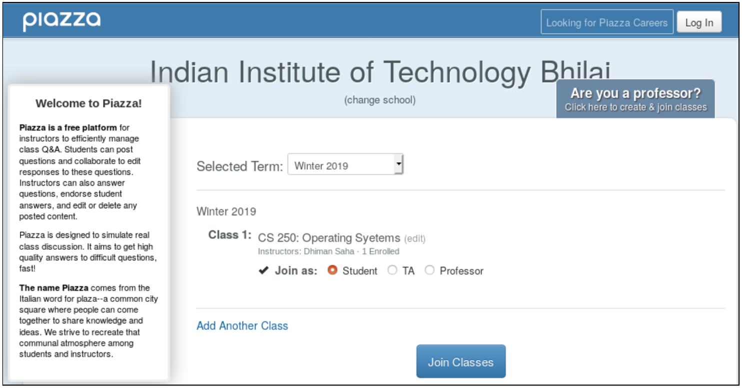
CS 250
OPERATING SYSTEMS

Instructor
Dr. Dhiman Saha

Image Source: Google

Course Link

piazza.com/iitbhilai.ac.in/winter2020/cs250


piazza
Looking for Piazza Careers
Log In

Indian Institute of Technology Bhilai

(change school)

Are you a professor?
Click here to create & join classes

Welcome to Piazza!

Piazza is a free platform for instructors to efficiently manage class Q&A. Students can post questions and collaborate to edit responses to these questions. Instructors can also answer questions, endorse student answers, and edit or delete any posted content.

Piazza is designed to simulate real class discussion. It aims to get high quality answers to difficult questions, fast!

The name Piazza comes from the Italian word for plaza--a common city square where people can come together to share knowledge and ideas. We strive to recreate that communal atmosphere among students and instructors.

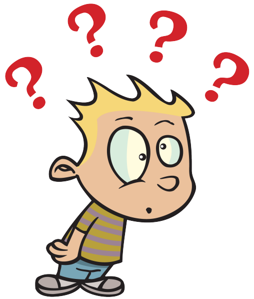
Selected Term: Winter 2019

Winter 2019

Class 1: CS 250: Operating Syetems (edit)

Instructors: Dhiman Saha · 1 Enrolled

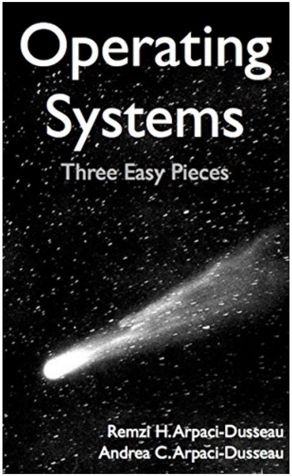
✓ **Join as:** [✓ Student] TA Professor



- OS?
- Why is it needed?
- Why should I study it?
- Is it difficult?
- Will I pass?

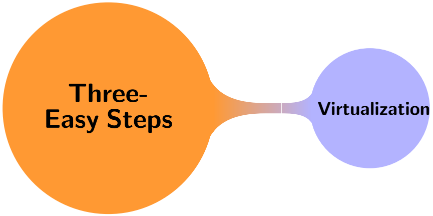
Image Credits: ComputerHope.com

The Book

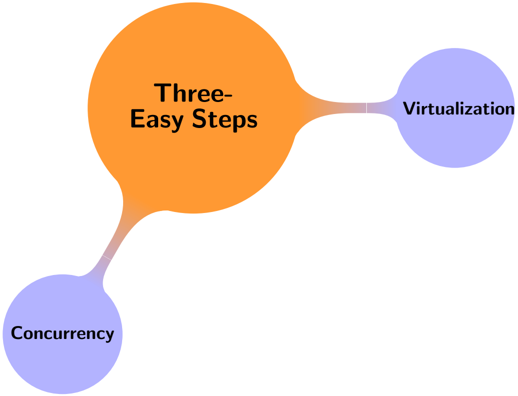


- 7 hard copies available at institute library
- Soft-copy (chapter-wise) available freely on book-site
- Unofficial consolidated version available on GitHub
- Personal Copy: `https://pothi.com/pothi/book/remzi-h-arpaci-dusseau-operatin`

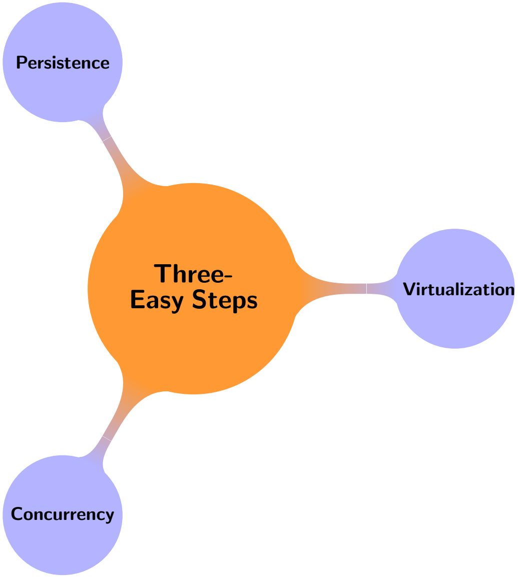
Course Plan



Course Plan



Course Plan


Persistence
Three-Easy Steps
Virtualization
Concurrency

What happens when a program runs?

The Von Neumann model of computing

- It executes instructions
- The processor fetches an instruction from memory
- Decodes it and
- Executes it
- Then processor moves to the next instruction
- Repeats this cycle until the program completes

Is this really that simple? Of course not!

- Lot of wild things will happen
- Primary motivation: ease of use

What happens when a program runs?

The Von Neumann model of computing

- It executes instructions
- The processor fetches an instruction from memory
- Decodes it and
- Executes it
- Then processor moves to the next instruction
- Repeats this cycle until the program completes

Is this really that simple? Of course not!

- Lot of wild things will happen
- Primary motivation: ease of use

What happens when a program runs?

The Von Neumann model of computing

- It executes instructions
- The processor fetches an instruction from memory
- Decodes it and
- Executes it
- Then processor moves to the next instruction
- Repeats this cycle until the program completes

Is this really that simple? Of course not!

- Lot of wild things will happen
- Primary motivation: ease of use

What happens when a program runs?

The Von Neumann model of computing

- It executes instructions
- The processor fetches an instruction from memory
- Decodes it and
- Executes it
- Then processor moves to the next instruction
- Repeats this cycle until the program completes

Is this really that simple? Of course not!

- Lot of wild things will happen
- Primary motivation: ease of use

What happens when a program runs?

The Von Neumann model of computing

- It executes instructions
- The processor fetches an instruction from memory
- Decodes it and
- Executes it
- Then processor moves to the next instruction
- Repeats this cycle until the program completes

Is this really that simple? **Of course not!**

- Lot of wild things will happen
- Primary motivation: ease of use

What happens when a program runs?

The Von Neumann model of computing

- It executes instructions
- The processor fetches an instruction from memory
- Decodes it and
- Executes it
- Then processor moves to the next instruction
- Repeats this cycle until the program completes

Is this really that simple? **Of course not!**

- Lot of wild things will happen
- Primary motivation: ease of use

The Middle-Man

- Middleware between user programs and system hardware

OS-CR Analogy Instructor ↔ CR ↔ Class

OS is a body of software To do what?

- To run programs easily
- Allowing programs to share memory
- Enabling programs to interact with devices
- So on and so forth.

- Also known as **supervisor** or the **master control program**

How does the OS do what is does?

The Notion of Virtualization

The OS takes a physical resource such as

- the processor or
- memory or
- a disk

and transforms it into a more general, powerful, and easy-to-use **virtual** *form of itself*

- Earns OS the alternative name as a **virtual machine**
- The OS provides a **standard library** to applications
- The OS acts as a **resource manager** fairly and efficiently managing resources like the CPU, memory or the disk.

How to virtulalize resources?

How does the operating system virtualize resources?

- What mechanisms and policies are implemented by the OS to attain virtualization?
- How does the OS do so efficiently?
- What hardware support is needed?

CS250 will answer these questions!

Virtualizing The CPU

Simple Example

```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);
        printf("%s\n", str);
    }
    return 0;
}
```

Code That Loops And Prints

Running a single program

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

Running Many Programs At Once

```
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
C
B
D
...
```

Turning a single CPU (or small set of them) into a seemingly infinite number of CPUs and thus allowing many programs to seemingly run at once is what we call **virtualizing** the CPU

Virtualizing The Memory

Simple Example

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

int
main(int argc, char *argv[])
{
    int *p = malloc(sizeof(int));                       // a1
    assert(p != NULL);
    printf("(%d) address pointed to by p: %p\n",
           getpid(), p);                                // a2
    *p = 0;                                             // a3
    while (1) {
        Spin(1);
        *p = *p + 1;
        printf("(%d) p: %d\n", getpid(), *p);           // a4
    }
    return 0;
}
```

A Program That Accesses Memory

Running The Memory Program Once

```
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
```

Running The Memory Program Multiple Times

```
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
...
```

Illusion Vs Reality

Illusion

As far as the running program is concerned, it has physical memory all to itself.

Reality

Physical memory is a shared resource, managed by the operating system.

OS is virtualizing memory

- Each process accesses its own **private virtual address space**, which the OS **somehow** maps onto the physical memory of the machine.
- A memory reference within one running program does not affect the address space of other processes (or the OS itself)

Somehow?

CS250 will answer these questions!

Concurrency

Problems that arise, and must be addressed, when working on many things at once (i.e., concurrently) in the same program.

Details deferred

When there are many concurrently executing **threads** within the same memory space:

- How can we build a correctly working program?
- What primitives are needed from the OS?
- What mechanisms should be provided by the hardware?
- How can we use them to solve the problems of concurrency?

Persistence

- What happens to the data when you switch-off your system?

How to store data persistently?

The **file system** is the part of the OS in charge of managing persistent data.

- What techniques are needed to do so correctly?
- What mechanisms and policies are required to do so with high performance?
- How is reliability achieved, in the face of failures in hardware and software?

Design Goals

- Convenience
- **Abstraction** of hardware resources for user programs
- Efficiency of usage of CPU, memory, etc.
- Isolation between multiple processes

Evolution

- Early OSes: Act as a library to provide common functionality across programs
- Later: Go beyond libraries - offer protection
- Evolution from **procedure call** to **system call**.

Procedure call Vs System call

- Can you tell the difference between these?

- When a system call is made to run OS code, the CPU executes at a **higher privilege level**
- The era of multiprogramming - Evolved from running a single program to multiple processes concurrently
- The modern era: Windows/Mac/Linux