CS304 Operating Systems

DR GAYATHRI ANANTHANARAYANAN

gayathri@iitdh.ac.in

Materials in these slides are borrowed from textbooks and existing operating systems courses

Course Details & Logistics

Monday – 11.30AM to 12.15PM

Wednesday – 10.30AM to 11.15AM

Friday – 8.30AM to 9.15AM

**CS314 – Operating Systems Lab

TA information:

Chandrasekhar - chandrashekar.s@iitdh.ac.in

Pavan - pavankumar.patil@iitdh.ac.in

References :

"Operating System: Design and Implementation (The Minix Book)", 3rd edition by Andrew S Tannenbaum and Albert S Woodhull

''Operating System Concepts'', 8th edition, by Abraham Silberschatz, Peter B. Galvin, and Greg Gagne, Wiley-India edition

Tentative Grading Components

CS304 – OS Theory

Midsem – 20%

Quizzes – 15%

Seminars – 15%

Assignments –20%

Endsem – 30%

CS314 – OS-Lab

7-8 Labs/Experiments in total (each of varying complexity)

For each experiment, marks will be based on code and report submitted, Demos & Viva

**Lab Exam/Test

Note:

Academic honesty is expected from each student participating in the course. NO sharing (willing, unwilling, knowing, unknowing) of assignments/ codes between students is allowed. Academic violations will be handled as per actions mentioned in https://intranet.iitdh.ac.in:444/pdf/disciplinaryactionsiitdharwad.pdf

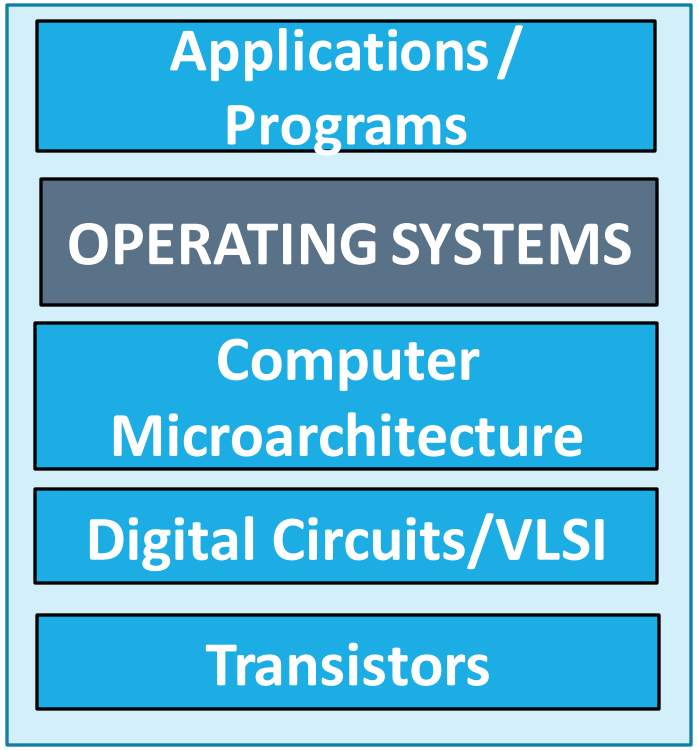
Course Outcomes

1. The student will be able to understand different functionalities in the OS ranging from scheduling and synchronization to memory management and file systems.

2. The student will be able to differentiate between types of operating systems and understand the inner workings of kernel level implementations

3. The student will gain sufficient knowledge to be able to design and write his/her own schedulers, driver codes etc.

Abstractions in a Computer System



What is an OS?

Middleware between user programs and system hardware

Manages hardware:CPU, main memory, I/Odevices (disk, networkcard, mouse,keyboard etc.)

- Hardware Abstracter - turns hardware into something that applications can use

- Resource Manager - manage system's resources [What about security?]

What happens when you execute a program on the CPU?

Execution of a Program

A compiler translates high level programs into an executable (“.c” to “a.out”) [eg. gcc, icc]

The exe contains instructions that the CPU can understand, and data of the program (all numbered with addresses)

Instructions run on CPU: hardware implements an instruction set architecture (ISA)

CPU also consists of a few registers, pointer to current instruction (program counter or PC), Operands of instructions, memory addresses

Execution of a Program

To run an exe, CPU

–fetches instruction pointed at by PC from memory

–loads data required by theinstructions into registers

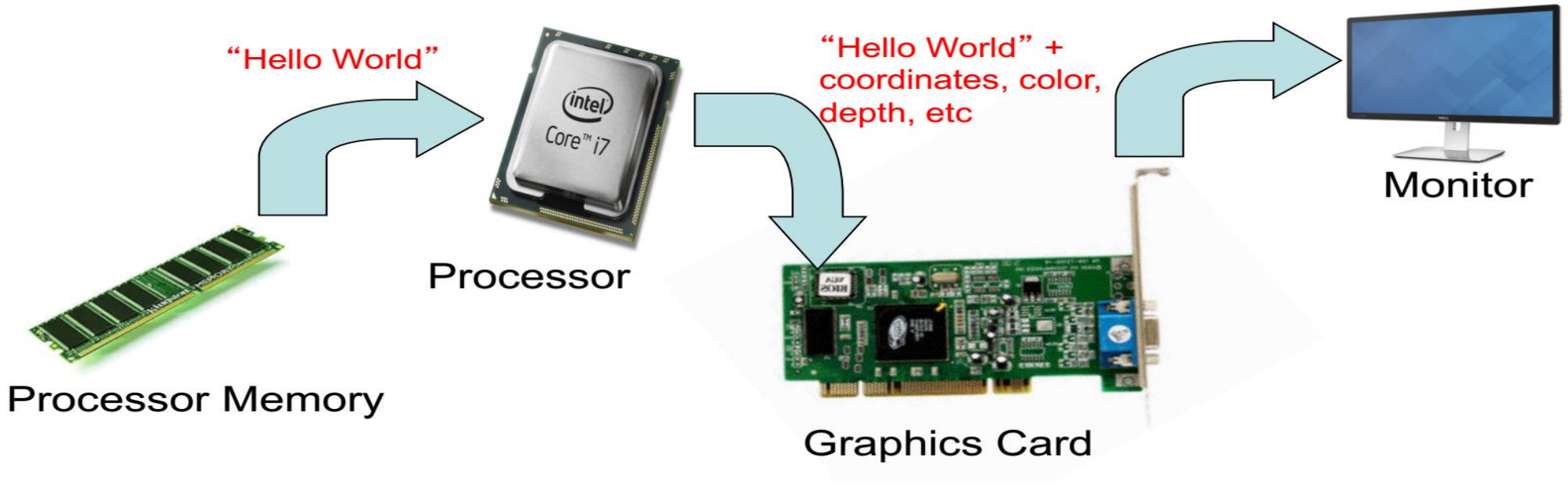
–decodes and executes theinstruction

–stores results to memory

Most recently used instructions and data are in CPU caches for faster access

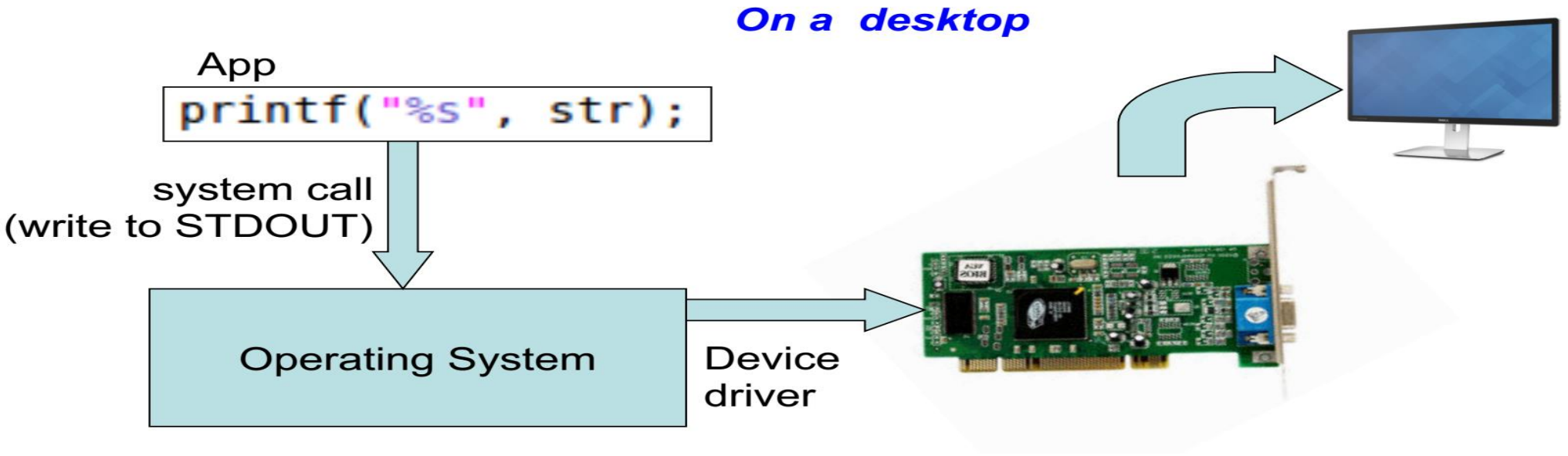
What happens in this case?

```
#include<stdio.h>
int main(){
char str[] = "Hello World\n";
printf("%s", str);
}
```



Hardware Dependent - What should be done for a new hardware? Can be complex and tedious

Without an OS, all programs need to take care of every nitty gritty detail.



Easy to program apps – No more nitty gritty details for programmers

Reusable functionality – Apps can reuse the OS functionality

Portable - OS interfaces are consistent. The app does not change when hardware changes

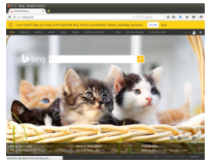
What does OS do?

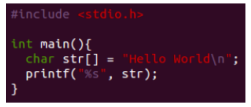
- Manages program memory–Loads program executable(code, data) from disk to memory
- Manages CPU–Initializes program counter (PC) and other registers to begin execution
- Manages external devices–Read/write files from disk.

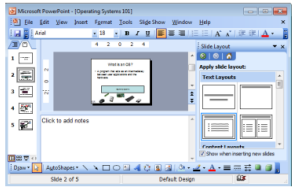
Resource Manager ?

- Multiple apps but limited hardware

Apps





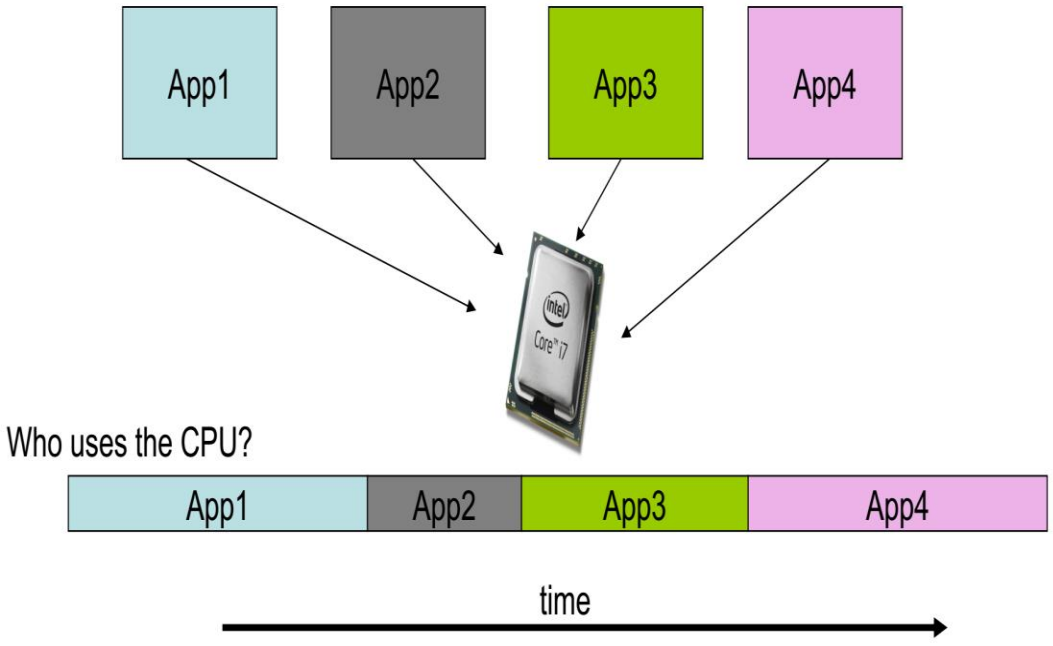


Operating Systems

A few processors

Allows multiple apps to share resources
– protects apps from each other
– Improves performance by efficient utilization of resources
-Isolation of resources

Resource Manager ?



OS provides the process abstraction

–Process: a running program

–OS creates and manages processes

•Each process has the illusion of having the complete CPU, i.e., OS virtualizes CPU

•Timeshares CPU between processes

•Enables coordination between processes

Resource Manager

OS manages the memory of the process: code, data,stack, heap etc

•Each process thinks it has a dedicated memory space for itself, numbers code and data starting from 0(virtual addresses)

•OS abstracts out the details of the actual placement inmemory, translates from virtual addresses to actual physical addresses

OS has code to manage disk, network card,and other external devices: device drivers

•Device driver talks the language of the hardware devices

- Issues instructions to devices (fetch data from a file)
- Responds to interrupt events from devices (user has pressed a key on keyboard)

•Persistent data organized as a filesystem on disk

Summary

An operating system is a layer of systems software that :

- Directly has privileged access to the underlying hardware;
- Hides the hardware complexity;
- Manages the hardware on behalf of one or more applications according to some predefined policies;
- In addition, it ensures that applications are isolated and protected from each other;

Trivia Time

Which of the following are likely components of an operating system ? Write all that apply

A) File editor

B) File System

C) Device Driver

D) Cache Memory

E) Web Browser

F) Scheduler

Trivia Time

For the following options indicate if they are examples of abstraction (AB) or arbitration (RM) [Resource Management]

A) Distributing memory between processes

B) Supporting different types of speakers

C) interchangeable access of hard disk or SSD

OS Examples

Based on the environment in which the hardware is operated (Desktop/ Server/Embedded)

Microsoft Windows – OS since early 1980's

Unix Based – Originated at Bell Labs in 1960's

- Mac OS X (BSD) Berkley System Distribution
- Linux – Open source, bundled with may popular s/w libraries [Ubuntu, CentOS]

Android – Embedded form of Linux

IOS -

Symbian

What makes each of them different ? Design & Implementation

OS Elements

Abstractions

Process, Thread, File, Socket, Memory Page

Mechanisms

Create, Schedule, Open, Write, Allocate

Policies

Least Recently Used [LRU], Earliest Deadline First [EDF]

OS elements Example

Abstractions

Memory Page – Some addressable region in memory

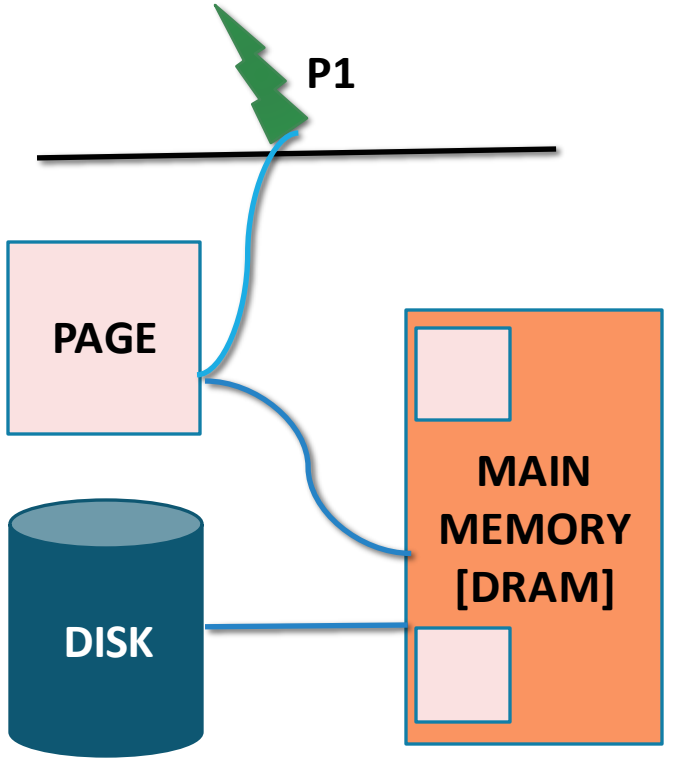
Of some fixed size [eg 4K]

Mechanisms

Allocate that page in DRAM**, map to a process** [address space]

-Allows the process to access actual physical memory

that corresponds to the contents of the page



Policies

Page may be moved across different locations in DRAM/ moved to DISK [Page Swapping]

-LRU [Least recently used ones likely will not be used anytime in future]

Design Principles

Separation of Mechanism and Policy

Implement flexible mechanisms to support many policies [eg. For Mem Management LRU, LFU, Random]

Optimize for the common case

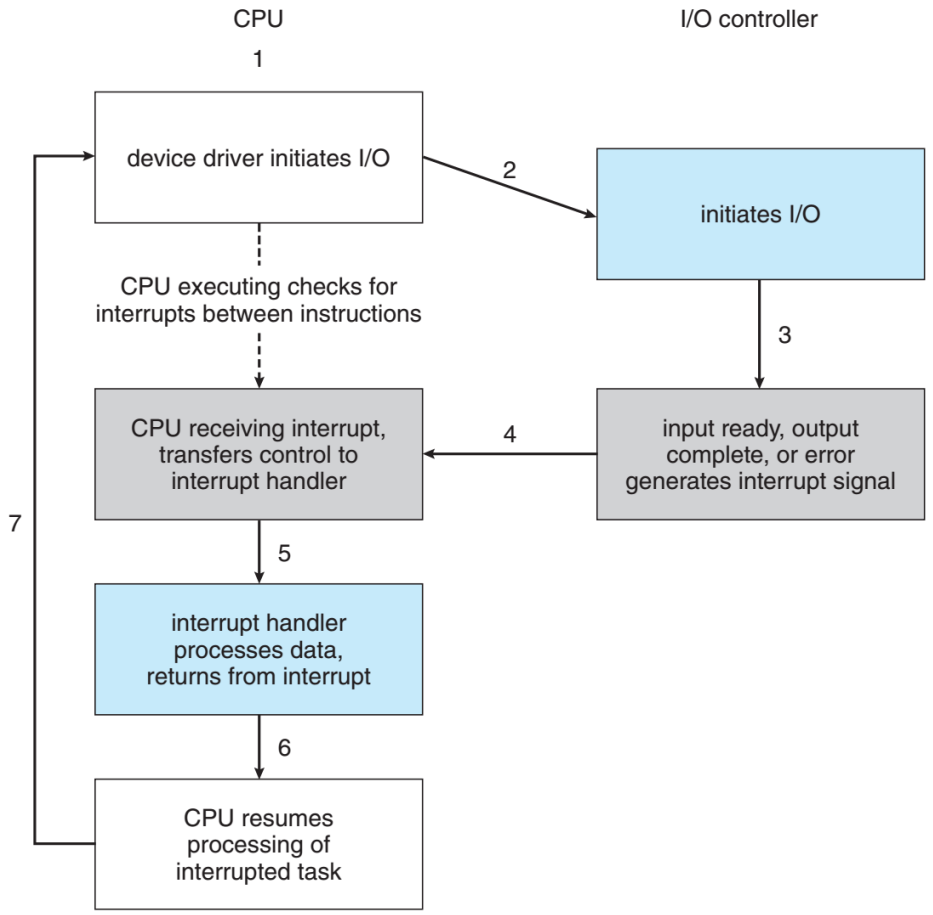
Where the OS will be used ?

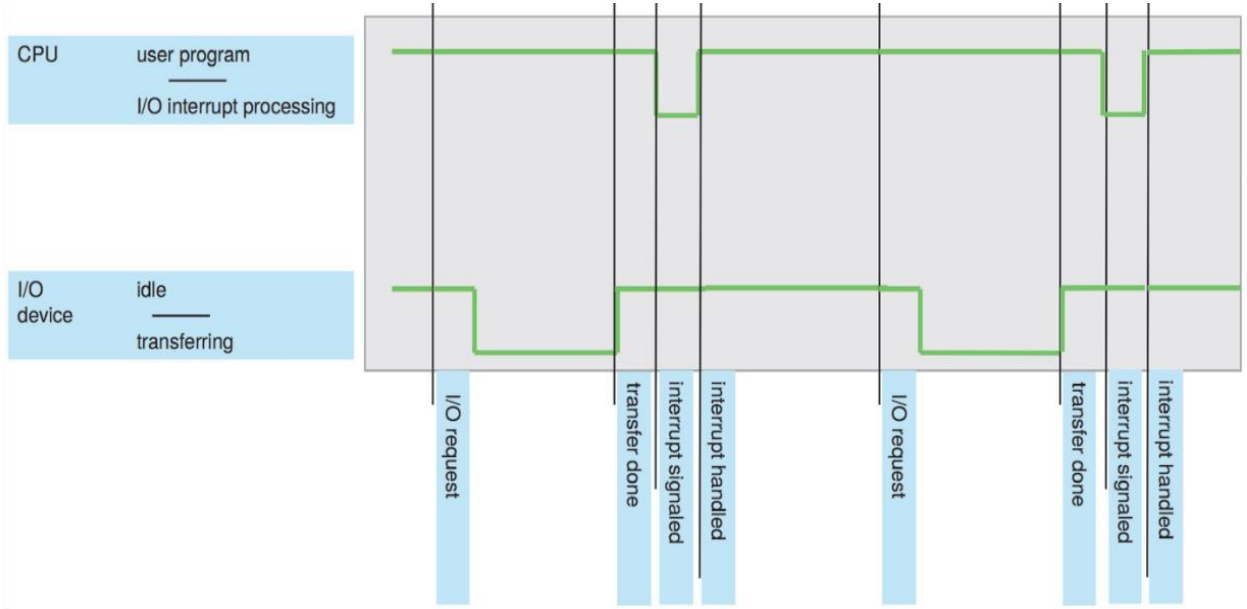
What will the user want to execute on that machine ?

What are the workload requirements?

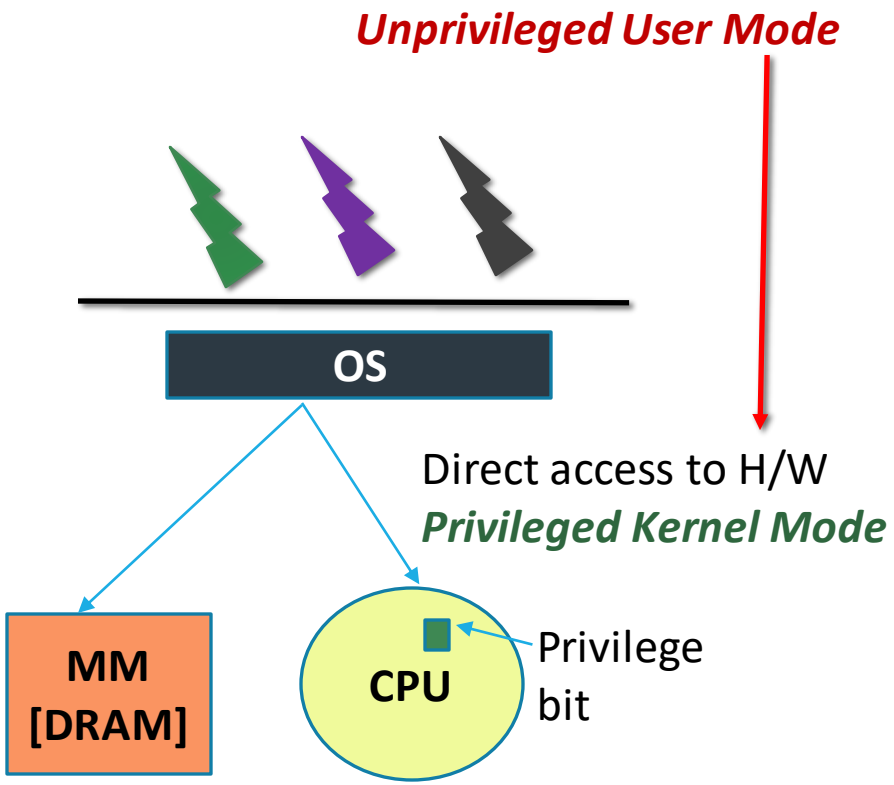
- **Convenience, abstraction of hardware resources for user programs**
- **Efficiency of usage of CPU, memory, etc.**
- **Isolation between multiple processes**

Hardware Interrupts





User/Kernel Protection Boundary



User to Kernel level switching is supported by H/W

In Kernel mode; **a special bit is set in CPU**; If this bit is set; any instruction that manipulates H/W is permitted to execute

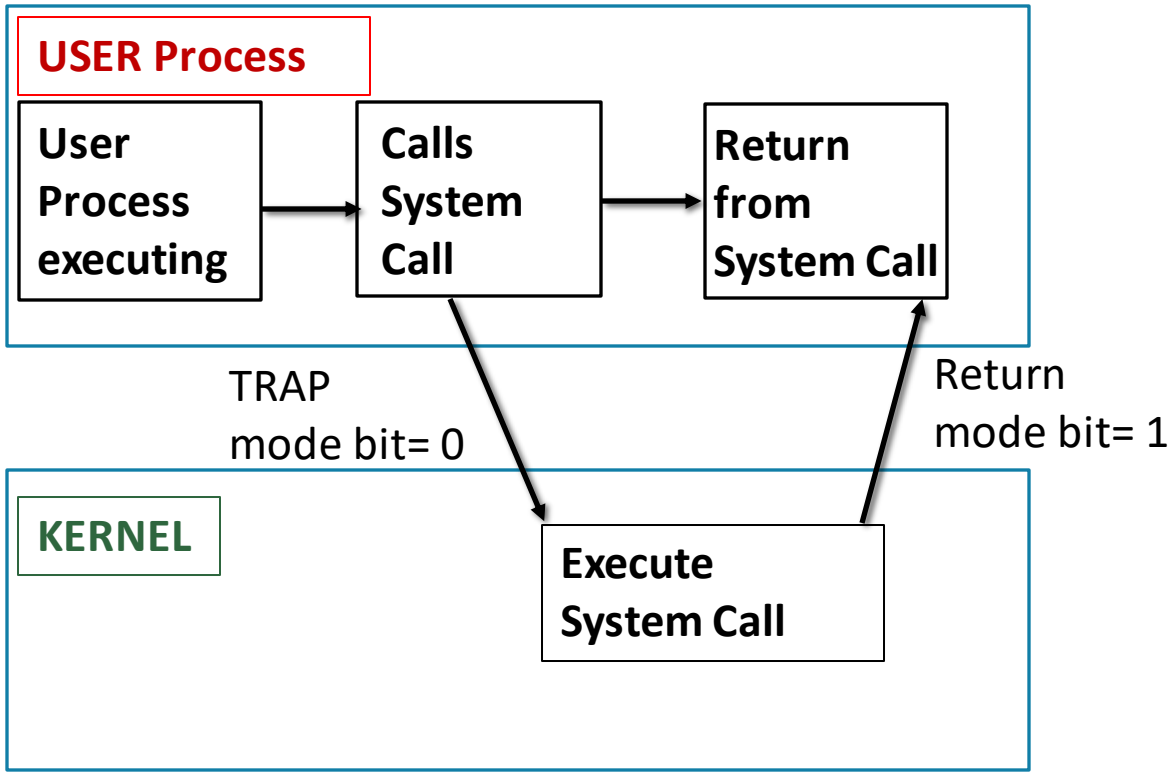
In User mode; privilege bit is not set and so instructions would be forbidden to manipulate H/W

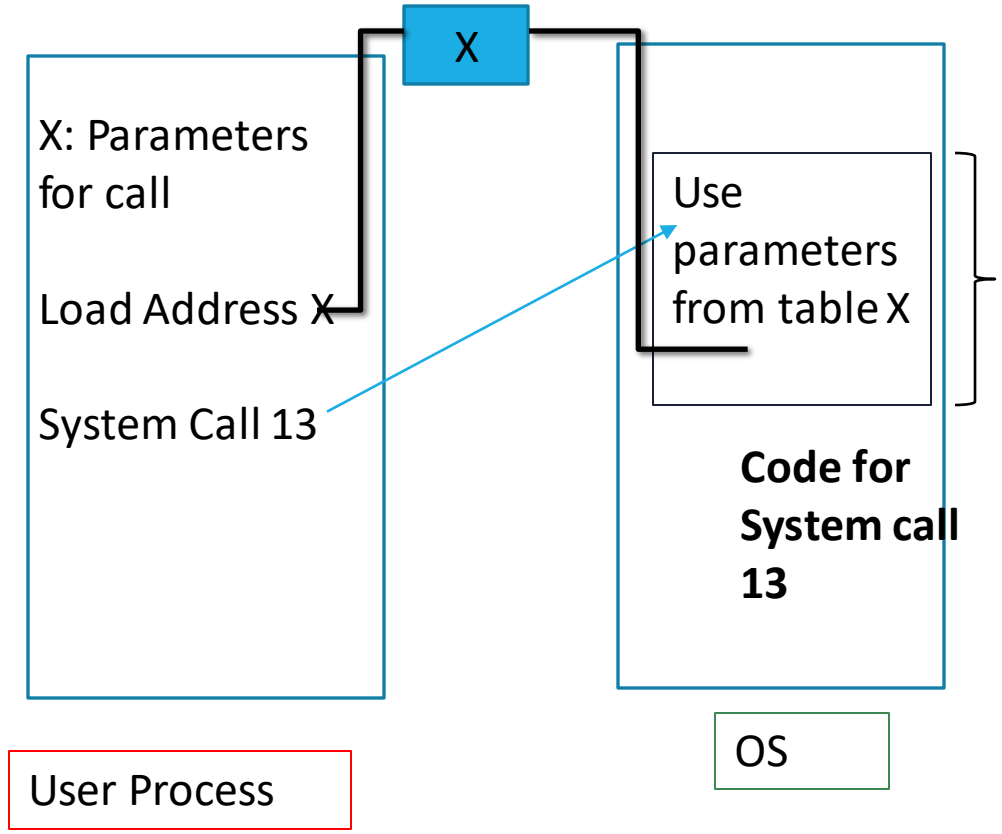
TRAP instruction is generated when user mode tries to perform privileged operations [Application execution interrupted, control transfer to OS at a specific location; OS will check what caused the trap to occur [verify to grant access or NOT]]

SYSTEM CALLS –Privileged operations will be performed by OS on behalf of applications {open[file], send[socket], mmap [memory]}

SIGNALS – OS notifications to Applications [kill]

System Calls





To make a system call; app must write arguments; save relevant data at well-defined location

System Calls

	Windows	Unix

Process control	`CreateProcess()`
	`ExitProcess()`
	`WaitForSingleObject()`
		`fork()`
		`exit()`
		`wait()`

File management	`CreateFile()`
	`ReadFile()`
	`WriteFile()`
	`CloseHandle()`
		`open()`
		`read()`
		`write()`
		`close()`

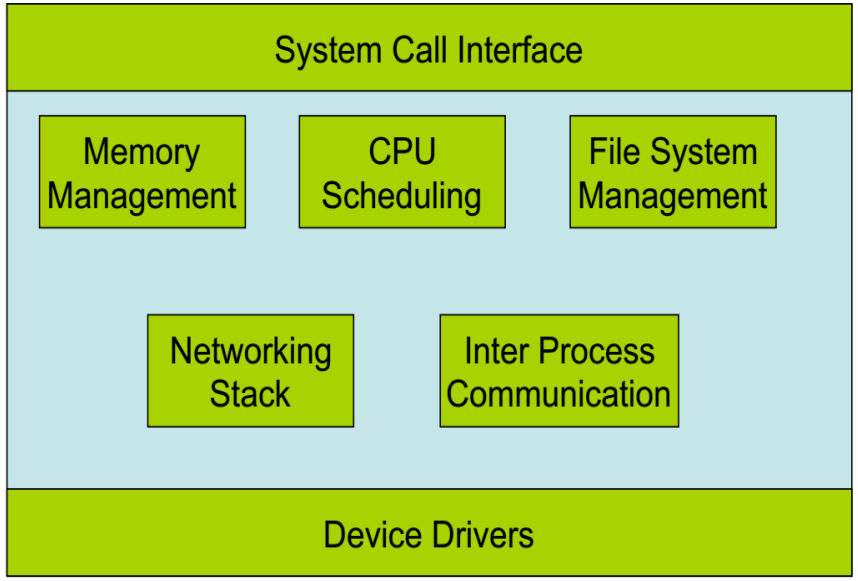
Device management	`SetConsoleMode()`
	`ReadConsole()`
	`WriteConsole()`
		`ioctl()`
		`read()`
		`write()`

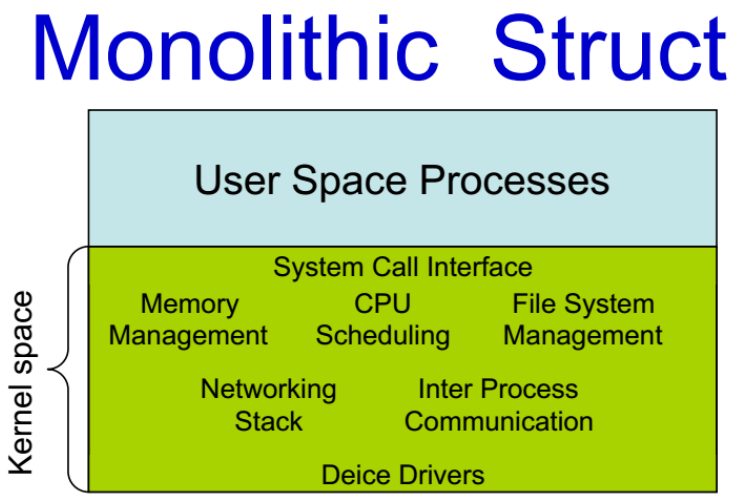
Information maintenance	`GetCurrentProcessID()`
	`SetTimer()`
	`Sleep()`
		`getpid()`
		`alarm()`
		`sleep()`

Communications	`CreatePipe()`
	`CreateFileMapping()`
	`MapViewOfFile()`
		`pipe()`
		`shm_open()`
		`mmap()`

Protection	`SetFileSecurity()`
	`InitlializeSecurityDescriptor()`
	`SetSecurityDescriptorGroup()`
		`chmod()`
		`umask()`
		`chown()`

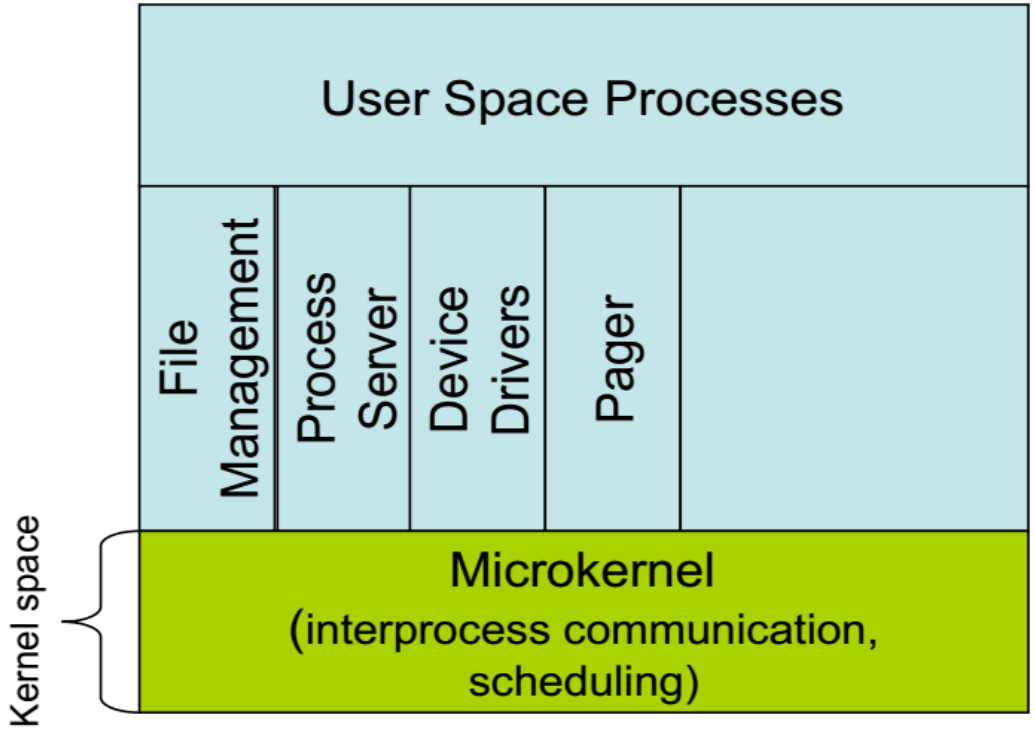
Monolithic OS





- Linux, MS-DOS, xv6
- All components of OS in kernel space
- Cons : Large size, difficult to maintain, likely to have more bugs, difficult to verify
- Pros : direct communication between modules in the kernel by procedure calls

Microkernel



Eg. QNX and L4

- Highly modular.
  - Every component has its own space.
  - Interactions between components strictly through well defined interfaces (no backdoors)
- Kernel has basic inter process communication and scheduling
  - Everything else in user space.
  - Ideally kernel is so small that it fits the first level cache

Comparison

	Monolithic	**Microkernel**
Inter process communication	Signals, sockets	Message queues
Memory management	Everything in kernel space (allocation strategies, page replacement algorithms,)	Memory management in user space, kernel controls only user rghts
Stability	Kernel more ‘crashable’ because of large code size	Smaller code size ensures kernel crashes are less likely
I/O Communication (Interrupts)	By device drivers in kernel space. Request from hardware handled by interrupts in kernel	Requests from hardware converted to messages directed to user processes
Extendibility	Adding new features requires rebuilding the entire kernel	The micro kernel can be base of an embedded system or of a server
Speed	Fast (Less communication between modules)	Slow (Everything is a message)

Started out as a library to provide common functionality across programs

- Later, evolved from procedure call to system call: what's the difference?
- When a system call is made to run OS code,the CPU executes at a higher privilege level
- Evolved from running a single program to multiple processes concurrently

Evolution driven by Hardware improvements + User needs

– eg. low power requirements, Increased / reduced security, lower latency

– Evolution by

- New/better abstractions
- New/better resource management
- New/better low level implementations