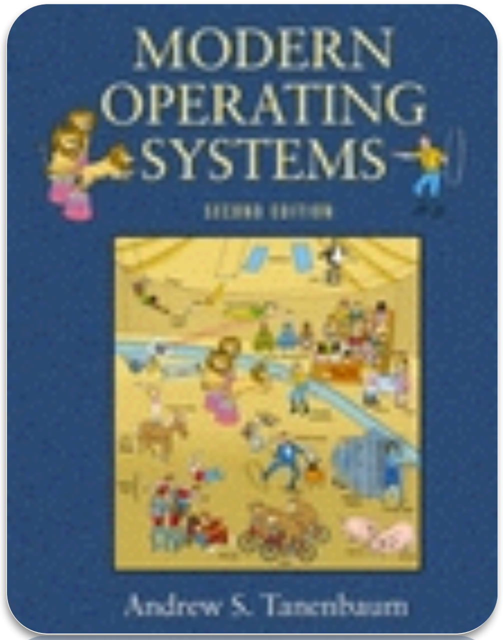
CSE 313
Operating System

Lecture By: Rezwana Reaz

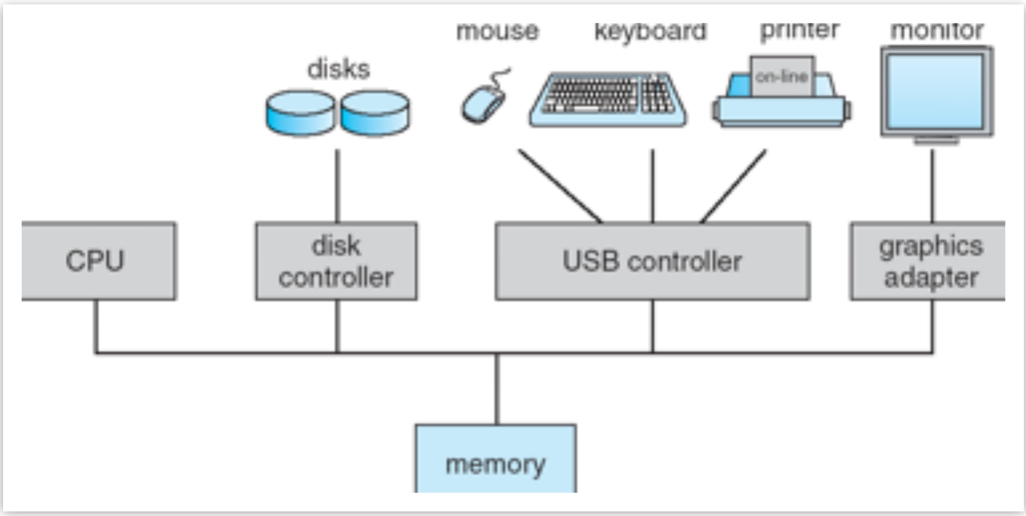
Textbooks

- Modern Operating Systems
  - Andrew S. Tanenbaum
  - 4th Edition or Newer



Modern Computer System

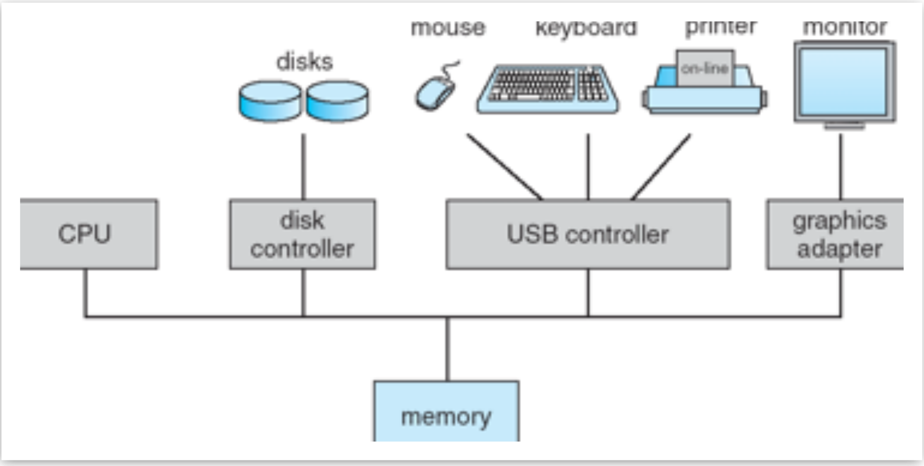
- Complex system
  - Many H/W components
  - Many programs running simultaneously



- Each running program
  - Executed in the CPU
  - Resides in the Memory
  - May Interact with several devices

Modern Computer System

- To ensure the correct operation someone needs to
  - Understand how all these components work
  - Manage them wisely
  - Allocate them efficiently



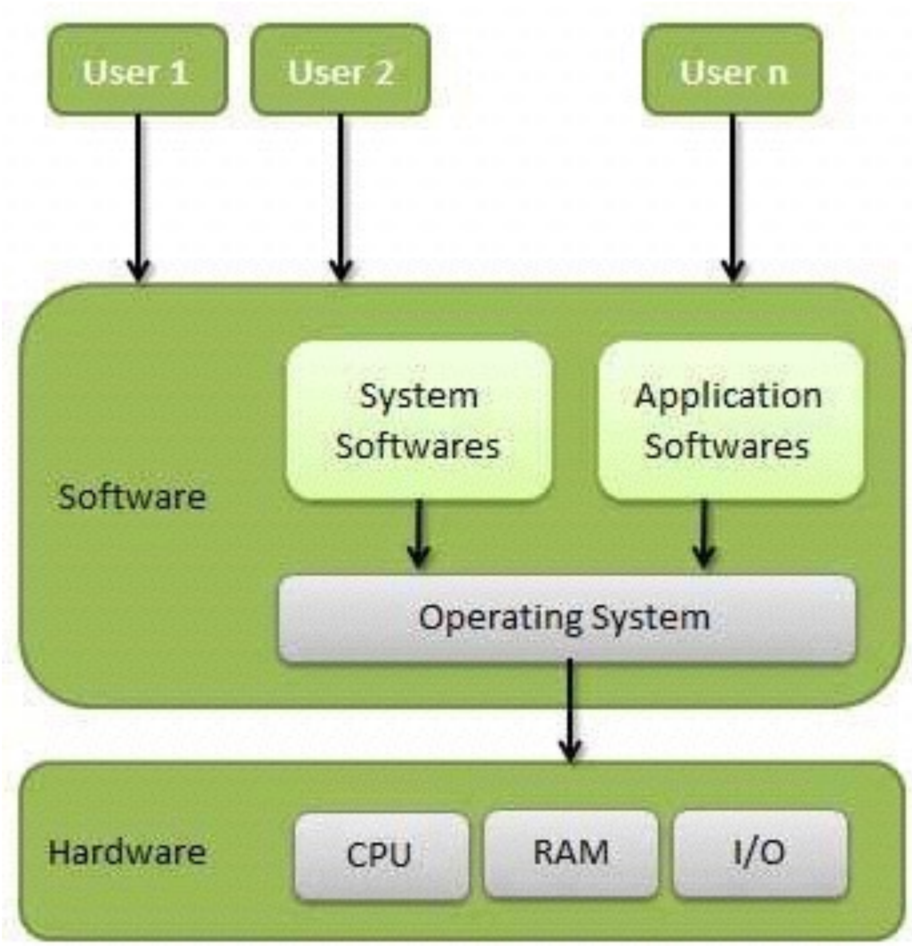
- **A very challenging task!!!**

- If application programmer had to consider everything
  - No code would ever get written

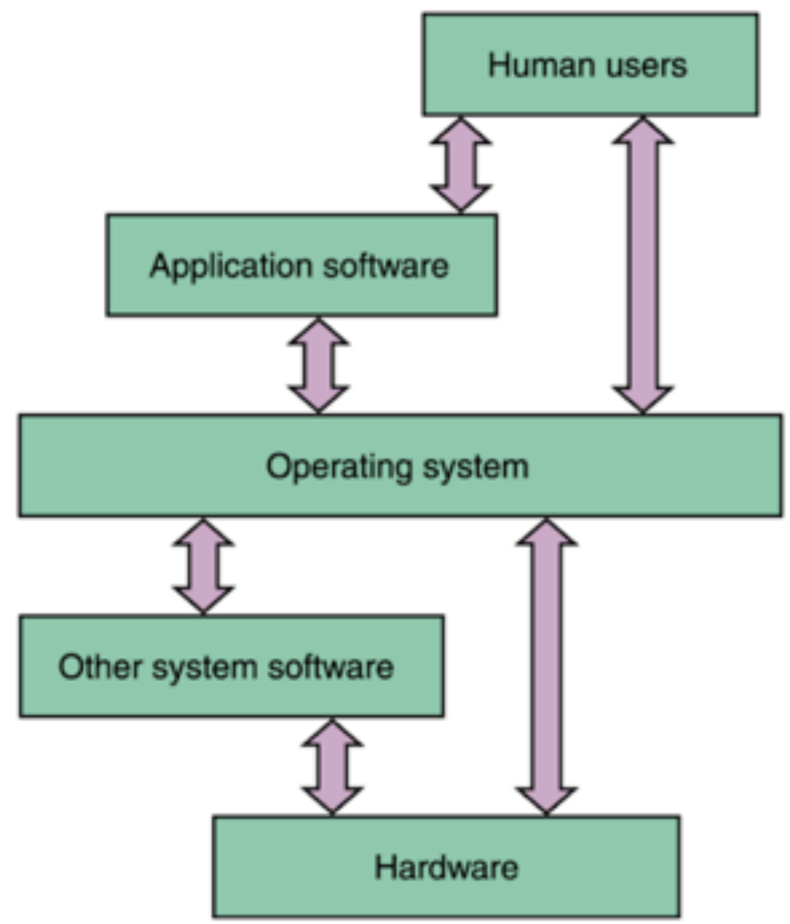
- So, computers are equipped with a **special software**
  - THE OPERATING SYSTEM

What is an OS?

- Operating System is nothing but a software



Placement of OS



Scope of Interaction

Some Common Definitions of OS

- An operating system (OS) is the program that, after being initially loaded into the computer by a boot program, manages all of the other application programs in a computer.
- An operating system (OS) is system software that manages computer hardware and software resources, and provides common services for computer programs
- An operating system (OS) is a software program that serves as the fundamental interface between computer hardware and the user.

OS as An Extended Machine

- Dealing with hardware can be very difficult.
  - Need to understand the operation
  - Need low level programming
  - **Consider direct read write to hard disk**
- Programmers want a simple, high-level abstraction to deal with hardware.
- The job of the OS is to create good abstractions
- Example:
  - OS abstracts hard disk by the file-system

OS as A Resource Manager

- Allows multiple programs to run at the same time

- Provide an orderly and controlled allocation of
  - Processors
  - Memories
  - I/0 devices

- Among the competing programs

- Example:
  - 3 programs running on some computer
  - All tried to print their output simultaneously on the same printer
  - The OS can bring order to the potential chaos by buffering all the output on the disk

Resource Management

- Resources can be shared in two different ways:
  - **Time multiplexing:**
    - In a single CPU system, several running programs take turns.
  - **Space multiplexing:**
    - main memory is divided up among several running programs

Booting the Computer

- OS loads every other program into RAM for running
- So OS has to load into RAM before any other programs
- But how to load the OS into RAM after power-on?
- The procedure of starting a computer by loading the OS into RAM is known as ***booting the system***

Booting the Computer

- Motherboard includes a ***non-volatile*** ROM chip
- The ROM chip is shipped with a **start up program** referred to as the BIOS (***Basic Input/Output System)***

Booting the Computer

- When the computer is booted, ***BIOS Program is started***
- BIOS first runs ***diagnostics*** to determine the state of the machine
    - checks how much RAM is installed and
    - checks whether the keyboard and other basic devices are installed and responding correctly
    - …

Booting the Computer

- BIOS then determines the ***boot device*** by trying a list of devices (Hard drive, CD-ROM, Flash drive), stored in the CMOS memory

- The BIOS is the program, and the CMOS is where the BIOS stores the date, time, and system configuration details it needs to start the computer.

- The first sector from the ***boot device*** is read into memory and executed
  - This sector contains a program that examines the ***partition table*** at the end of the ***boot sector*** to determine the ***active partition***

- Then a secondary boot loader is read in from that partition.
  - This loader reads in the OS from the ***active partition*** and starts it.

- OS performs the initialization tasks, creates whatever background processes are needed, and starts up a login program or GUI.

Fundamental Concepts

Kernel

- The most fundamental part of an operating system
    - Heart of an OS
- The first thing that is loaded into memory when OS starts loading
- Runs at all times on the computer
- Major role includes memory management, process management, disk management etc.
- Often used as another name of OS

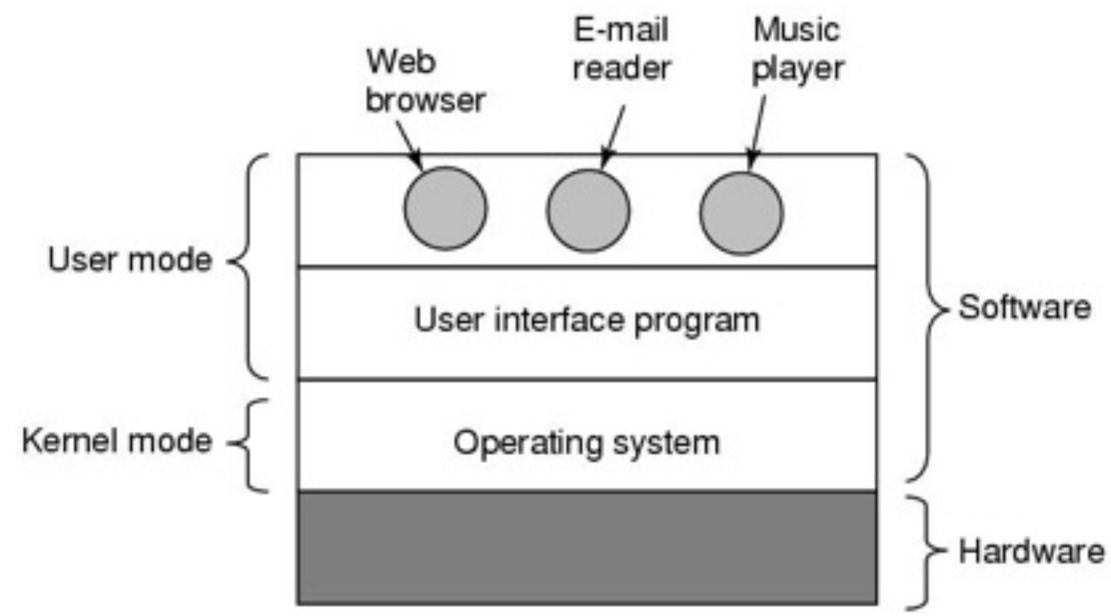
Dual-Mode Operation

- CPU executes in 2 Modes
  - Kernel mode
    - CPU can execute all machine instructions
    - CPU can use every hardware feature
  - User mode
    - permits only a subset of the instructions and a subset of the hardware features.

- Allows OS to protect itself and other system components

Dual-Mode Operation

- OS runs in kernel mode, user programs in user mode
  - OS is **Boss**, the applications are laborers
- Mode bit provided by **hardware**
  - Provides ability to distinguish when system is running user code or kernel code
  - Some instructions designated as privileged, only executable in kernel mode



Kernel Mode Execution

- When does CPU start executing in kernel mode?
  A. System Boot - starting a computer
  B. Hardware Interrupt - generated by hardware devices to signal that they need some attention from the OS.
  C. Trap
     A. A software-generated interrupt caused either by an error (i.e. division by 0 or invalid memory access) or
     B. By a specific request from a user program that an operating-system service needs to be performed.

Switching Modes

- To obtain services from the operating system,
  - an user program must make a ***system call***, which ***traps*** into the kernel and invokes the operating system.
- The **TRAP** instruction switches from user mode to kernel mode and starts the operating system.
- When the work has been completed, control is returned to the user program at the instruction following the system call.

System Calls

- Programming interface to the services provided by the OS

- Typically used from a high-level language (C or C++) program.

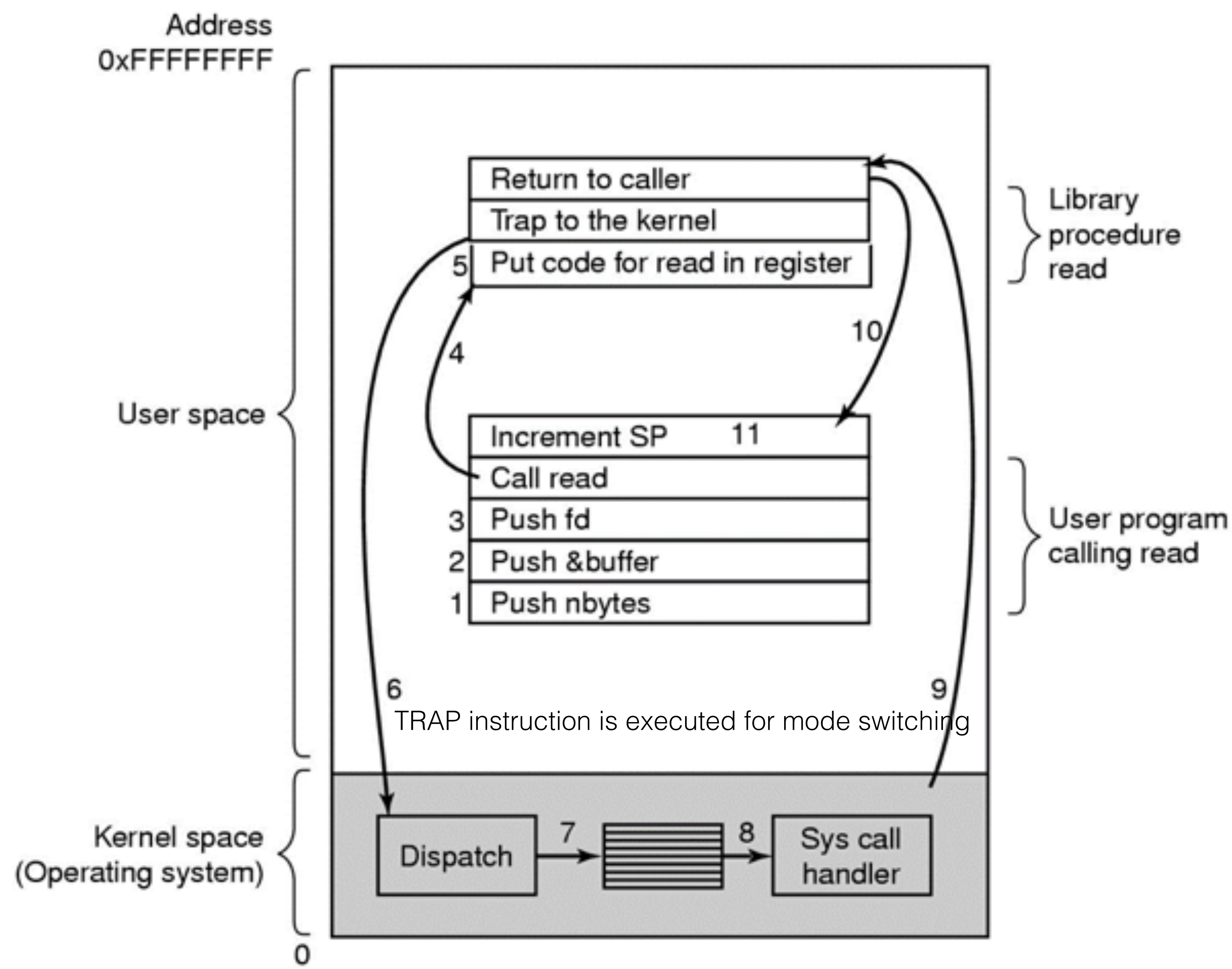
- Typically a number is associated with each system call, and the OS maintains a table indexed according to these numbers.

Steps in making a System Call

- UNIX has a *read* system call for reading files

- invoked from C programs by calling a library procedure with the same name *read* like this:

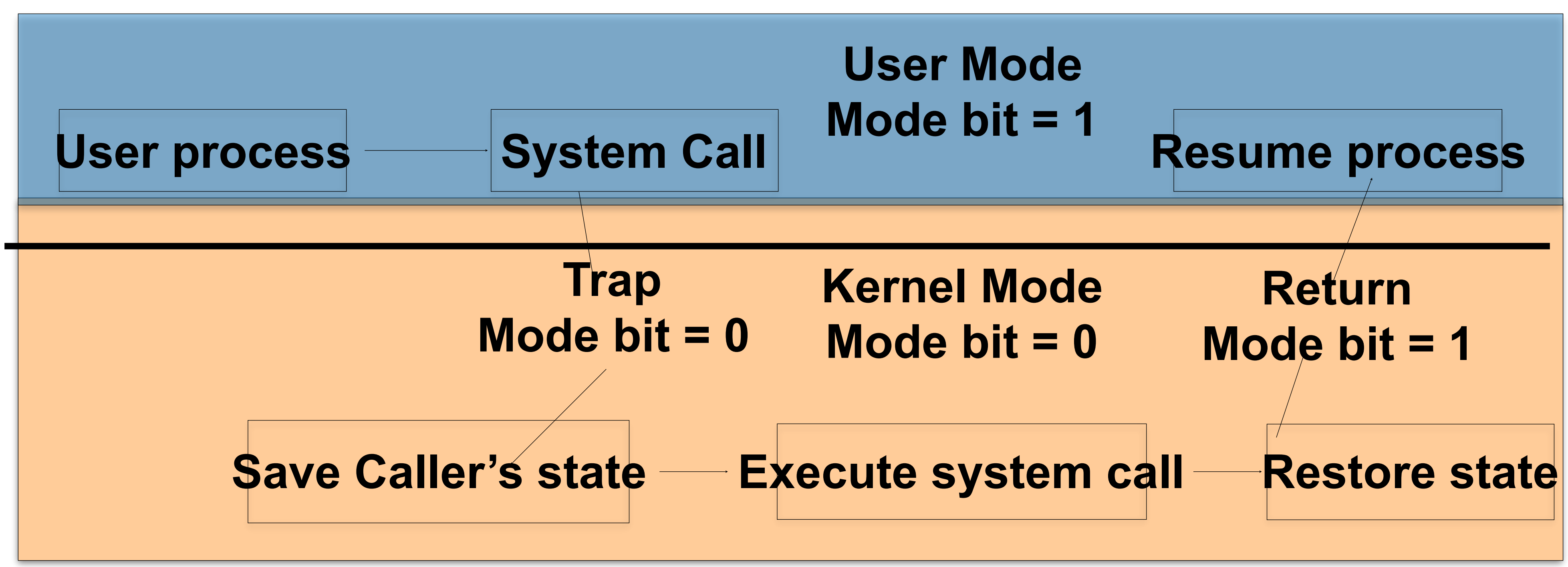
```
count = read(fd, buffer, nbytes);
```



See the steps in more details in Section 1.6

Trapping into kernel is costly

- Save minimal CPU state (PC, sp, …) – done by hardware
- Switches to KERNEL mode
- KERNEL determines what system call has occurred
- KERNEL verifies that the parameters passed are correct and legal
- Restore state before returning to user program



Examples of Windows & Unix System Calls

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

Some OS Components/Services/functions

- Process Management
- Memory Management
- I/O Management
- Deadlock Management
- File System

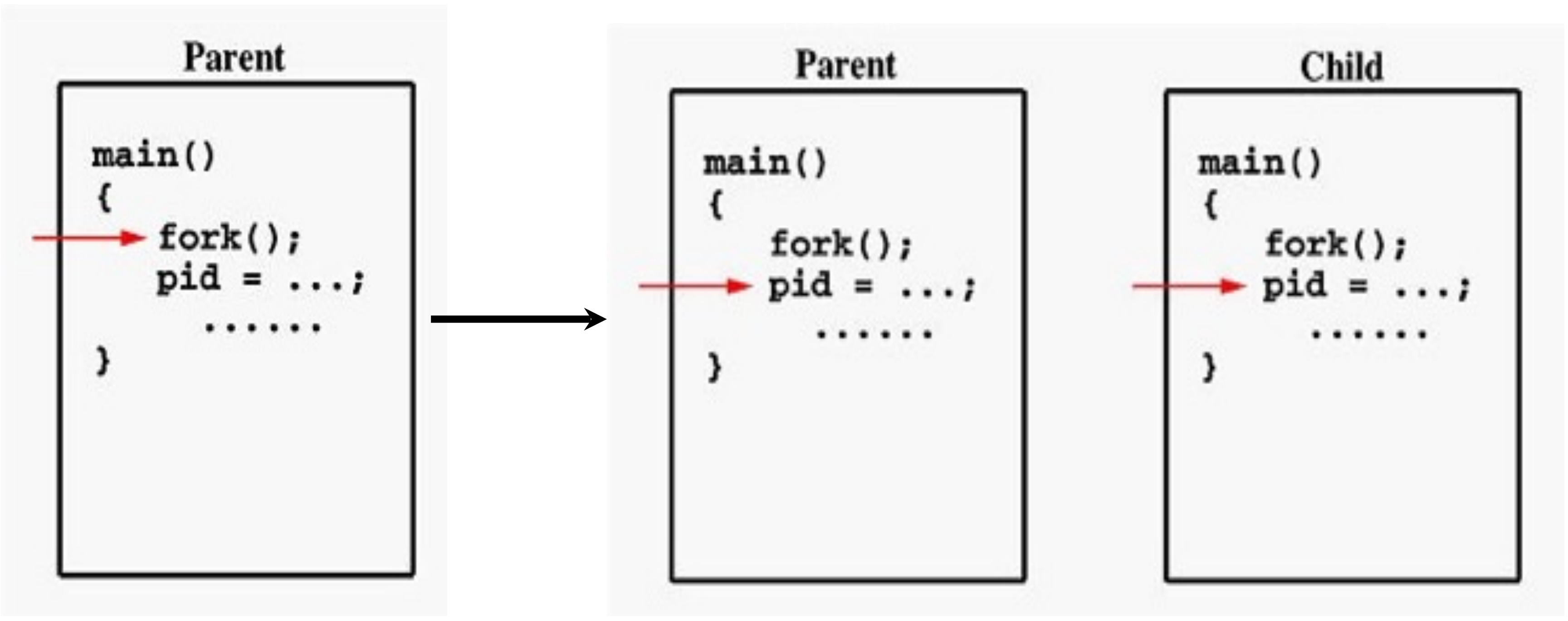
Some UNIX System Calls For Process Management

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

The `fork()` System Call

- When `fork()` is called in process A
  - Control is switched to kernel
  - Kernel creates new process B which is exact duplicate of process A
  - OS now has two identical processes to run. Both resume from after fork().
  - return value of fork() will be 0 in the child process and will be equal to the child's process identifier (PID) in the parent process



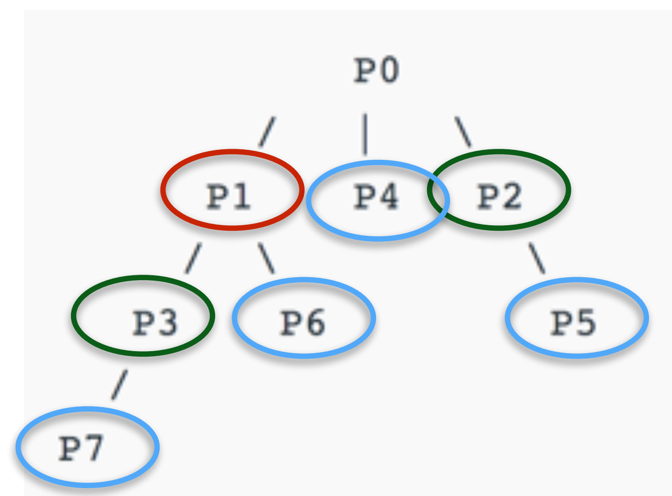
fork() Example 1

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{

    // make two process which run same
    // program after this instruction
    fork();

    printf("Hello world!\n");
    return 0;
}
```

fork() Example 2

```c
#include <stdio.h>
#include <sys/types.h>
int main()
{
    fork();
    fork();
    fork();
    printf("hello\n");
    return 0;
}
```

- What is the output of this code?
- Can you draw the fork() tree?



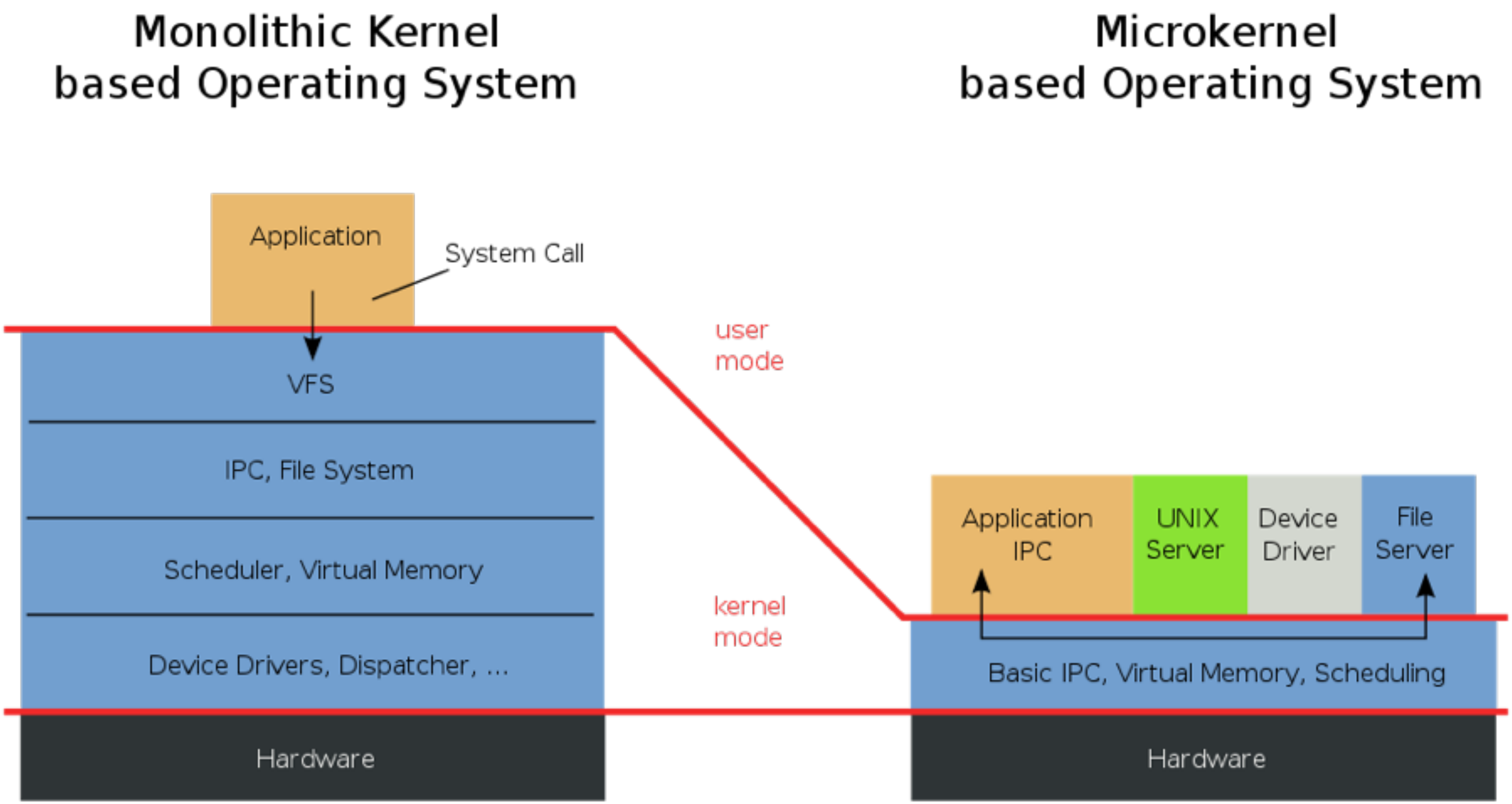
For further practice:

https://www.geeksforgeeks.org/fork-system-call/

https://www.youtube.com/watch?v=IFEFVXvjiHY

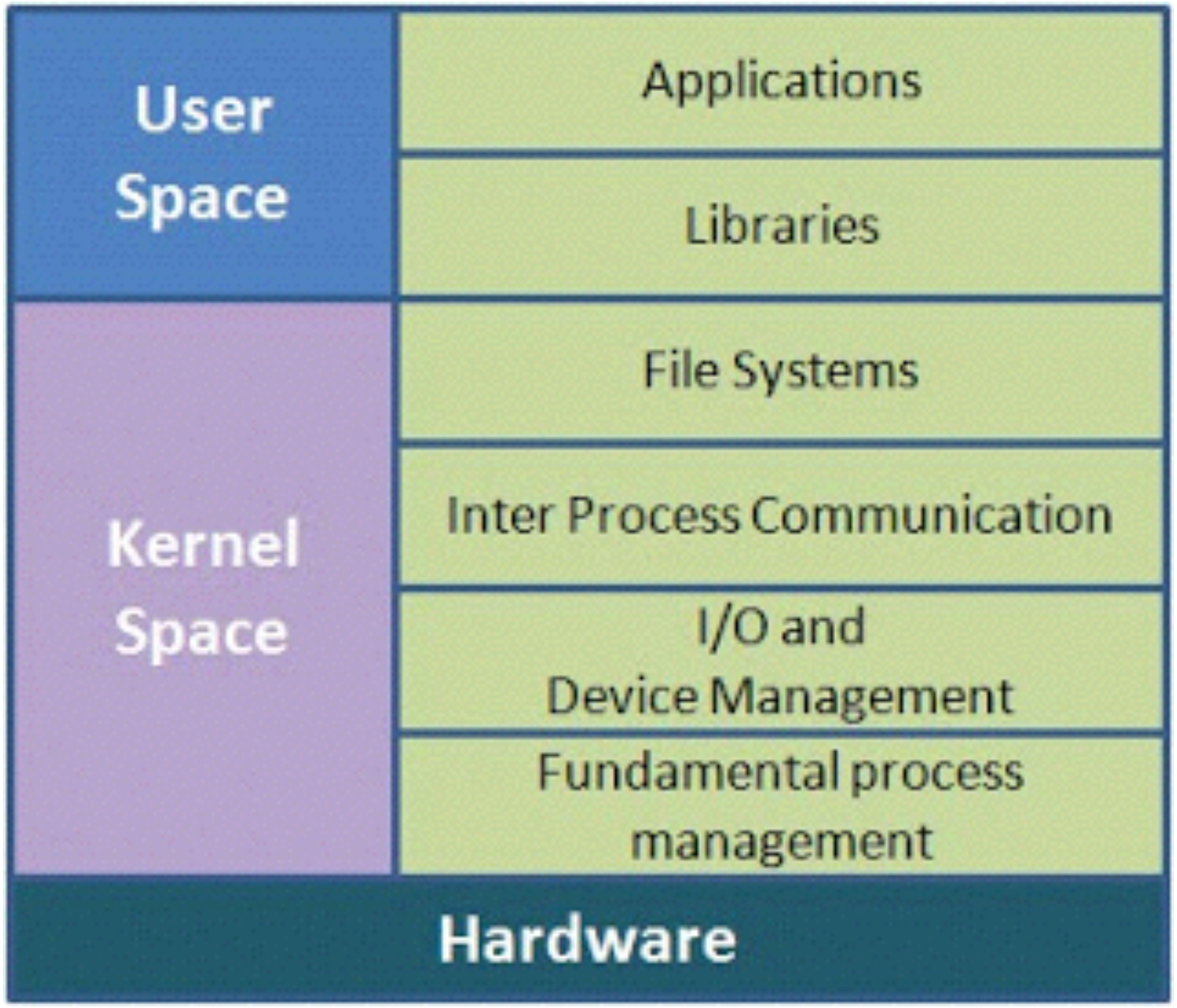
OS architecture/structure

- 2 main architectures:
  - monolithic kernel
  - microkernel


Monolithic Kernel
based Operating System
Microkernel
based Operating System
Application
System Call
VFS
IPC, File System
Scheduler, Virtual Memory
Device Drivers, Dispatcher, ...
Hardware
user
mode
kernel
mode
Application
IPC
UNIX
Server
Device
Driver
File
Server
Basic IPC, Virtual Memory, Scheduling
Hardware

Monolithic kernel

- Entire operating system runs as a single program in kernel mode.
- Set of system calls implement all ***operating system services*** such as process management, memory management etc.

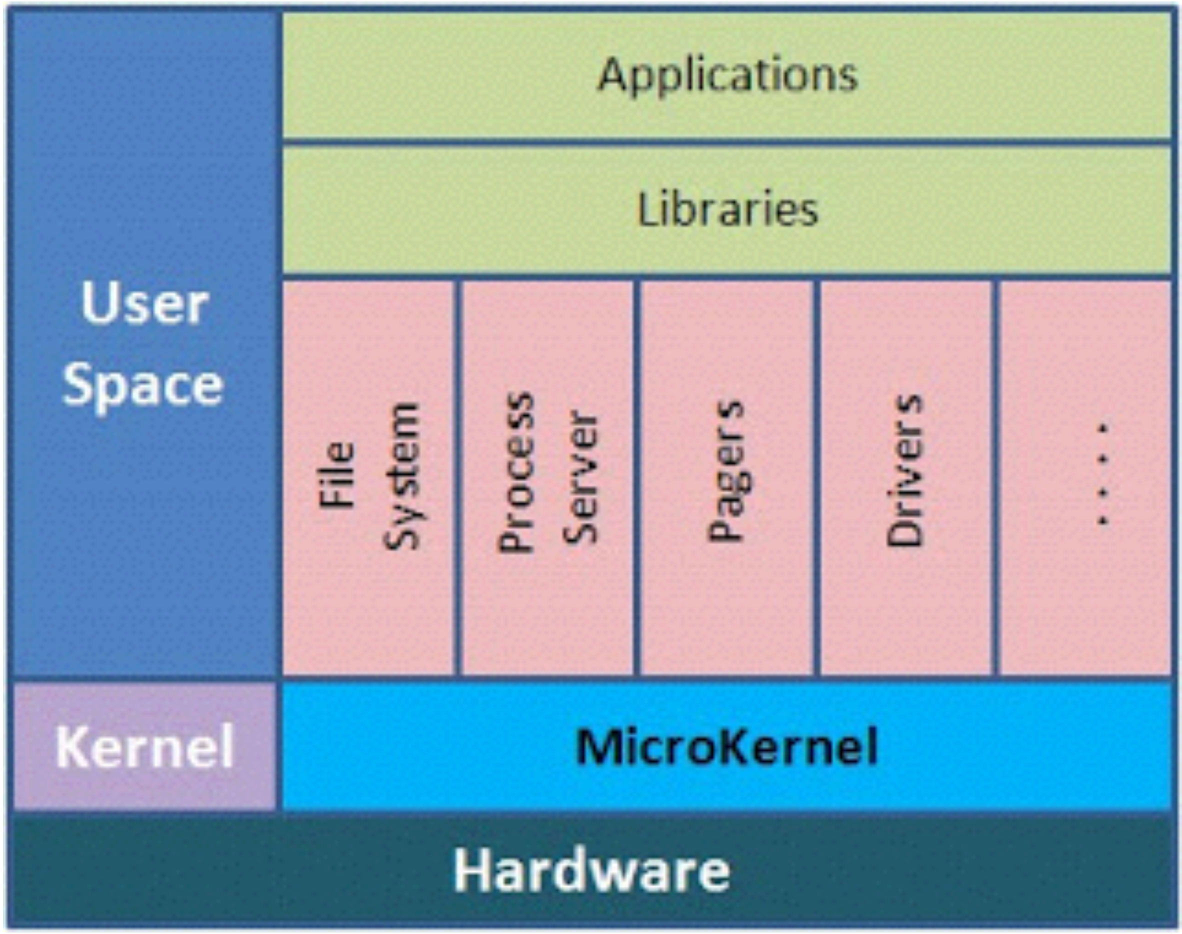


Monolithic kernel

- Drawbacks:
  - increased kernel size
  - difficult to understand
  - bad maintainability
- Advantage
  - Good performance

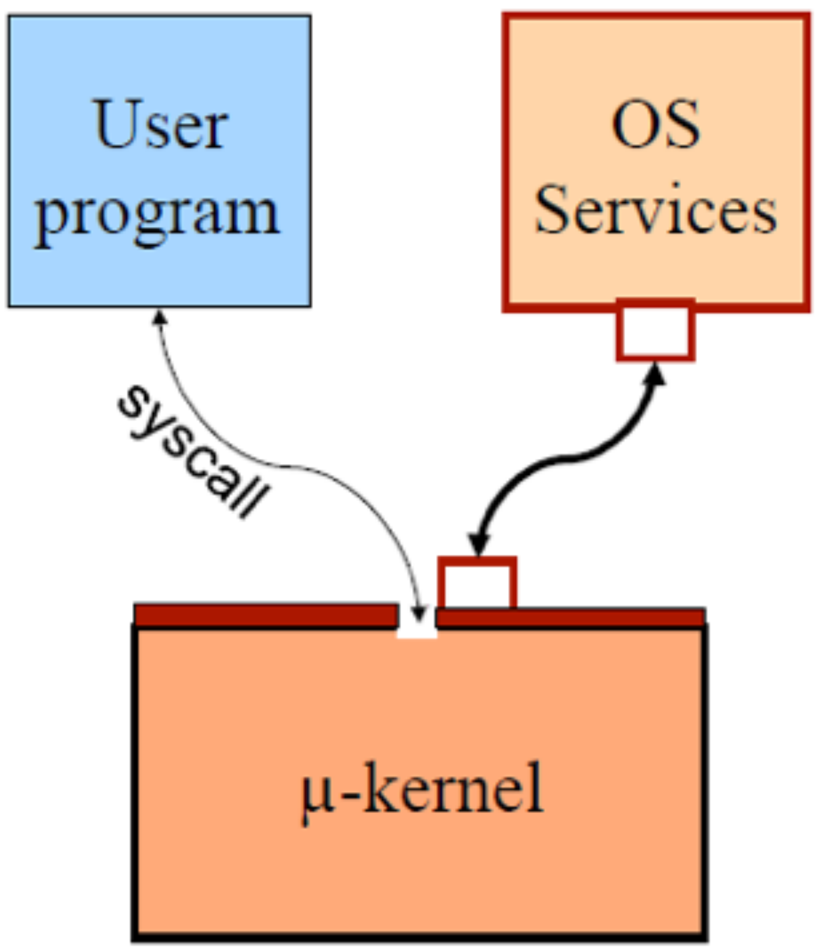
Microkernel

- reduce the kernel to basic process communication and I/O control
- let the other system services reside in user space in form of normal processes
- user space services are called as servers
  - Do most of the work of the operating system.
  - One server for managing memory issues
  - one server does process management
  - another one manages drivers, and so on.



Microkernel

- The main function of the microkernel is to provide a communication facility between the user program and the various servers
- if the user program wishes to access a file, it must interact with the file server.
- The user program and service communicate indirectly by exchanging messages with the microkernel.



Microkernel

- Drawbacks:
  - Performance overhead of user space to kernel space communication
- Advantage
  - Easier to extend the OS
    - All new services are added to user space and consequently do not require modification of the kernel
  - More secure & reliable
    - most services are running as user processes. If a service fails, the rest of the operating system remains untouched.

Example

- Bugs in the kernel can bring down the system instantly.
- What will be the impact of a bug (severe one) in audio driver
  - In a Monolithic Kernel based operating system?
  - In a Microkernel based operating system?

Multiuser OS

- A multi-user operating system is an operating system that allows multiple users to access underlying hardware resources simultaneously.
  - Supports concurrent access, which means that multiple users can access the system at the same time.
  - Users can access the system remotely, logging in from a remote location using a network connection.
  - System administrator can allocate system resources, such as CPU time, memory, and storage space, to different users.
  - Some of the types of Multi-User Operating systems are : Distributed System, Time sliced system, Multiprocessor system

Multi-processor OS

- A multiprocessing operating system is a type of operating system that makes use of multiple processors to work parallelly to perform the given task. All the available processors are connected to peripheral devices, computer buses, physical memory, and clocks.
    - multiprocessing os shares a common memory between all the available processors
    - OS does the job of allocation of resources for each processor
    - multiprocessing os improves the overall performance of the system.
    - UNIX, LINUX, Solaris, Windows Server are the most widely used multi-processing operating system.