# CS-446/646
Course Introduction
Operating Systems Primer

C. Papachristos
Robotic Workers (RoboWork) Lab
University of Nevada, Reno

## Your Instructor

**Christos Papachristos**

Assistant Professor

Department of Computer Science & Engineering

- Areas of Activity:

  Robotic Workers Lab (https://www.roboticworkerslab.com/) - (RoboWork)

  **Research / Coding / SW Engineering** for Field Robotics

- Education

  **MS** in Electrical & Computer Engineering

  **PhD** in Autonomous (Aerial) Robotics

- Find me at:

  **WPEB - 309** (Office) or **WPEB – 316/302** (the RoboWork Lab)

  cpapachristos@unr.edu , papachric@gmail.com

# Course Rules

## Grading Policy:

| Component | Percentage |
|---|---|
| Homework Assignments (Undergraduate) | 30% |
| Homework Assignments (Graduate) | 25% |
| Presentations (Graduate-only) | 5% |
| Midterm Exam | 30% |
| Final Exam | 40% |

You cannot earn a **C** in the course without having earned a **C** on both:

- ➢ The *weighted average* of the Midterm and Final exams.
- ➢ The *weighted average* of the Homework Assignments (& Grad Presentations if applicable).

Plus/Minus grading will be assigned as indicated in the Syllabus.
Grade re-scaling may be assigned based on an outstanding or inferior Final exam.

*Note:* Also, you cannot earn a passing grade (**D-**) in the course without a passing grade on both categories.

Class participation will be factored in. As per the University Administrative Manual (UAM) 3,020, students are expected to attend classes in which they are enrolled.

## Course Rules

### Tasks & Responsibilities:

**Weekly / Bi-Weekly Homework Assignments**

Turn in via WebCampus! LATE submission policy:

- 1 day late: 15% penalty
- 2 days late: 30% penalty
- 3 days late: 60% penalty
- >3 days : Zero credit

### Academic Dishonesty:

Cheating, plagiarism or otherwise obtaining grades under false pretenses constitute academic dishonesty according to the code of this university. Academic dishonesty will not be tolerated and penalties can include filing a final grade of "F"; reducing the student's final course grade one or two full grade points; awarding a failing mark on the coursework in question; or requiring the student to retake or resubmit the coursework. For more details, see the University of Nevada, Reno General Catalog.

(Also, refer to Academic Standards in course syllabus and online)

## Course Rules

Tasks & Responsibilities:

**Weekly / Bi-Weekly Homework Assignments**

Turn in via WebCampus! LATE submission policy:

- 1 day late: 15% penalty
- 2 days late: 30% penalty
- 3 days late: 60% penalty
- >3 days : Zero credit

### Academic Dishonesty:

*Note*:

There exist widely accessible and reliable tools to cross-compare you code:

*vs* your classmates' code !

*vs* students' code from previous semesters !

Semantic analysis and comparison means IT WILL CATCH similarly structured code, even if you rename your variables / change indentation / etc. !

# Course Rules

## Academic Standards Policy for Writing Code – CSE Department:

- ➢ Sharing ideas with other students is fine, but you should write your own code. Never copy or read other students' code, including code from previous years. Cosmetic changes, such as rewriting comments, changing variable names, and so forth to disguise the fact that your work is copied from someone else, are easy to detect and not allowed.
- ➢ It is your responsibility to keep your code private. Sharing your code in public is prohibited, and may result in zero credit for the assignment.
- ➢ If you find some external code (such as an open-sourced project) that could be reused as part of your assignment, you should first contact the instructor to see whether it is fine to reuse it. If the instructor permits it, she/he may announce it to the entire class so that all students could use it. And if you decide to reuse the external code, you should clearly cite it in comments and keep the original copyright in your code, if applicable.
- ➢ You should be prepared to explain any code you submit, including code copied/modified from external sources.
- ➢ Every student turning in a submission is assumed to be signing the following statement:
  "This code is my own work. It was written without consulting a tutor or code written by other students."

## Course Rules

### Generative AI use is NOT allowed for any purpose :

- For the purposes of this course, any and all uses of generative artificial intelligence (AI)/large language model tools (such as ChatGPT, DALL-E, Gemini, Microsoft Copilot, etc.) will be considered a violation of the UNR Academic Integrity Policy (UAM 6,502), specifically the prohibition against cheating or submitting work that is not your own.
- This applies to all assessments in the course, including case studies, written assignments, discussions, quizzes, exams, and problem sets.
- The following actions are prohibited:
  - Submitting any part or all of an assignment statement or test questions as part of a prompt to a large language model AI tool.
  - Incorporating any part of an AI-written response into a submitted assignment or assignment component.
  - Using AI to summarize or contextualize reading assignments or source materials.
  - Submitting your own work for this class to a large language model AI tool for iteration or improvement.

## Course Objectives

Theoretical Understanding of:

- Core **Operating Systems (OS)** Principles
  - **Virtualization** : Processes, Scheduling, Virtual Memory, Paging, Dynamic Memory
  - **Concurrency** : Threads, Synchronization, Semaphores & Monitors
  - **Persistence** : I/O, Disks, File Systems
  - **Networking** : Protocols, Network File System
  - **Security** : Authentication, Access Control
  - **Virtual Machines** : Virtual Machine Monitors, HW-Assisted Virtualization

Gain a holistic understanding of the big picture:

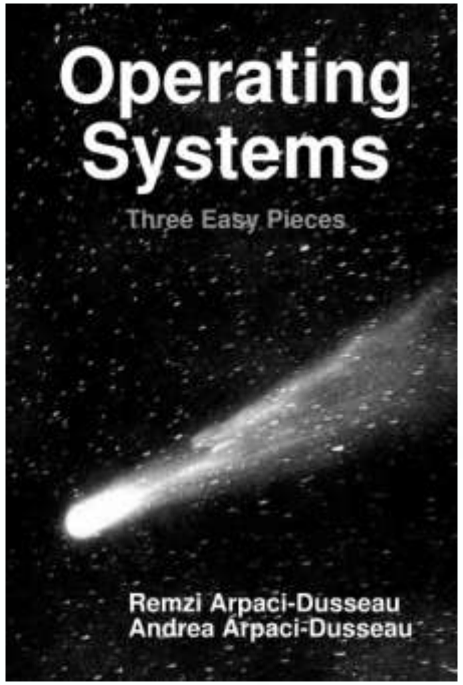
- How do hardware/architecture, programming language, compiler, algorithms, OS work together.

Coding in C / Bash Scripting – level application of relevant concepts

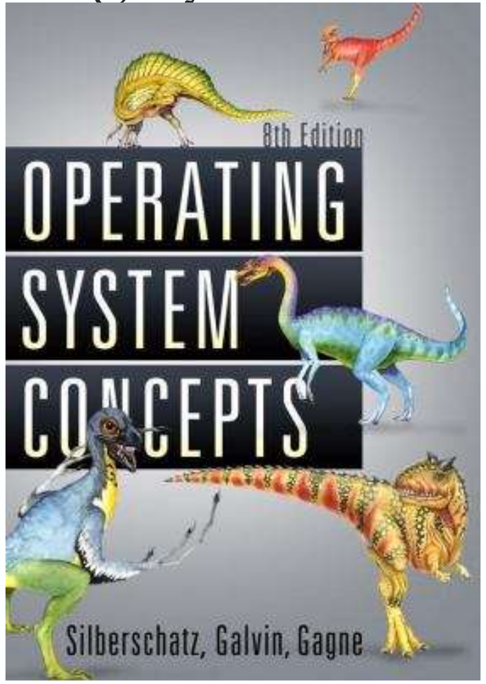
Learn some transferrable skills

## Course Resources

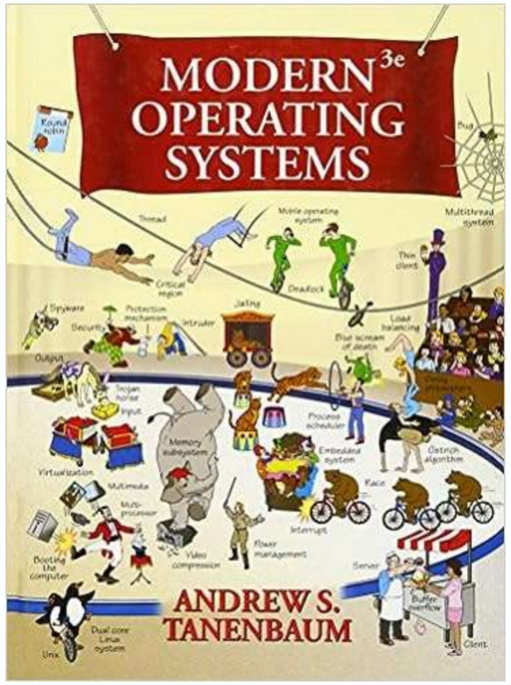
Operating Systems – Three Easy Pieces



Operating System Concepts



Modern Operating Systems



Also, Webcampus - *Slides developed following content of*

*i) Stanford CS 140 by David Mazières,*

*ii) Johns Hopkin's CS 318 by Ryan Huang,*

*iii) Columbia Engineering W4118 by Junfeng Yang*

## Course Help

### Course & Projects:

Teaching Assistants:

Murugappan Muthuganesan
mmuthuganesan@ [unr.edu]

### Disability Services:

Any student with a disability needing academic adjustments or accommodations is requested to speak with the Disability Resource Center as soon as possible to arrange for appropriate accommodations.

## Challenges

Get comfortable with:

The Linux environment. All code must compile and run on Ubuntu 18.04.

Start with your Homework Assignments early on !

**(X)Ubuntu** systems available at the **ECC** (SEM 231) and **WPEB Labs**.

To access Ubuntu on one of those machines, you have to hard restart the computer by pushing the button on the tower, and then select CS135 from the available partition options.

Don't wait to seek help. Benefit from all sources.

Your TA(s).

Your Instructor.

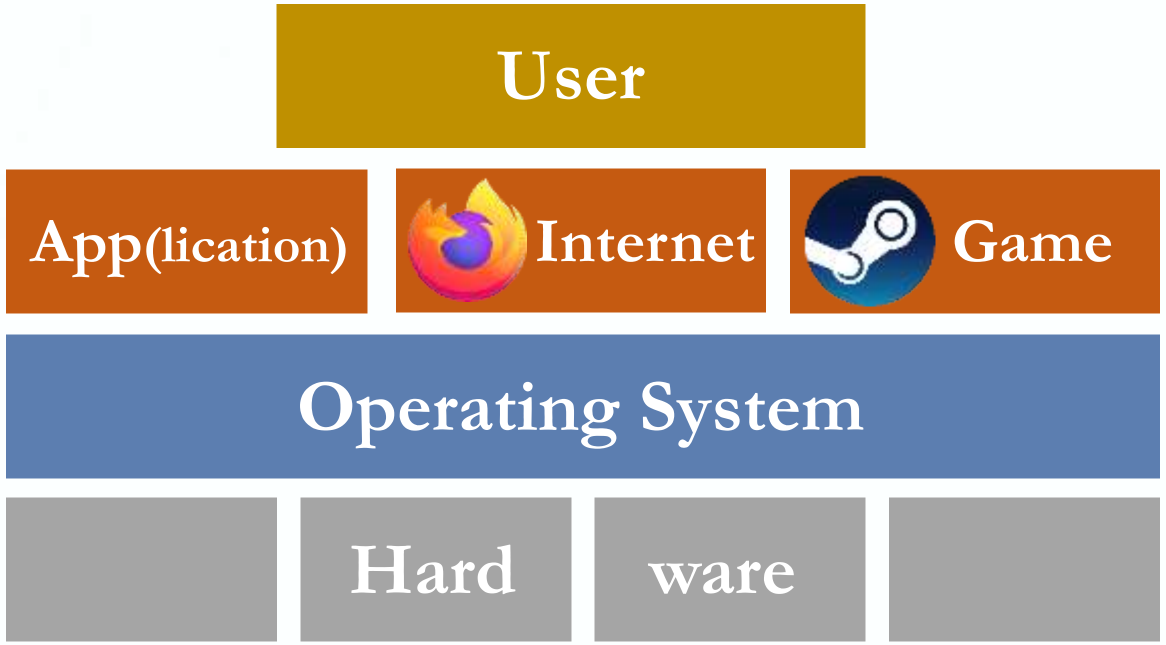
WebCampus material (Lectures, Samples, etc.) & Discussions.

# Operating System Basics

What is an Operating System (OS)

Layer between *Applications* and *Hardware*

- *Middleware*



I.e. all the foundation on which you can develop your programs/apps.

# Operating System Basics

## OS and Hardware

Management of Hardware *Resources*

*Computation*

*Volatile Storage*

*Persistent Storage*

*Communication*

*I/O*

Provides *Abstractions* to hide details of hardware from applications:

- Processes, Threads
- Virtual Memory
- File Systems
- …

## Operating System Basics

OS and Hardware

Mediates *Access* from different applications

- Who has access at which point and for how long

Why?

Provides benefits applications by offering:

- Simplicity (no need to tweak Device *Registers* at *User Level*)
- Device Independence (all Networking cards "look" the same)
- Portability (across Win95/-Win10 versions)

## Operating System Basics

OS and Applications

***Virtual Machine*** interface

- Each *Program* thinks it owns the entire computer / system *Resources*

Provides ***Protection***

- Prevents *Processes (/Users)* from clobbering one another

Provides ***Sharing***

- *Concurrent* execution of multiple *Programs* (Time Slicing)
- Communication among multiple *Programs* (Pipes, Cut & Paste)
- Shared implementations of common *Resources* & *Facilities* (e.g. *File System*)

# Operating System Basics

## Different OSs

Numerous popular OS options exist

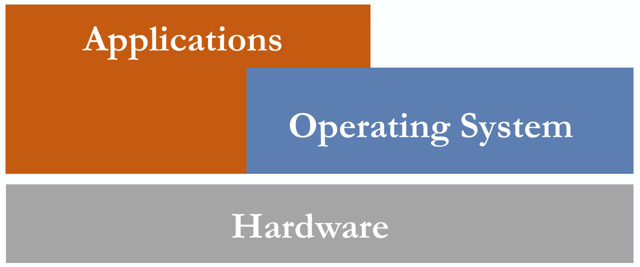
- Free / Commercial
- Open-source / Closed-source
- Embedded OSs
- Real-Time OSs
- Different capabilities (e.g. UNIX named pipes)
- Different Security architectures

## Operating System Basics

Requirements for a functioning Operating System

A primitive OS idea:

- e.g. an implementation as a library of standard services



Issues:

- Support for single-*Program* execution at a time
- Waiting until *Program* execution makes poor utilization of *Resources* (CPU and other hardware, even the *User*'s engagement time)
- No accounting for malicious *Programs* / *Users*

## Operating System Overview

**Multitasking**

The ability for more than once *Processes* to be executing (running) at the same time

*Note: Concurrently* does *NOT (necessarily)* mean *in parallel*

- When one *Process* "*Blocks*" (waits for disk/network/user input), run another *Process*

Mechanism: ***Context-Switching***

- When a *Process* resumes, it can continue from its last execution point (*State*)

Requirements:

- ***Scheduling***

  Avoid *Processes* going into infinite loop and never relinquishing CPU

  - *Limiting:* Fair sharing, take CPU away from looping *Process*

- ***Virtual Memory***

  Avoid *Processes* scribbling over each other's data

  - *Isolation:* Protect one *Processes' Memory* from one another

# Operating System Overview

**Multitasking**

The ability for more than once *Processes* to be executing (running) at the same time

*Note: Concurrently* does *NOT (necessarily)* mean *in parallel*

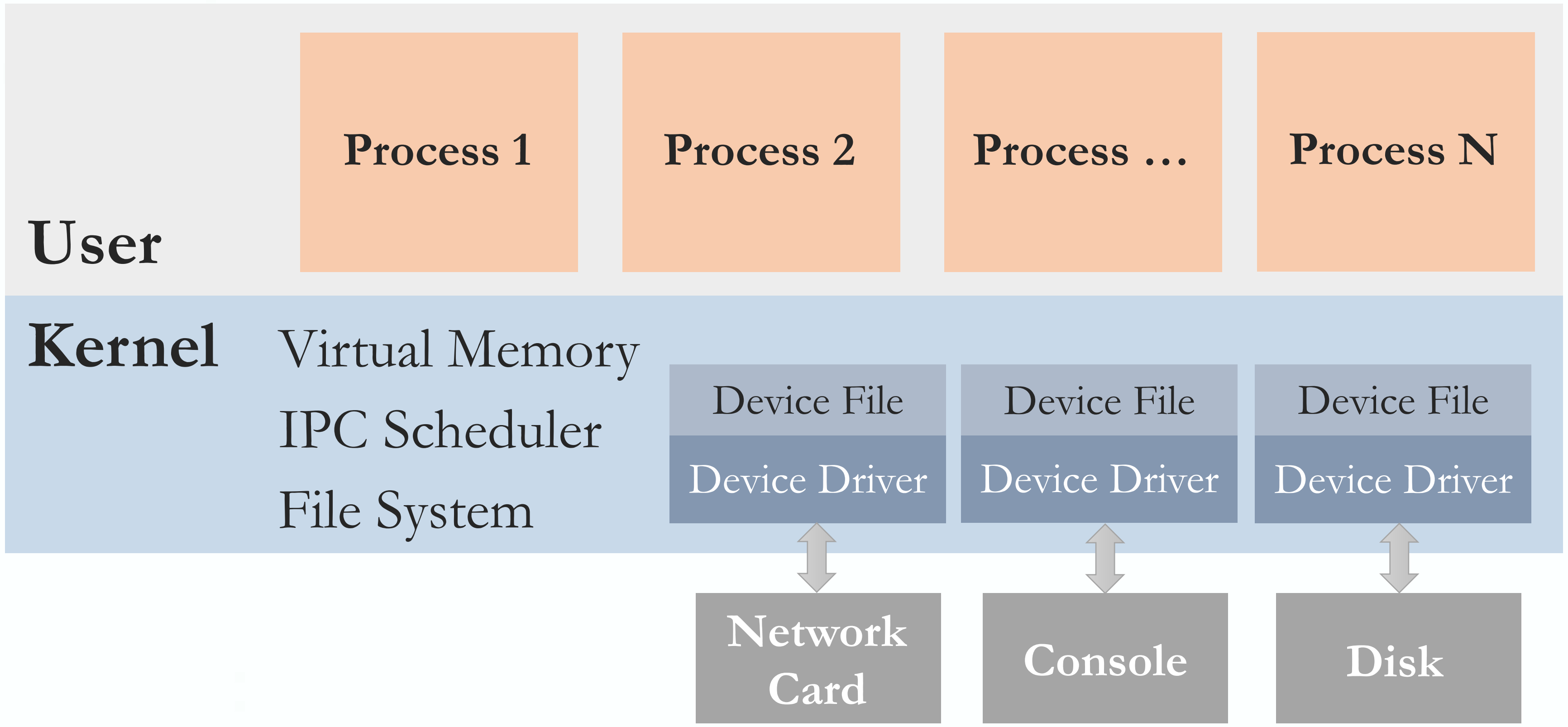
- When one *Process* "*Blocks*" (waits for disk/network/user input), run another *Process*

Mechanism: ***Context-Switching***

- In a traditional CPU each *Process* utilizes the CPU *Registers* to store data and hold the current *State*. In a *Multitasking* OS, the Operating System "*Context Switches*" between *Processes* or *Threads* to allow the execution of multiple *Processes* simultaneously. For every switch, it saves the *State* of the current *Process*, followed by loading the next *Process*' *State*, which will run on the CPU.
- *Context Switches* are usually computationally intensive; *Context Switching* from one *Process* to another requires time for doing the administration (saving and loading *Registers* and *Memory Maps*, updating various tables and lists, etc.)
- What is actually involved in a *Context Switch* depends on the *Architectures*, the OS-specific design, and the number of *Resources* shared (e.g. threads of the same *Process* share many *Resources* compared to unrelated non-cooperating *Processes*).

# Operating System Overview

**OS Structure**



- Most software runs as a **User-Level** *Process*
- OS Kernel runs in **Privileged mode**

# Operating System Overview

***User Mode*** (Unprivileged mode, Restricted mode, or Slave mode)
(User-Mode "bit": 1)

- No direct access to system *Resources*; in order to access the *Resources*, a "*System Call*" must be made
- If an *Interrupt* occurs, a single *Process* can fail
- All *Processes* get separate *Virtual Address Spaces* by the OS
- Only *Memory* allocated for *User Mode* can be referenced
- Not all CPU *Instructions* can be directly executed on their own, or reference any *Memory Block*

***Kernel Mode*** (Privileged mode, System mode, or Master mode)
(User-Mode "bit": 0)

- Direct & unrestricted access to system *Resources*
- If an *Interrupt* occurs, the entire system might go down (become unrecoverable)
- All *Processes* share a single *Virtual Address Space*
- *Memory* allocated for both *Kernel* & *User Mode* can be referenced
- CPU *Instructions* can be directly executed, and any Memory *Block* is allowed to be referenced

*Note:* E.g., **CS** *Register*'s CurrentPrivilegeLevel is 2-bit field corresponding to privileges 0 (highest)-3(lowest)
Linux only uses levels 0 & 3, respectively called *Kernel Mode* & *User Mode*

# Operating System Overview

***System Call*** ( *syscall* )

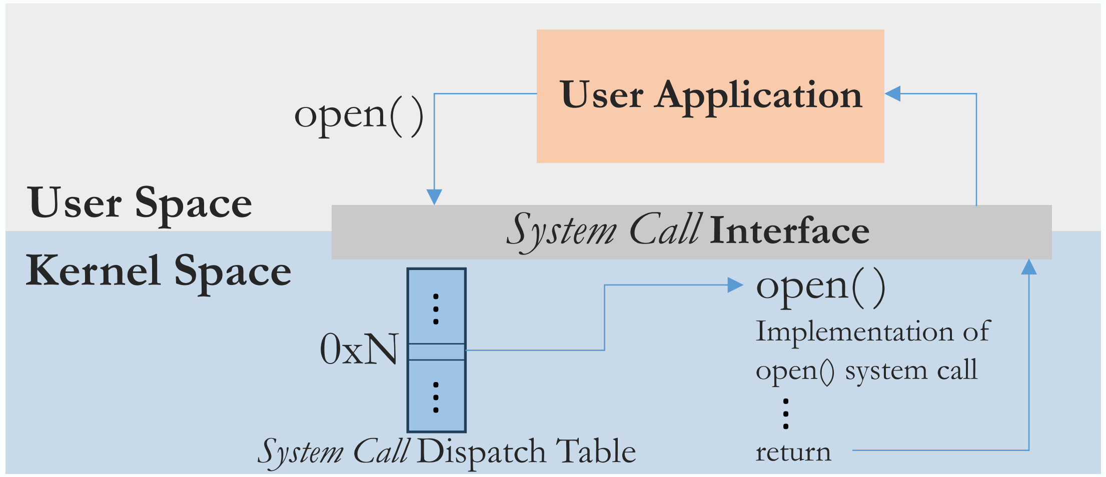
Applications can invoke the OS Kernel through *System Calls*

- Through a special CPU *Instruction* that serves to transfer control to the Kernel
- Subsequently, *System Call Handlers* are dispatched
  - (and in most Unix-like systems are processed in *Kernel Mode*)

*Note:* In most OSs, *no Context Switch* is necessary (but a Privilege Context Switch occurs)

i.e. OS sets/unsets User Mode "bit"

```
#include <fcntl.h>
#include <unistd.h>
int main()
{
 int fd = open("cs318.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
 if (fd < 0) {
  write(2, "Failed to open cs318.txt\n", 25);
  _exit(1);
 }
 write(fd, "Hello, OS!\n", 11);
 close(fd);
 return 0;
}
```



*Note:* User Mode **open()** ≠ Kernel Mode **open()** *System Call* implementation

# Operating System Overview

***System Call*** ( *syscall* )
The only way for an application to invoke OS services

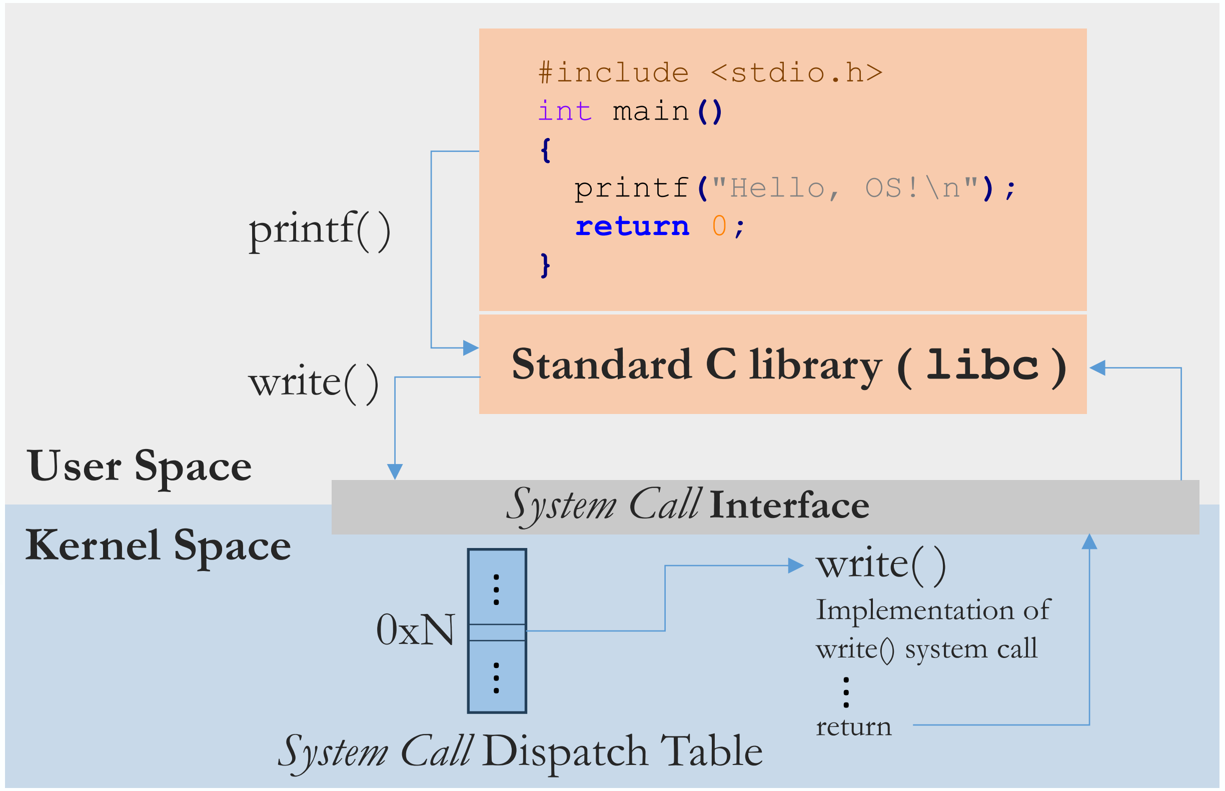
*Goal:* Do things application isn't allowed to do in *User Mode* (Unprivileged)
- ➢ Like a library call, but "taps" into more Privileged Kernel code

Kernel supplies well-defined *System Call* Interface
- ➢ Application sets up *System Call* arguments and "*Traps* to Kernel "
- ➢ Kernel performs operation and returns result

*Note:* (from *A Commentary on the 6th Edition Unix OS - 9 Hardware Interrupts and Traps*):
"**`trap`**" *Instructions* may be deliberately inserted in user mode programs to catch the attention of the Operating System with a request to perform a specified service. This mechanism is used as part of the facility known as "*System Calls*". (http://warsus.github.io/lions-/lionc/sect0116.html)

# Operating System Overview

***System Call*** ( *syscall* )



```
#include <stdio.h>
int main()
{
  printf("Hello, OS!\n");
  return 0;
}
```

C Standard Library calls are built on *System Calls*

# Operating System Overview

***Trap* / Software *Interrupt*** ("*Interrupts*" (of all types) sometimes also referred to as "*Exceptions*")

***System Call*** ( *syscall* ):

- Made by a *User Level* program to the OS to perform a Privileged operation (read/write to file, allocate *Memory*, etc.)
- To make a *syscall*, the program executes a special *Instruction* (e.g. **`int 0x2E`** / **`syscall`**) that triggers a ***Software Interrupt***
- OS transfers control to *Kernel Mode* – ***Traps to Kernel*** – and executes a *System Call Handler*, which checks the type of *System Call* and takes appropriate action (read from *File/Resource*, allocate *Memory*, etc.)

***Trap*** :

- *Synchronous* (from events internal to CPU) *Interrupt* generated by the CPU when a *User Level* program attempts to execute a *Privileged Instruction* or encounters an error (e.g. divide by zero, stack overflow in PDP-11 architectures, etc.)
- When a *Trap* occurs, the CPU transfers control to Kernel and executes a *Trap Handler*, which checks the type of *Trap* and takes appropriate action (call **`SIGFPE`**, terminate program, perform Privileged operation on behalf of it, etc.)

Both *System Calls* & *Traps* involve transferring control to the Kernel to perform Privileged operations, but:

- *"Traps"* are usually generated automatically by the CPU when a program encounters an error or attempts to execute a Privileged *Instruction*
- *"System Calls"* are initiated by the Program itself to request Privileged operations

## CS-446/646

Time for Questions !