RUTGERS UNIVERSITY
Computer Sciences Department

CS 416 Operating Systems Design

Sudarsun Kannan

Administrative

What to expect?

Goals:

- Understanding of OS and the OS/architecture interface/interaction
- Learn a little bit about distributed systems

Prerequisites:

- (CS 214) OR (ECE 331 AND ECE 351)

What to expect:

- We will cover core concepts and issues in lectures
- In recitations, you and your TA will mostly talk about the programming assignments
- 4 large programming assignments in C/C++; randomly chosen demos
- 1 midterm exam and 1 final exam (cumulative)

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

Grading?

Rough guideline

- Midterm 25%
- Final 25%
- Projects 40%
- Assignments and In-class Quiz 10% (Can be crucial, please show up)

Programming assignments can be done in groups of **2 or 3** or individually. Group members cannot be changed unless you have a very legitimate reason. Each member must clearly mention what they contributed

Demos might change grades. Exams and written homeworks are individual

Cheating policy: Collaboration at the level of ideas is good. Copying code or words is not good.

Grading?

- Project hand-ins MUST be on time
  - Late hand-ins will not be accepted
  - Programming assignments must be turned in electronically
    - Instructions will be posted on the web
    - We will promptly close the turn-in at the appointed time

Final Note About Grading

Things that I will not do at the end of the course:

- Give you an incomplete because you think that your grade was bad
- Give you extra work so that you can try to improve your grade
- Bump your grade up because you feel you deserve it
- Give you an F if your grade should actually be a D
- Review earlier assignments or exams to try to find extra points for you
- Give you a passing grade so that you can graduate
- Bump up your grade because you think you worked more than others

Bottom line: You get exactly the grade that your exam and assignment scores determine. If you want a good grade, the best approach is to take the course seriously from day one.

Warning!

Do NOT ignore these warnings!

We will be working on large programming assignments. If you do not have good programming skills or cannot work hard consistently on these assignments, don't take this course.

Cheating will be punished severely.

For more information on academic integrity, see:
http://www.cs.rutgers.edu/policies/academicintegrity/

You will learn a lot during this course, but you will have to work very hard to pass it!

Computing Resources

If you need computing resource, you can use iLab

If you already do not have an iLab account, then you can request one

See the below link for more details
https://www.cs.rutgers.edu/resources/instructional-lab

Communication and Office Hours

Questions on Projects - Email TAs

TA 1 – Yuji Ren (yr181@scarletmail.rutgers.edu)

TA 2 – Baber Khalid (baber.khalid@rutgers.edu)

Instructor's Office Hours – Monday, 10:30am to 12:30pm (after class)

Email queries: Subject line must start with [CS 416] when communicating with instructor or TAs

Please do not wait until the project or exam deadline day and ask your questions well ahead.

RUTGERS UNIVERSITY
Computer Sciences Department

CS 416 Operating Systems Design

Sudarsun Kannan

Introduction

Questions answered in this lecture:

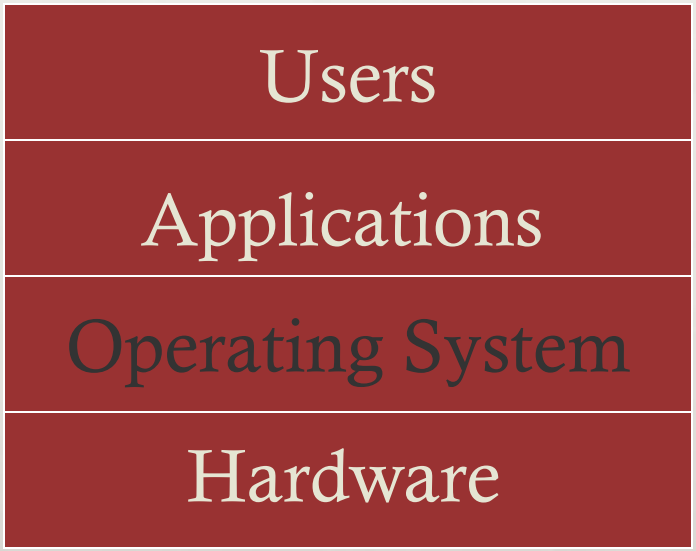
- What will you do in this course?
- What is an OS and why do you want one?
- Why study operating systems?

To do:

- Take a look at course web page for updates
  https://www.cs.rutgers.edu/~sk2113/course/cs416

What is an Operating System?

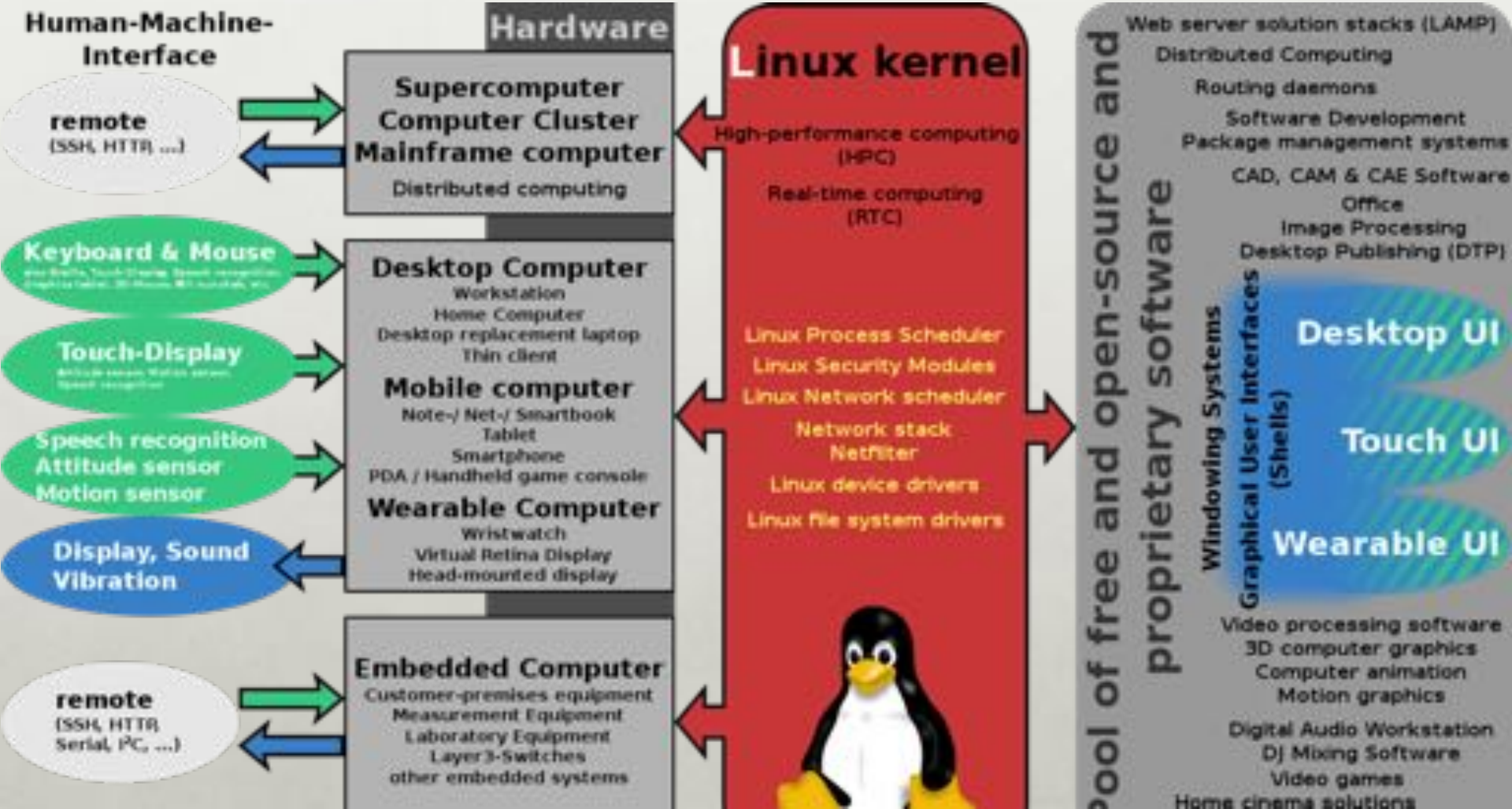
Not easy to define precisely…



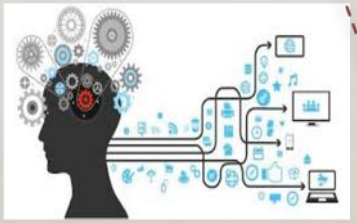
Operating System (OS):

Software that converts hardware into a useful form for applications

Where is OS Used?



Why study Operating Systems?



Lot of data

Process Fast

Use H/W efficiently

Reduce H/W and Operational Cost (e.g., Energy)

Why study Operating Systems?

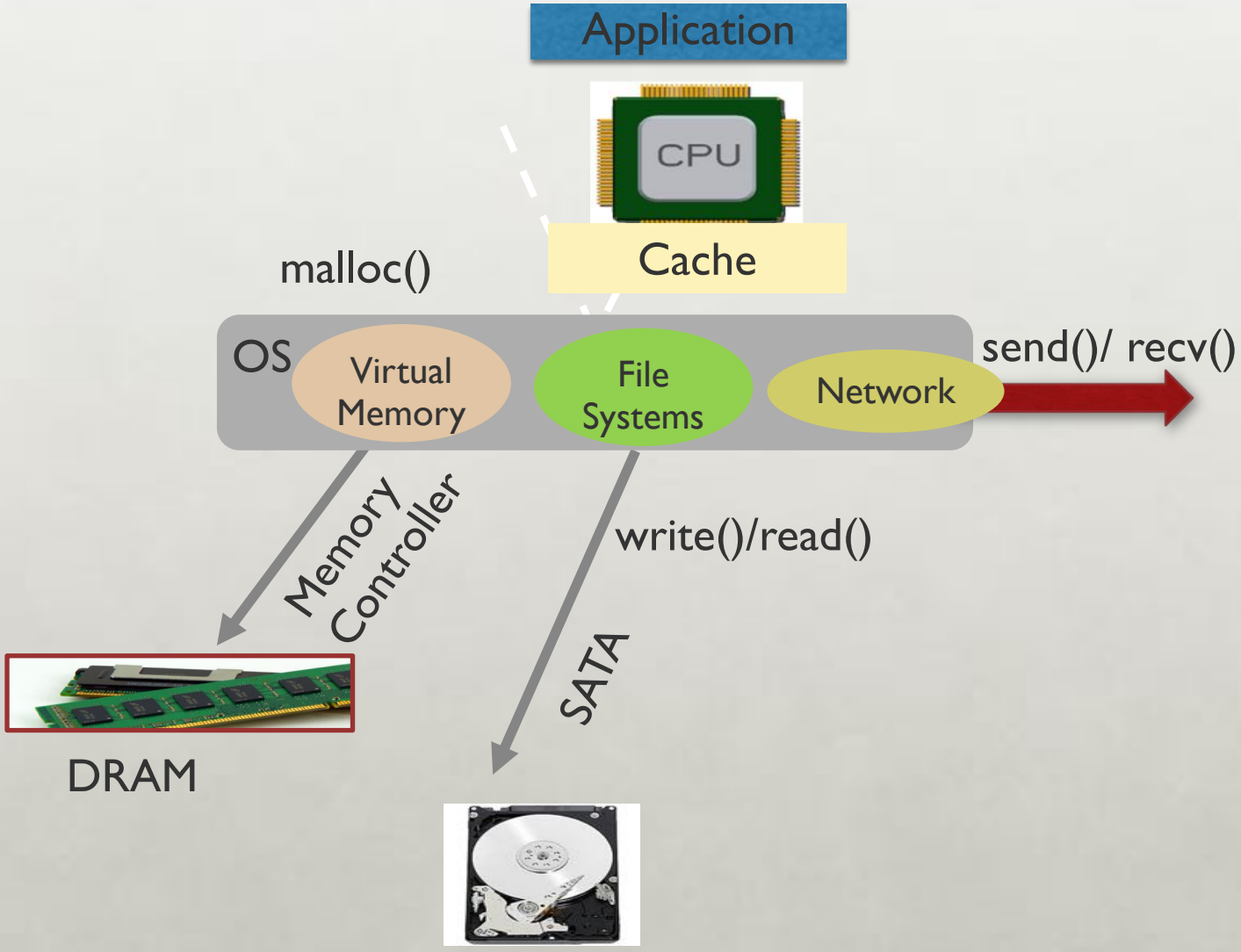
Build, modify, or administer an operating system

Understand system performance

- Behavior of OS impacts entire machine
- Tune workload performance
- Apply knowledge across many layers
  - Computer architecture, programming languages, data structures and algorithms, and performance modeling

Fun and challenging to understand large, complex systems

High-level View



What DOES OS Provide?

Role #1: Abstraction - Provide standard library for resources

What is a resource?

- Anything valuable (e.g., CPU, memory, disk)

What abstraction does modern OS typically provide for each resource?

- CPU:
  - process and/or thread
- Memory:
  - address space
- Disk:
  - 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 hardware should be exposed?

System Calls

- System call allows user to tell the OS what to do.
- The OS provides some interface (APIs, standard library).
- A typical OS exports a few hundred system calls.
- Run programs - Access memory - Access devices

What DOES OS PROVIDE?

Role #2: Resource management – Share resources well

Advantages of OS providing resource management?

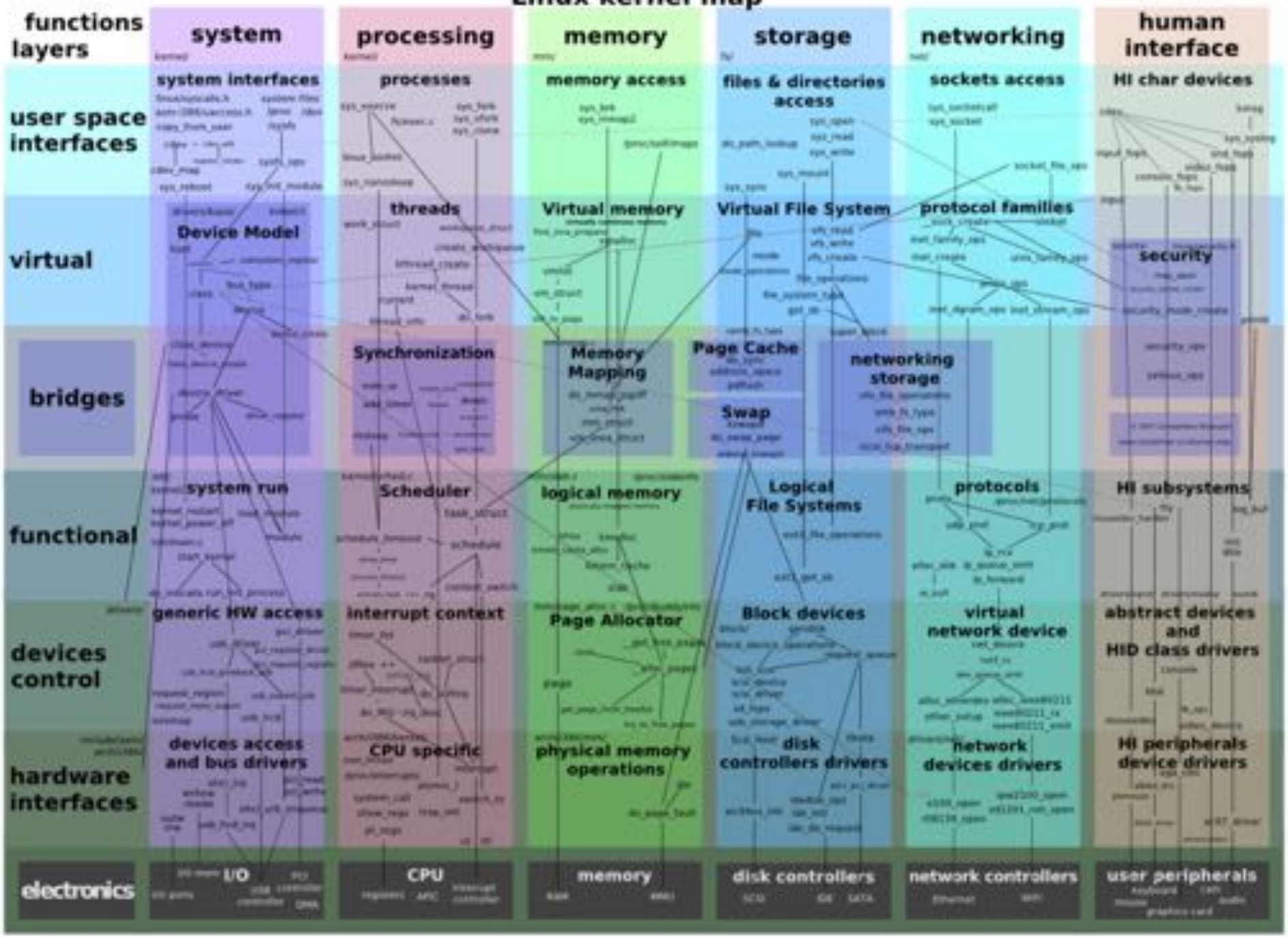
- Protect applications from one another
- Provide efficient access to resources (cost, time, energy)
- Provide fair access to resources

Challenges

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

OS Organization

- How to cover all the topics relevant to operating systems?


Linux kernel map
functions layers
system
processing
memory
storage
networking
human interface
user space interfaces
system interfaces
processes
memory access
files & directories access
sockets access
HI char devices
virtual
Device Model
threads
Virtual memory
Virtual File System
protocol families
security
bridges
Synchronization
Memory Mapping
Page Cache
Swap
networking storage
functional
system run
Scheduler
logical memory
Logical File Systems
protocols
HI subsystems
devices control
generic HW access
interrupt context
Page Allocator
Block devices
virtual network device
abstract devices and HID class drivers
hardware interfaces
devices access and bus drivers
CPU specific
physical memory operations
disk controllers drivers
network devices drivers
HI peripherals device drivers
electronics
I/O
CPU
memory
disk controllers
network controllers
user peripherals

Three PIECES: Virtualization

- Virtualization
  - Make each application believe it has each resource to itself
- Demo
  - Virtualize 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

```
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); // Repeatedly checks the time and
                             returns once it has run for a second
17                  printf("%s\n", str);
18          }
19          return 0;
20  }
```

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 of 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 o f 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
^C
```

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

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
^C
```

- It is as if each running program has its own private memory .
- Each program has allocated memory at the same address
- Each updates the value at 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: 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

Demo
Interacting threads must coordinate access to shared data

Three PIECES: CONCURRENCY

A Multi-threaded Program (thread.c)

```
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
16  int
17  main(int argc, char *argv[])
18  {
19      if (argc != 2) {
20          fprintf(stderr, "usage: threads <value>\n");
21          exit(1);
22      }
```

Three PIECES: CONCURRENCY

```
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      }
```

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

Three PIECES: CONCURRENCY

```
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      }
```

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

Three PIECES: 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: 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

Three PIECES: Persistence

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

```
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      }
```

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

Three PIECES: Persistence

What OS does in order 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

Advanced Topics

Current systems

- Multiprocessors
- Networked and distributed systems
- Virtual machines

To DO

Take a look at course web page

We will have a short self assessment assignment posted weekend.

The assessment will not be accounted for final grade