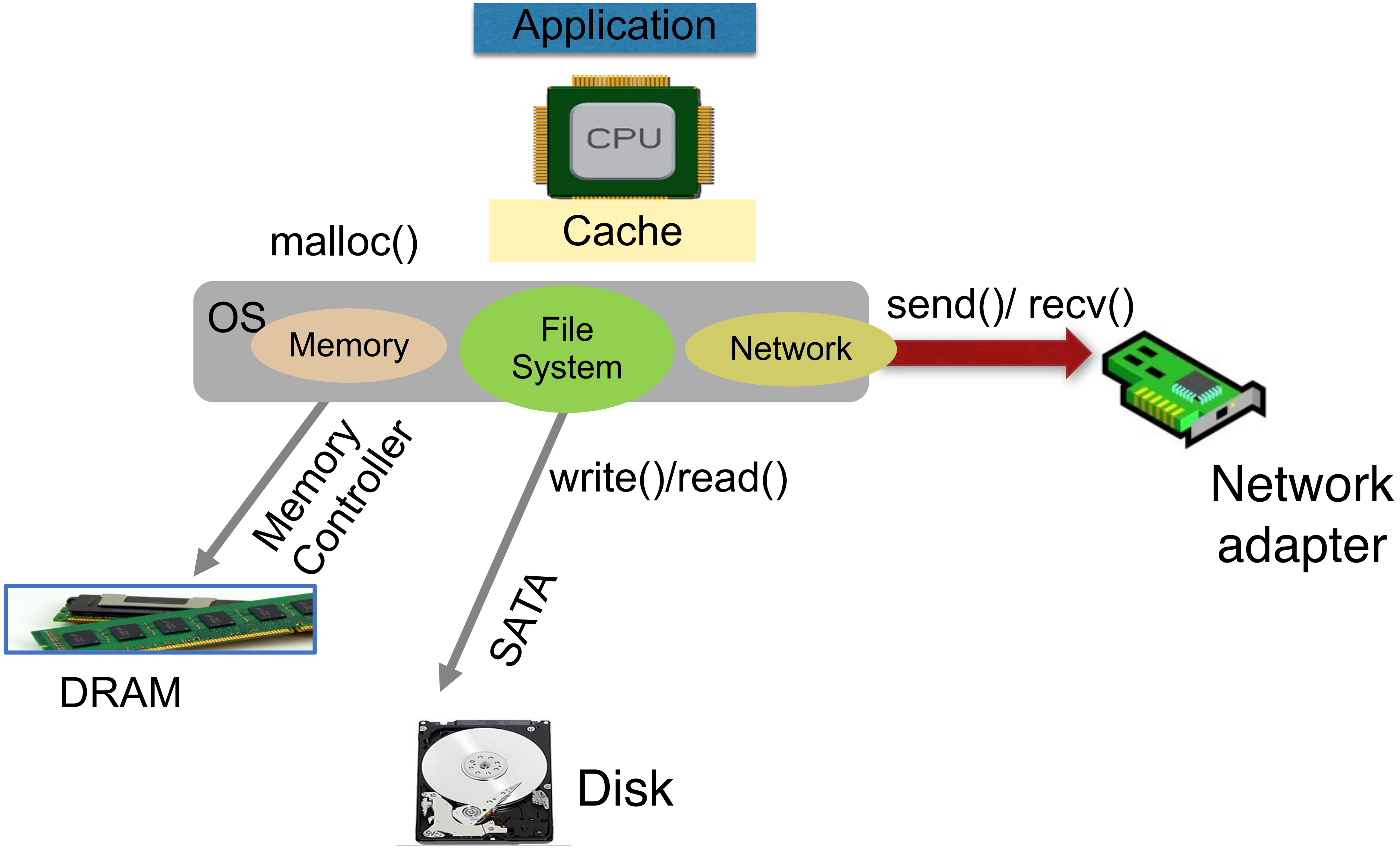
Conceptual learning and intensive programming

What is an Operating System?

Software that abstracts and manages hardware resources.



What hardware are we talking about?



Operating system provides…

- Software library (abstraction) between applications and hardware to make the hardware easier to use
  - Simple, uniform view of diverse hardware devices
- Mechanisms and policies for resource management, to provision and isolate hardware across many applications
  - Effective multi-tenant and multi-application systems

#1. Abstraction

What abstraction does modern OS typically provide for each resource?

- CPU: process and/or thread
- Memory: address space
- Storage: files

Advantages of OS providing abstraction?

- Allow applications to reuse common facilities
- Make different devices look the same
- Provide higher-level or more useful functionality

Challenges

- What are the correct abstractions?
- How much of the hardware capabilities should be exposed?

System Calls

- System call allows user to tell the OS what to do on hardware
- The OS provides a standard software interface (APIs)
- A typical OS exports a few hundred system calls
- Run programs, access memory, access hardware devices, …

#2. Resource Management

Want fair and efficient use of hardware across applications

Advantages of OS providing resource management:

- Protect applications from one another
- Provide efficient access to resources (cost, time, energy)
- Provide fair access to resources

Challenges

- What are the correct mechanisms?
- What are the correct policies?

Benefits of studying Operating Systems

Pragmatic: Understand the limits of software performance

Behavior of OS impacts entire machine

Tune application performance

Apply knowledge across many layers

- Computer architecture, programming languages, data structures and algorithms, and performance modeling

Puzzle solving: Fun to understand large, complex systems

Technology: Build, modify, or administer an operating system

Course Logistics

About us

- Faculty Instructor: Srinivas Narayana
  - http://www.cs.rutgers.edu/~sn624
  - sn624@rutgers.edu
  - Office hours in person Wed 4—5 pm ET (or by appointment)
    - Subject to change in the next 1—2 weeks. TA office hours TBA
  - Lectures Wed 8:30—11:30 ET
- TAs and Recitations: two UG sections, one G section
  - Qiongwen Xu (qx51), Harishankar Vishwanathan (hv90), Adithya Murugadass (am3372)
- Post q's to Piazza (Canvas announcement to sign up)
- Class info: http://www.cs.rutgers.edu/~sn624/416-F23/

Class philosophy

- We want you to learn and to be successful
- Attend recitations and office hours regularly to discuss material
- Be proactive: interact, ask, support.
  - Use Piazza
- Full video lectures will be made available

Grading

- 45% programming projects
- 20% weekly quizzes
- 15% mid term
- 20% final exam (cumulative)

- Schedule of projects, exams, etc. will be made available at https://www.cs.rutgers.edu/~sn624/416-F23

- Book: https://pages.cs.wisc.edu/~remzi/OSTEP/

- This course uses absolute grading. There is no curve

45% programming projects

- 4 large software projects
- Groups of 2. Pick partner and keep them throughout semester
- Program and short write-up required
- Use hosted VMs (will provide instructions for use)
- Hand projects in on Canvas

45% programming projects

- Please follow all instructions carefully and exactly
- You will lose significant points if:
  - We are unable to run your code
  - Your information (e.g., team member names and netids) is incorrect or incomplete
  - We do not receive your submission in a timely fashion

Collaboration and Integrity policies

- Intellectual collaboration is welcome and encouraged
- Do
  - Ask questions on Piazza
  - Discuss projects and problem sets with us and each other
  - Read references (textbooks, Internet tutorials) widely
  - Acknowledge each other and all the references in psets & project reports
- Each problem set & project has a prompt on collaboration
  - Include who you talked to, references (including on the web) you consulted
  - Be as accurate and complete as possible

Collaboration and Integrity policies

- All your written (coded) work must be your (team's) own
  - Understand the problem deeply and produce your own solutions
- Do not
  - blindly lift or incorporate other solutions
  - look at other people's code or solutions
  - copy code from the web (e.g., other people's GitHub projects)
  - post problem sets or projects (questions or solutions) on piazza, GitHub, Chegg, CourseHero, etc.
- Ask us for permission if you are ever in doubt
- We will check for plagiarism across submissions from this year and the last few years

Programming projects are time-intensive

- Cannot score high by pulling all-nighters close to the due date
- Please approach the projects (and this course) diligently from day 1
  - Get the most out of this course
- Use the projects to improve your programming skills
  - Job search, grad school, better learning outcomes

20% weekly quizzes

- Due every Tuesday night over the semester (including next Tue)
- Work individually
- Can consult the textbook and own notes
- No collaboration or searching for answers on the Internet
- We will consider the 10 highest scores out of 13

Late policy

- Don’t be late
- If you must be late, inform us in advance
- If you cannot inform us in advance (e.g., medical), provide official medical note of absence through the University
- Unexcused late submissions will result in losing significant fraction of points

24/7 Grading Policy

- You may not dispute a grade or request a regrade before 24 hours or after 7 days of receiving it

- Please contact us if you have a legitimate regrading request:

  - After 24 hours of receiving the grade: Please take the time to review your case before contacting the instructors

  - Before 7 days have elapsed: we don’t want to forget what the test/project was all about.

Help, Accommodations, etc.

- We'll make every effort to accommodate reasonable requests that support your learning better

- sn624@cs.rutgers.edu

- Course staff is committed to help you succeed

Recommendation Letters

For students applying to grad school or jobs, and seeking a reference letter:

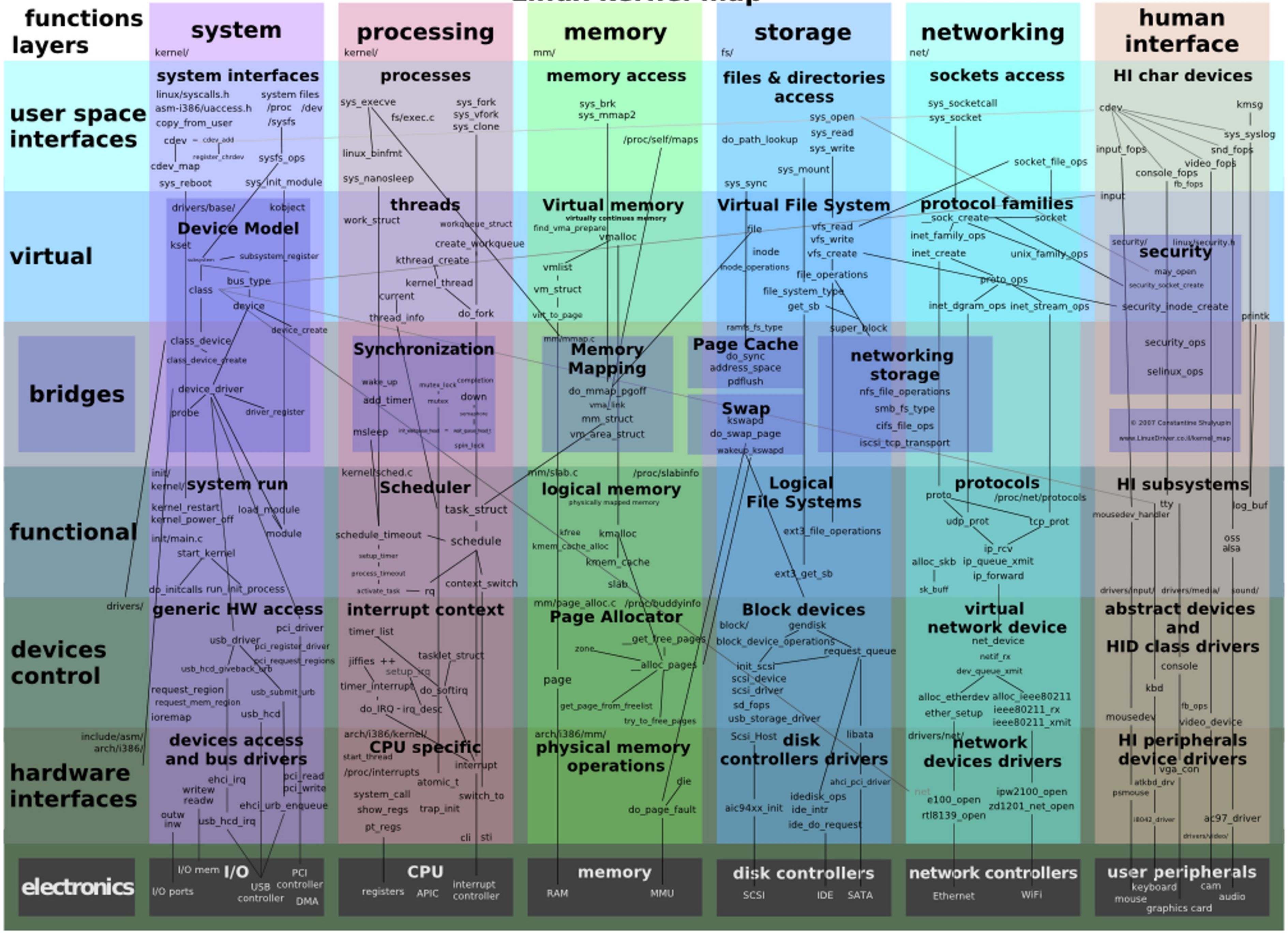
Do well in the class (i.e., get an A)

Ask questions and interact with the instructor during office hours

Three Easy Pieces

Operating systems are complex.

Linux kernel map



How to systematically approach studying it?

Three pieces: (1) Virtualization

- Make each application believe it has each hardware resource to itself

- Resources considered in this course: CPU and memory

Virtualizing CPU

- The system has a very large number of virtual CPUs.
- Turning a single CPU into a seemingly infinite number of CPUs.
- Allowing many programs to seemingly run at once, virtualizing the CPU

Virtualizing CPU

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <assert.h>
#include "common.h"

int
main(int argc, char *argv[])
{
        if (argc != 2) {
                fprintf(stderr, "usage: cpu <string>\n");
                exit(1);
        }
        char *str = argv[1];
        while (1) {
                Spin(1); // Repeatedly checks the time and returns once it has run for a second
                printf("%s\n", str);
        }
        return 0;
}
```

Simple Example(cpu.c): Code That Loops and Prints

Virtualizing CPU

Execution result 1.

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

Run forever; Only by pressing "Control-c" can we halt the program

Virtualizing CPU

Execution result 2.

```
prompt> ./cpu A & ; ./cpu B & ; ./cpu C & ; ./cpu D &
A
B
D
C
A
B
D
C
A
C
B
D
```

Even though we have only one processor, all four programs seem to be running at the same time!

Virtualizing Memory

- The physical memory is an array of bytes.
- A program keeps all of its data structures in memory.
- Read memory (load):
  - Specify an address to be able to access the data
- Write memory (store):
  - Specify the data to be written to the given address

Virtualizing Memory

```
# include < unistd.h >
# include < stdio.h >
# include < stdlib.h >
# include " common.h "

int main( int argc , char * argv [])
{
    int *p = malloc ( sizeof ( int ));  // a1: allocate some memory
    assert(p != NULL );
    printf ("(%d) address of p: %08x \ n",
               getpid (), (unsigned) p); // a2: print out the address of the memory

     * p = 0 ; // a3: put zero into the first slot of the memory
     while ( 1 ) {
         Spin( 1 );
         * p = *p + 1 ;
          printf ("(%d) p: %d \ n", getpid (), *p); // a4
      }
      return 0 ;
 }
```

gcc mem.c –o mem

Virtualizing Memory

```
prompt> ./mem
[1]  10
memory address of p: 00200000
p: 1
p: 2
p: 3
p: 4
p: 5
^C
```

The newly allocated memory is at address 00200000 .

Virtualizing Memory

```
prompt> ./mem &; ./mem &
[1]  10
[2]  11
memory address of p: 00200000
memory address of p: 00200000
(10)  p: 1
(11)  p: 1
(11)   p: 2
(10)  p: 2
(10)  p: 3
(10)   p: 3
ˆC
```

You may or may not get this output. (non-deterministic malloc()).

- It is as if each running program has its own private memory .
- Each program could allocate memory at the same address
- Each updates the value at address 00200000 independently.

Virtualizing Memory

- Each process accesses its own private virtual address space.
- The OS maps address space onto the physical memory.
- A memory reference within one running program does not affect the address space of other processes.
- Physical memory is a shared resource, managed by the OS.

Three pieces: (2) Concurrency

Concurrency: Events are occurring simultaneously and may interact with one another

OS must be able to handle concurrent events

Easier case

- Hide concurrency from independent processes

Trickier case

- Manage concurrency with interacting processes
  - Provide abstractions (locks, semaphores, condition variables, shared memory, critical sections) to processes
  - Ensure processes do not deadlock
  - Interacting threads must coordinate access to shared data

Concurrency

A Multi-threaded Program (thread.c)

```
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

volatile int counter = 0;
int loops;

void *worker(void *arg) {
        int i;
        for (i = 0; i < loops; i++) {
                counter++;
        }
        return NULL;
}

int
main(int argc, char *argv[])
{
        if (argc != 2) {
                fprintf(stderr, "usage: threads <value>\n");
                exit(1);
        }
```

Concurrency

```
        loops = atoi(argv[1]);
        pthread_t p1, p2;
        printf("Initial value : %d\n", counter);

        Pthread_create(&p1, NULL, worker, NULL);
        Pthread_create(&p2, NULL, worker, NULL);
        Pthread_join(p1, NULL);
        Pthread_join(p2, NULL);
        printf("Final value : %d\n", counter);
        return 0;
}
```

The main program creates two threads.

- Thread: a function running within the same memory space.
- Each thread start running in a routine called worker().
- worker(): increments a counter

Concurrency

Loops determines how many times each of the two workers will increment the shared counter in a loop.

- loops: 1000.

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

- loops: 100000.

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

Three pieces: (3) Persistence

Persistence: Access information permanently

- Lifetime of information is longer than lifetime of any one process
- Machine may be rebooted, machine may lose power or crash unexpectedly

Issues:

- Provide abstraction so applications do not know how data is stored : Files, directories (folders), links
- Correctness with unexpected failures
- Performance: disks are very slow; many optimizations needed!

Demo

- File system does work to ensure data updated correctly

Persistence

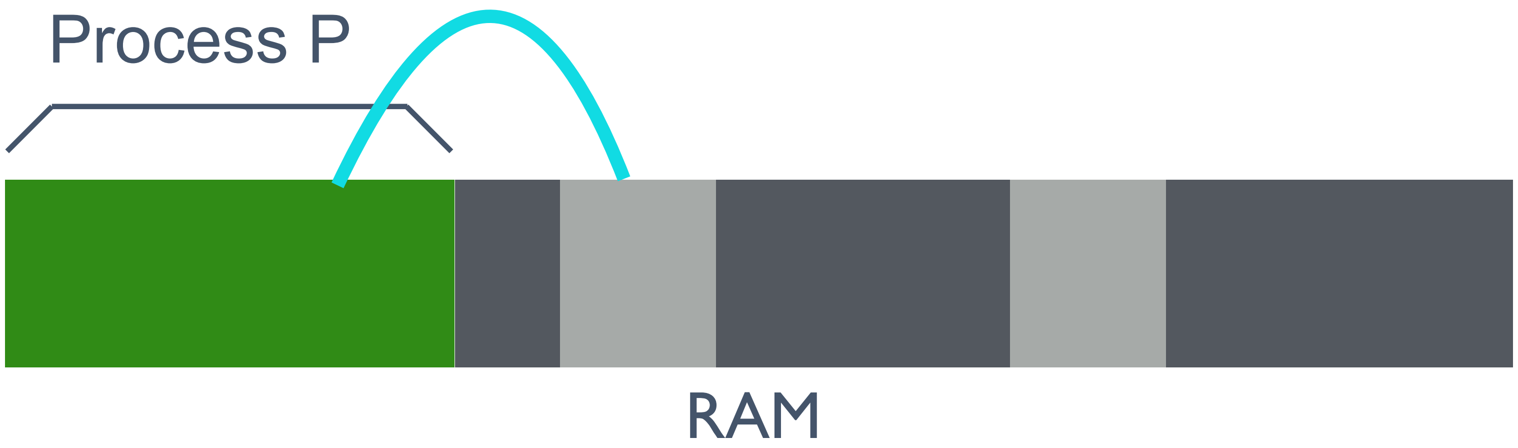
Create a file (`/tmp/file`) that contains the string "hello world"

```
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <fcntl.h>
#include <sys/types.h>

int
main(int argc, char *argv[])
{
        int fd = open("/tmp/file", O_WRONLY | O_CREAT
                      | O_TRUNC, S_IRWXU);
        assert(fd > -1);
        int rc = write(fd, "hello world\n", 13);
        assert(rc == 13);
        close(fd);
        return 0;
}
```

`open()`, `write()`, and `close()` system calls are routed to the part of OS called the file system, which handles the requests

System Call



movl $6, %eax;   int $64

syscall-table index

trap-table index

Persistence

What does the OS do to write to disk?

- Figure out where on disk this new data will reside
- Issue I/O requests to the underlying storage device
- File system handles system crashes during write

Journaling or copy-on-write

Carefully ordering writes to disk

Next lecture: CPU virtualization

Next steps

- Finish weekly quiz by next Tuesday 8 pm ET
- Look out for C self-assessment homework (not graded) and review
- Starting projects early helps the project grade significantly
- Sign up for class Piazza (link TBA on canvas announcement)
  - Ask questions well ahead of time
- Contact me if interested: independent study & research opp’s
- See you at next week’s lecture