Information Technology Institute

SWE Tracks' renovation

In view of SWEBOK V3.0

Information Technology Institute

Operating Systems

Lecture 1

Ahmed M. Abdelfattah

Project Manger

ITP (Intensive Training Program)

ITI (Information Technology Institute)

Course Duration and Evaluation

- Duration: 6 hours
  - 2 Lectures (6 hours)

- Evaluation Criteria:
  - A comprehensive exam after finishing all the main conceptual courses

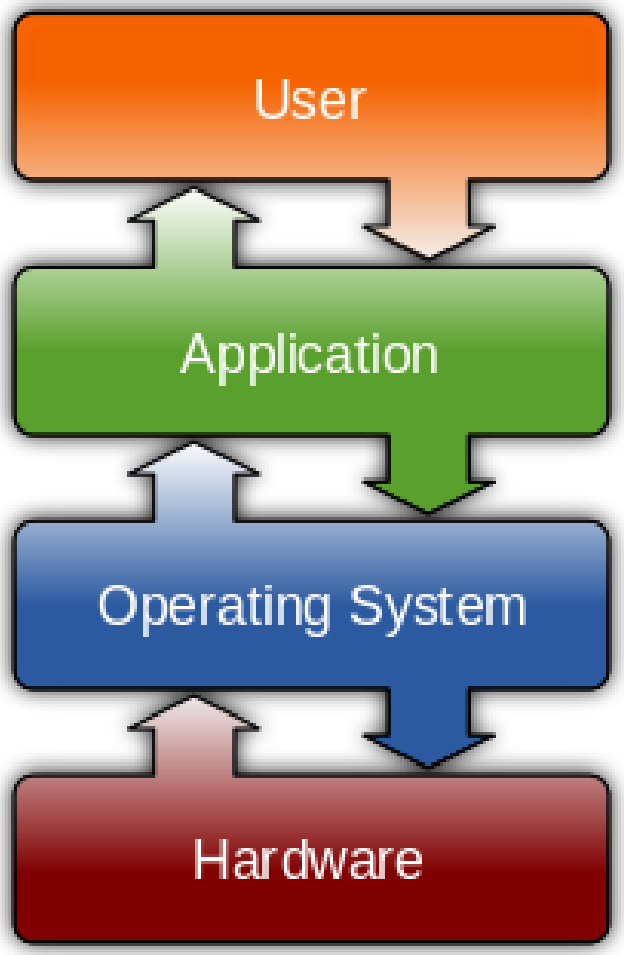
Course Outlines

- Lesson 1: Introduction to Operating System and Computer System
- Lesson 2: Processes and Scheduling
- Lesson 3: Memory Management
- Lesson 4: I/O Management
- Lesson 5: File Systems
- Lesson 6: Access and Protection
- Lesson 7: Virtualization and User Interface and Shells

Introduction to Operating System and Computer System

Computer System Components

- ***Hardware:*** provides basic computing resources (CPU, memory, I/O devices)
- ***Operating system:*** controls and coordinates the use of the hardware among the various application programs for the various users
- ***Applications programs:*** define the ways in which the system resources are used to solve the computing problems of the users (compilers, database systems, video games, business programs)
- ***Users:*** people, machines, other computers



Operating System

- Is a program that controls the execution of application programs
- Is an interface between the user and hardware
- Masks the details of the hardware to application programs, so the OS must deal with hardware details

Examples of OS

- Some popular Operating Systems include:
  - Linux-based OS
  - Microsoft Windows
  - macOS (used for Apple's computers and client models)
  - iOS (used for Apple's smartphone/tablet models)
  - Android

All of these operating systems have different generations, versions, and upgrades.

What is an Operating System?

- The general definition of any system is:

Some components integrated together for perform a desired task.

- So, the definition of computer operating system is

Some components integrated together for operating the computer system (HW &SW).

What is an Operating System?

The definition of the operating system is:

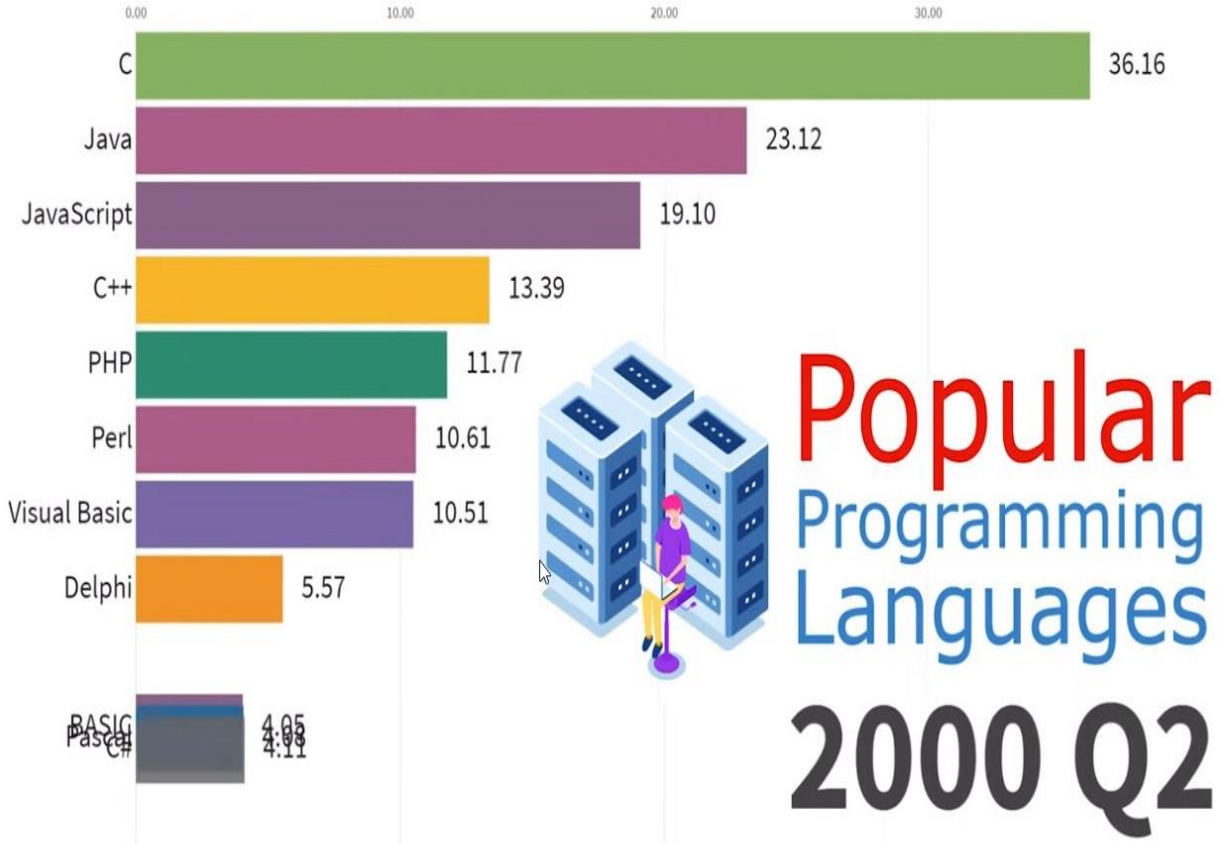
An operating system, commonly referred to as the OS, is a program that controls the execution of other programs running on the system.

It acts as a facilitator and intermediate layer between the different software components and the computer hardware

- When any operating system is built, it focuses on three main objectives:
  - *Efficiency* of the OS in terms of responsiveness, fluidity, and so on
  - *Ease of usability* to the user in terms of making it convenient
  - *Ability to* abstract and *extend* to new *devices and software*

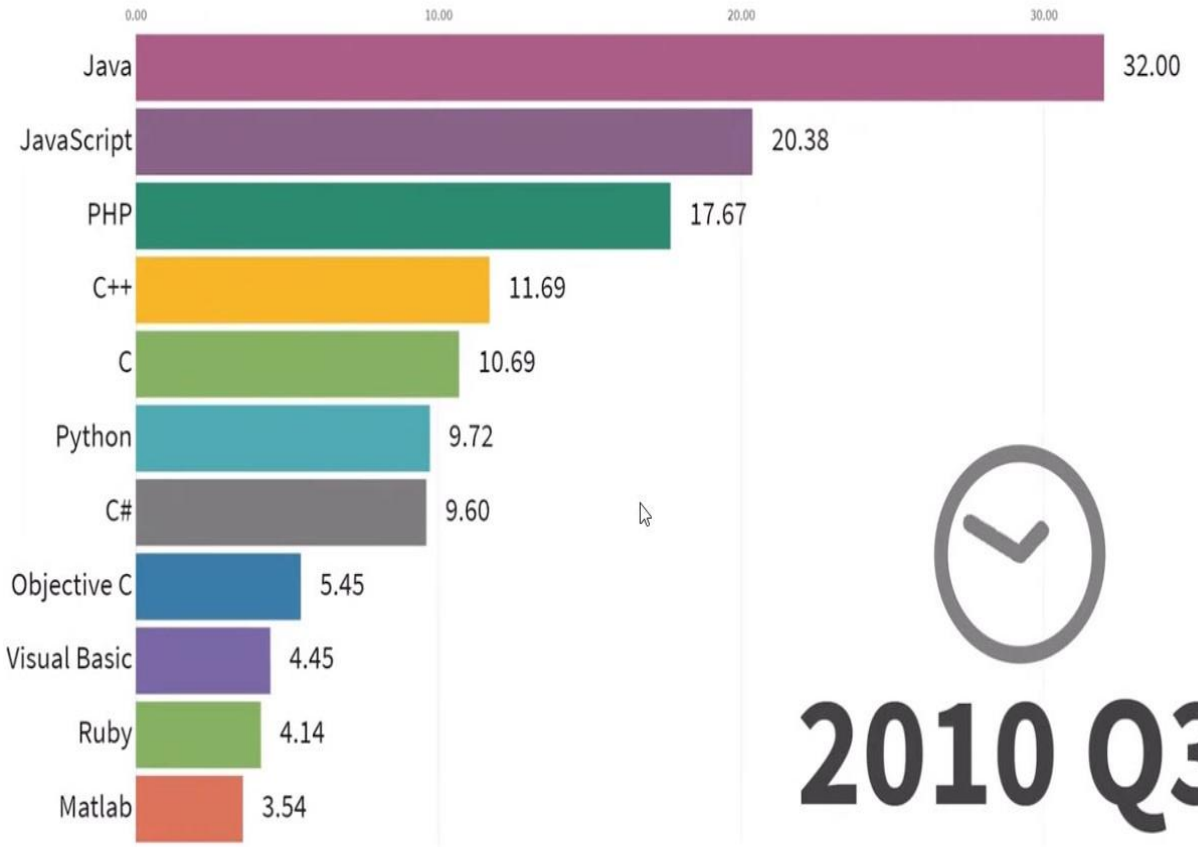
In which programming language the OS written ?

Most Popular Programming Languages



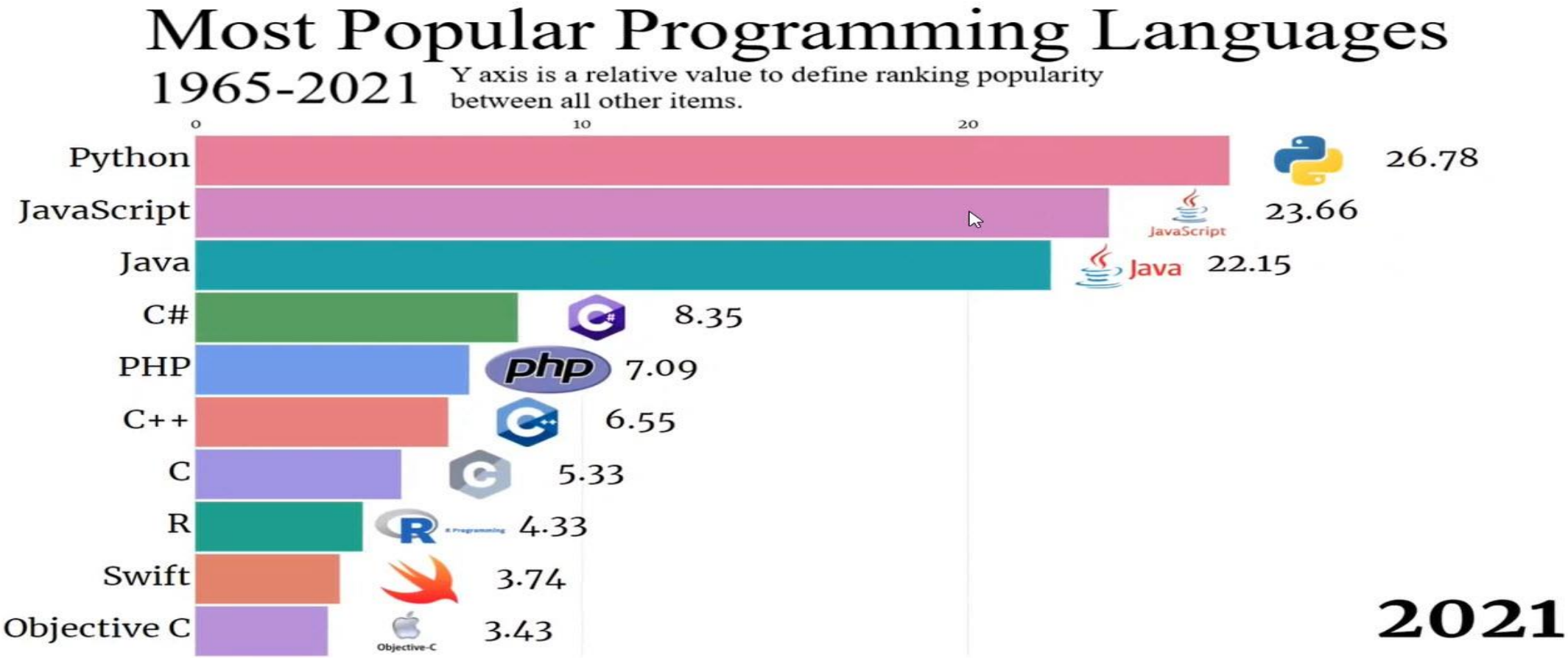
Popular Programming Languages

2000 Q2



2010 Q3

Most Popular Programming Languages



In which programming language the OS written ?

- The **UNIX** operating system's development started in 1969, and its code was **rewritten in C in** 1972. The C language was actually created to move the UNIX kernel code from assembly to a higher level language.

- In 1985 **Windows** 1.0 was released. Although Windows source code is not publicly available, it's been stated is **mostly written in C** with some parts in assembly.

- **Linux** kernel development started in 1991, and it is also **written in C**.

Ref: https://www.toptal.com/c/after-all-these-years-the-world-is-still-powered-by-c-programming

In which programming language the OS written ?

- Most of the operating systems are written in the C/C++ languages.

- These not only include Windows or Linux (the Linux kernel is almost entirely written in C), but also Google Chrome OS, RIM Blackberry OS 4.x, Symbian OS, Apple Mac OS X, iPAD OS, Apple iPhone iPod Touch, and Cisco IOS (which is mainly comprised of compiled C and C++ code).

Computer Operating System

- **When a computer turns on**, the processor will execute the instructions that are presented to it; generally, the first code that runs is for the boot flow, called ***bootstrap***, which is a framework stored in **ROM** contains some instructions called basic input output instructions (**BIOS**).
- All these need to be abstracted and handled efficiently and seamlessly. The user expects the system to "just work." The ***operating system*** facilitates all of this and more.

Role of the Bootstrap

- Bootstrapping is the process of loading a set of instructions when a computer is first turned on or booted.
- The bootloader or bootstrap program is then loaded to initialize the OS.

bootstrap

- Basically for a computer to start running to get an instance when it is powered up or rebooted it need to have an initial program to run. And this initial program which is known as **bootstrap** needs to be simple. It must initialize all aspects of the system, from CPU registers to device controllers and the contents of the main memory, and then starts the operating system.
- To do this job the bootstrap program basically finds the operating system kernel on disk and then loads the kernel into memory and after this, it jumps to the initial address to begin the operating-system execution.

Ref: https://www.geeksforgeeks.org/boot-block-in-operating-system/

Typical programs that load the OS are:

- NT Loader (NTLDR):

A bootloader for Microsoft's Windows NT OS that usually runs from the hard drive

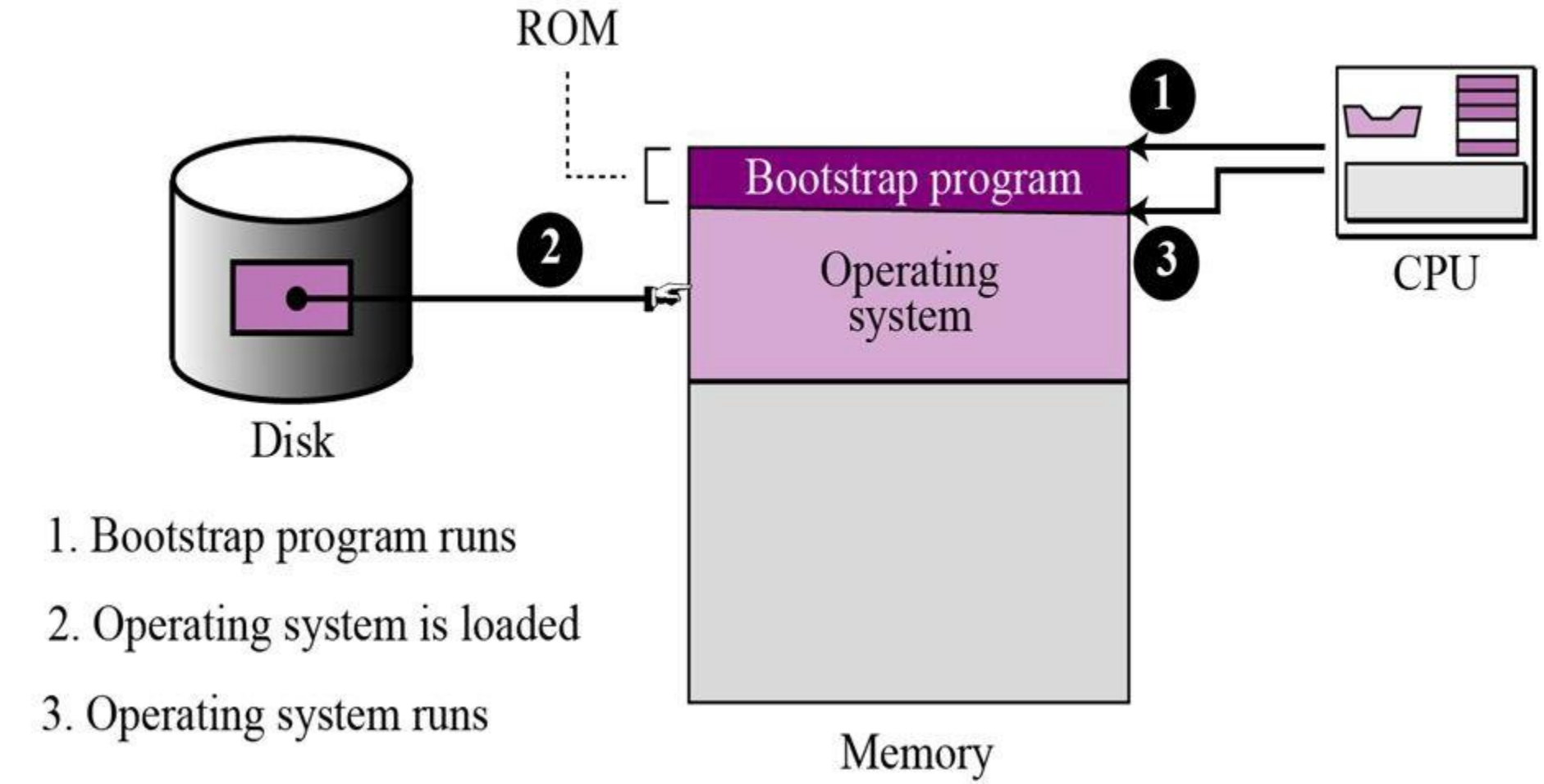
- Linux Loader (LILO):

A bootloader for Linux that generally runs from a hard drive or floppy disc

- GNU Grand Unified Bootloader (GRUB):

A multiboot specification that allows the user to choose one of several OSs

Bootstrap Process



Bootstrapping (cont.)

- Prior to bootstrapping a computer is said to start with a blank main memory.
- The bootstrap allows the sequence of programs to load in order to initiate the OS.
- The OS is the main program that manages all programs that run on a computer and performs tasks

Services Provided by the OS

- **Facilities for Program creation**
  - editors, compilers, linkers, and debuggers
- **Program execution**
  - loading in memory, I/O and file initialization
- **Access to I/O and files**
  - deals with the specifics of I/O and file formats
- **System access**
  - Protection in access to resources and data
  - Resolves conflicts for resource contention

Example using C Programming Language:

```
void main()
 {
   int * ptr;
   *ptr = 5;
 }
```

- in early versions of C under Dos will cause crash the program. So the machine must be rebooted again (set up the operating system “windows” from the beginning)!!

- So we need protection

Regarding the Securety, What is the most secure OS ?

Qubes OS (**Linux based**)is the most secure Operating system software because it isolates users' app programs into distinct Qube entities, giving it a semblance of operating from a **separate virtual machine**.

Is easy to use for those with the Linux experience to use them. It's not a beginner's operating system. But it is a private one.

Ref: https://secureblitz.com/most-secure-operating-systems/
Ref: https://www.youtube.com/watch?v=dmEqrgS6TsU

OS Categories – Usage Types

- Based on this, there are five main categories:
  1. ***Batch***: The batch operating system grouped jobs that perform similar functions. These job groups are processed as a batch and executed simultaneously. This operating system allows a system to execute the following batch processing tasks **(first come, first served)such as : (Old Mainframe and DOS)**
  2. ***Time Sharing***: For systems where many users (or many applications) access common hardware, there could be a need to timeshare the limited resources. The OSs in such cases are categorized as time-sharing OSs. (Windows, Mac, and Linux)
  3. ***Parallel (over tightly coupled systems)***: For hardware that is distributed physically and a single OS needs to coordinate their access, we call these systems distributed OSs. (AIX for IBM RS/6000 and Solaris for workstations)
  4. ***Network – Distributed (loosly coupled)***: Another usage model, similar to the distributed scenario, is when the systems are connected over a network protocol, like IP (Internet Protocol), and therefore referred to as network OSs. (Windows server 2008, Novell Netware)
  5. ***Real Time***: In some cases, we need fine-grained time precision in execution and responsiveness. We call these systems **real-time OSs.**

Ref: https://www.javatpoint.com/types-of-operating-systems

Comparison bet. Time Sharing and Real Time OS

	Time Sharing Operating System	Real-Time Operating System
1.	In it **Switching** method/function is available.	**Switching** method/function is not available.
2.	In it any modification in the program can be possible.	modification does not take place.
3.	In this OS, computer resources are shared	computer resources are not shared.
4.	In this OS, the response is provided to the user within a second.	While in real time OS, the response is provided to the user within time constraint.
5.	In time sharing system, high priority tasks can be preempted by lower priority tasks, making it impossible to guarantee a response time for your critical applications.	Real time operating systems, give users the ability to prioritize tasks so that the most critical task can always take control of the process when needed.

Ref: https://www.geeksforgeeks.org/difference-between-time-sharing-os-and-real-time-os/

Loosely and Tightly Coupled Multiprocessor System (Comparison)

	LOOSELY COUPLED MULTIPROCESSOR SYSTEM	TIGHTLY COUPLED MULTIPROCESSOR SYSTEM
memory	Each processor has its own memory module.	Processors have shared memory modules.
Efficient	Efficient when tasks running on different processors, has minimal interaction.	Efficient for high-speed or real-time processing.
Memory conflict	It generally, do not encounter memory conflict.	It experiences more memory conflicts.
Data rate	Low.	High.
Expensive	Less expensive.	More expensive.

Ref: https://techdifferences.com/difference-between-loosely-coupled-and-tightly-coupled-multiprocessor-system.html

Which OS used in Embedded System Domain ?

- **Linux and Android are two powerful operating systems** used in most of the embedded systems today.

Selection between the two will depend completely on the usage and requirement. For instance – If you want better wireless connectivity and graphics interface, you might consider Android OS over Linux.

Linux OS can do everything that Android OS can do. However, Linux comes with a complex flow and it might be difficult for a beginner to understand it. But, it can produce better results — once understood.

Device controller & device driver

- – A **device driver** **is a code** inside the OS that allows to be empowered with the specific commands needed to operate the associated device.

The code is implemented by the device manufacturer which allows the device to communicate with the computer's OS. Without device drivers, the computer won't be able to able to communicate properly with the hardware devices.

- **Device controller**, on the other hand, is like a bridge between the device and the operating system. It is an **electronic component** consisting of chips which control the device.

http://www.differencebetween.net/technology/difference-between-device-driver-and-device-controller/

Why Is It Important to Know About the OS?

- Software ***developers must have a good understanding of the environment, the OS, that their code is running in***, or they won't be able to achieve the things they want with their program.
- As you, a software developer, go through the stages of development, it is important for you to keep in mind the OS interfaces and functionality as this will impact the software being developed.
- For Example:
  - the choice of **language** and needed runtime features may be OS dependent.
  - the choice of inter-process communication **(IPC)** protocols used for messaging between applications will depend on the OS offerings
- During development and **debug**, there could be usages where the developer may need to understand and interact with the OS. For example, debugging a **slowly performing or nonresponsive** application may require some understanding of how the OS performs input/output operations.

Why Is It Important to Know About the OS?

- some questions that may come up during the debug:
  - Are you accessing the file system too often and writing repeatedly to the disk?
  - Is there a garbage collector in place by the software framework/SDK?
  - Is the application holding physical memory information for too long?
  - Is the application frequently creating and swapping pages in memory? What it the average commit size and page swap rate?
  - Is there any other system event such as power event, upgrades, or virus scanning that could have affected performance?
  - Is there an impact on the application based on the scheduling policy, application priority, and utilization levels?
- If the application needs to interface with a custom device, it will most likely need to interface some low-level functionality provided by the OS using the OS-provided API for communication.
- As a software developer, it may be required to understand these APIs and leverage the OS capabilities. There could also be a need to follow certain standard protocols provided by the OS for authenticating a given user of your application to grant permissions and access.

Responsibilities of an OS

- The OS needs to be able to abstract the complexities of the underlying hardware, support multiple users, and facilitate execution of multiple applications at the same time. The following table describe the Requirements and Solutions

Requirements	Solution
Applications require time on the **CPU** to execute their instructions.	The OS shall implement and abstract this using suitable **scheduling** algorithms.
Applications require access to system **memory** for variable storage and to perform calculations based on values in memory.	The OS shall implement **memory management** and provide APIs for applications to utilize this memory.
Each software may need to access different **devices** on the platform.	The OS may provide APIs for device and I/O management and interfaces through which these devices can be communicated.

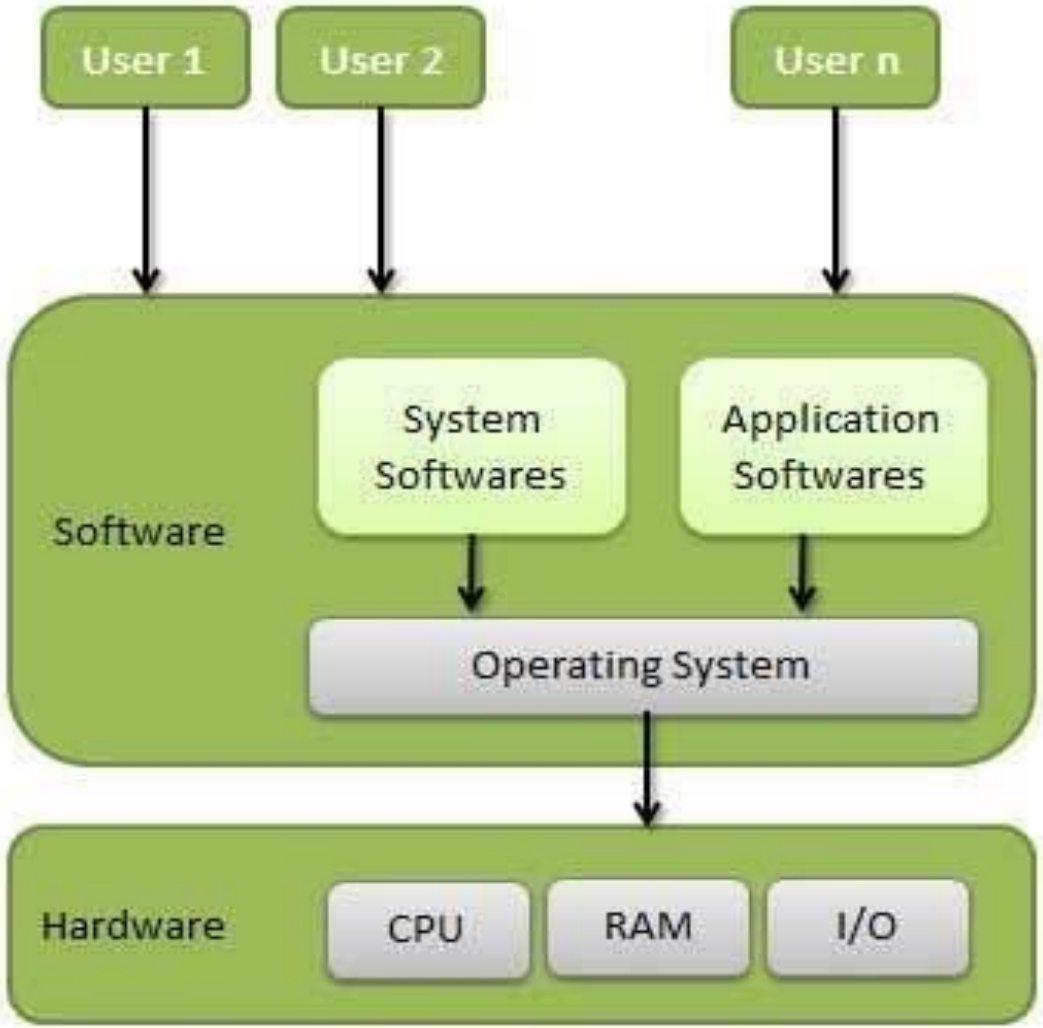
Responsibilities of an OS (cont.)

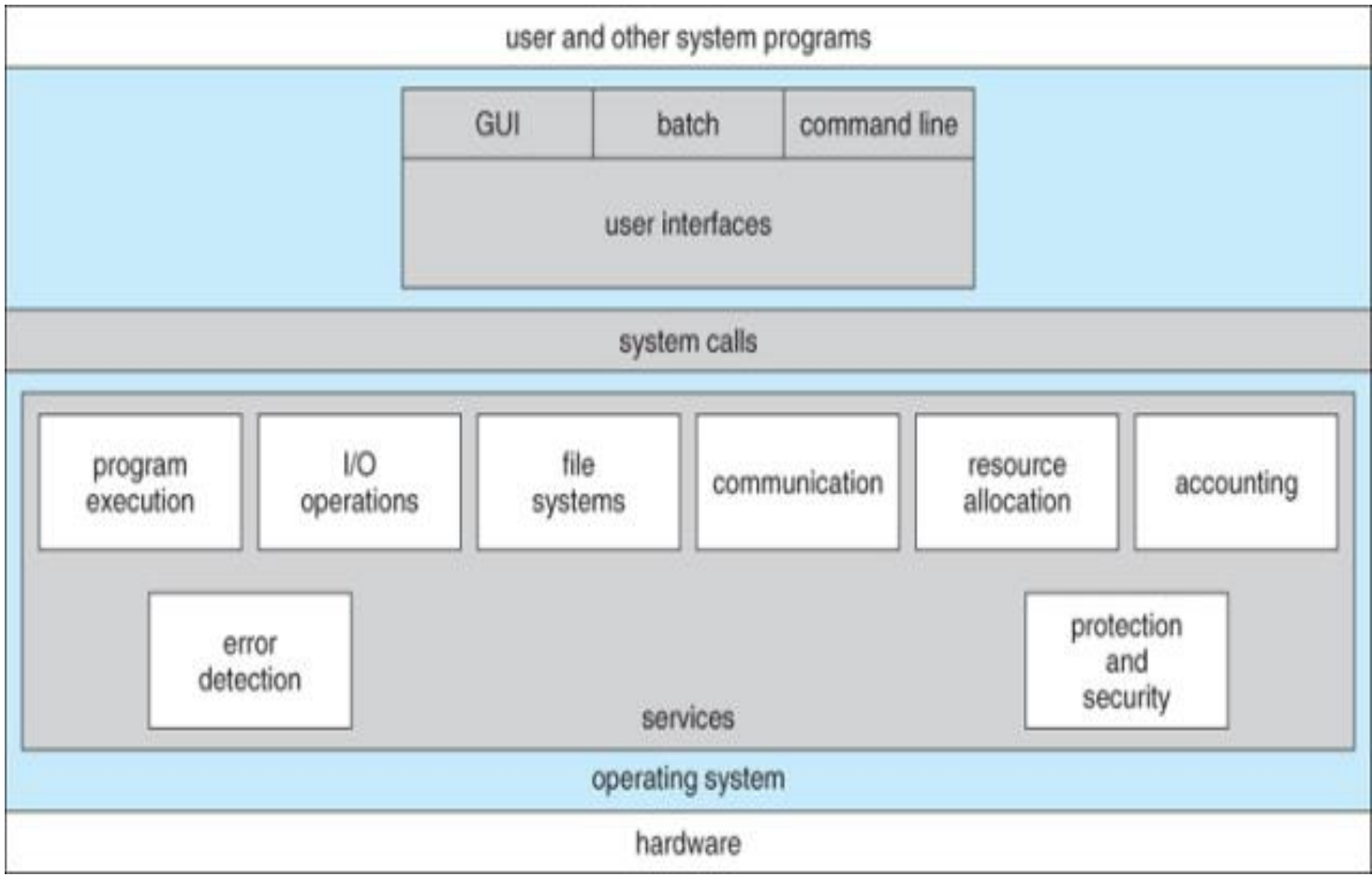
Requirements	Solution
There may be a need for the user or applications to save and read back contents from the **storage**.	Most OSs have a **directory and file system** that handles the storage and retrieval of contents on the disk.
It is important to perform all of the core operations listed in the preceding **securely** and efficiently.	Most OSs have a **security subsystem** that meets specific security requirements, virtualizations, and controls and balances.
Ease of access and usability of the system.	The OS may also have an additional GUI (graphical user interface) in place to make it easy to use, access, and work with the system.

Responsibilities of an OS (cont.)

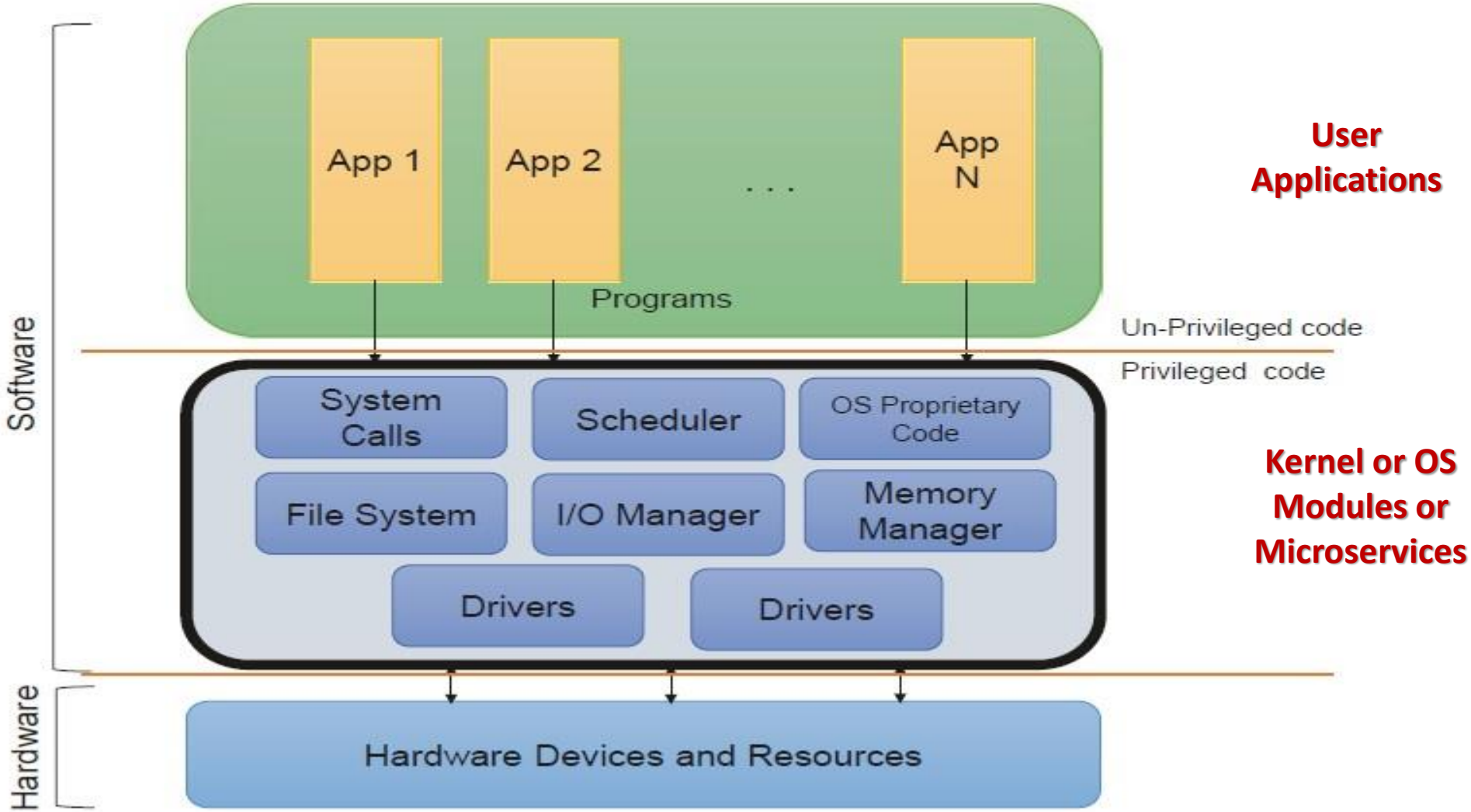
- To summarize, the OS performs different functions and handles multiple responsibilities for software to co-exist, streamlining access to resources, and enabling users to perform actions. They are broadly classified into the following functional areas:
  - Scheduling
  - Memory management
  - I/O and resource management
  - Access and protection
  - File systems
  - User interface/shell

Services Provided by the OS





Operating System Components



User Applications

Kernel or OS Modules or Microservices

What is an Operating System?

- Most OSs typically have at least two main pieces:
  - There is a core part that handles the complex, low-level functionalities and is typically referred to as the ***kernel*** *and* must be running at all times – resident in the main memory.
  - There are generally some *libraries*, *applications*, and *tools* that are shipped with the OS. For example, there could be browsers, custom features, frameworks, and OS-native applications that are bundled together.

Kernel

The core of all operating systems is called a kernel. The kernel provides the most basic interface between the computer itself and the rest of the operating system. The kernel is responsible for the management of the central processor.

Some of the main functions that kernel performs:

- switching between programs
- hardware device control and programming
- memory management
- process management
- scheduling (deciding what programs to run)
- inter-process communication

Utility programs -- system software -- APIs

A **utility program** is a type of **system software** that is used for effective management

Most operating system include different built in utility programs.

Types of utility programs:

- **File Manager** (to manage and view files in computer system)
- **Task manager utility**
- **Uninstaller**
- **File compressor**
- **Disk scanner**

Example from Utility programs (File Manager)

File Manager

- File manager is used to manage and view files in computer system.
- All operating systems provide file viewers. Windows explorer is an example of file viewer.
- Some important functions of file manager are as follows.
- Displaying a list of files
- Organizing files in folders
- Copying ,renaming,deleting,moving and sorting files and folders
- Creating shortcuts

Processes and Scheduling

Content

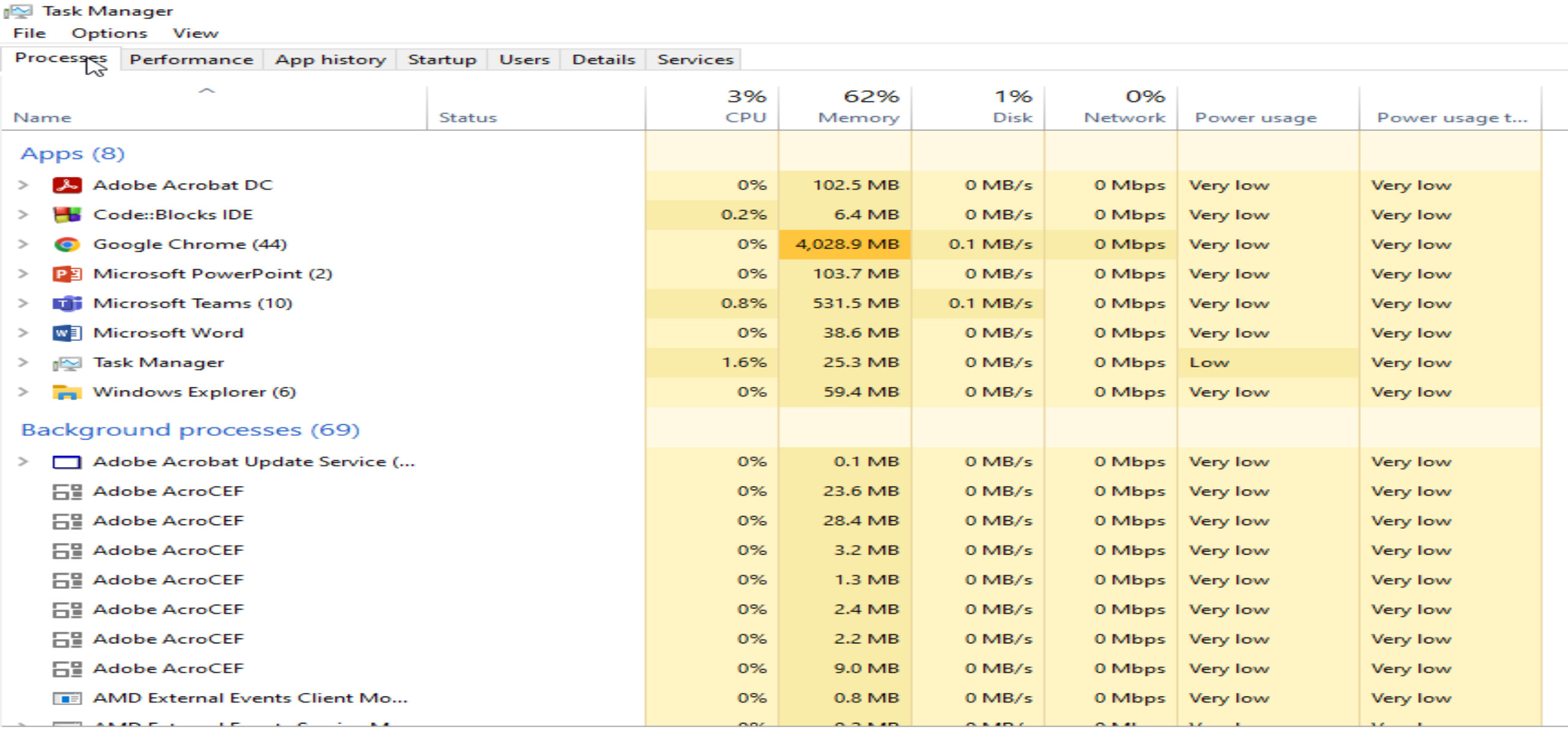
- Introduction to Scheduling
- Process Concept
- Process Contents
- Process States
- Process Control Block (PCB)
- Context Switching
- Scheduling

Course Outlines

Program and Process Concept

- When a software developer builds a solution, the set of capabilities it provides is usually static and embedded in the form of processed code that is built for the OS. This is typically referred to as the program.
- When the program gets triggered to run, the OS assigns a process ID and other metrics for tracking.
- At the highest level, an executing program is tracked as a process in the OS.
- Note that in the context of different operating systems, jobs and processes may be used interchangeably. However, ***process refer to a program in execution***.

Program and Process Concept



Introduction to Scheduling

- One of the primary functionalities of the OS would be to provide the ability to run multiple, concurrent applications on the system and efficiently manage their access to system resources.
- As many programs try to run in parallel, there may be competing and conflicting requests to access hardware resources such as CPU, memory, and other devices.
- The operating system streamlines these requests and orchestrates the execution at runtime by scheduling the execution and subsequent requests to avoid conflicts.

scheduling algorithms (Examples) - FCFS

Process	Burst Time	Arrival Time
P1	20	0
P2	12	3
P3	4	2
P4	9	5

Waiting time : start time – arrival time

P1 = 0 – 0 = 0

P2 = 24 – 3 = 21

P3 = 20 – 2 = 18

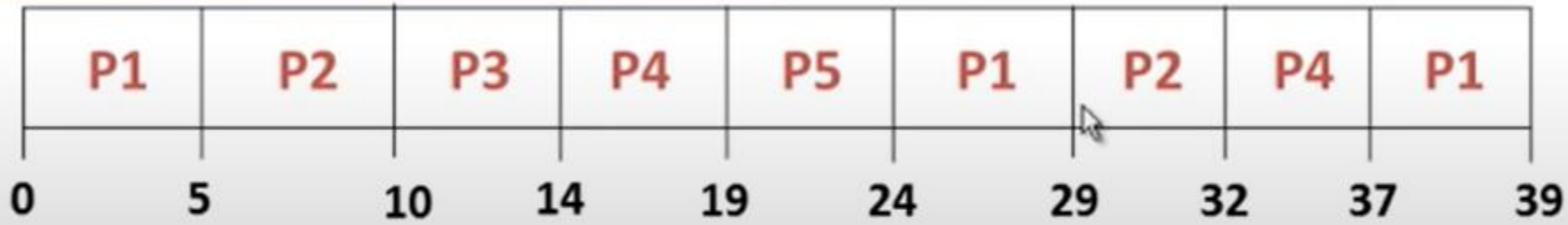
P4 = 36 – 5 = 31

scheduling algorithms (Examples) - RR

Process	Burst Time
P1	12 7 2
P2	8 3
P3	4
P4	10 5
P5	5

Waiting time :

P1 = 0 + (24 - 5) + (37 - 29) = 27
P2 = 5 + (29 - 10) = 24
P3 = 10
P4 = 14 + (32 - 19) = 27
P5 = 19



Average waiting time = (27 + 24 + 10 + 27 + 19) / 5 = 107 / 5 = 21.4

scheduling algorithms (Examples) - SJF

Process	Burst Time	Arrival Time
P2	12	0
P3	8	3
P4	4	5
P1	10	10
P5	6	12

Waiting time : start time – arrival time

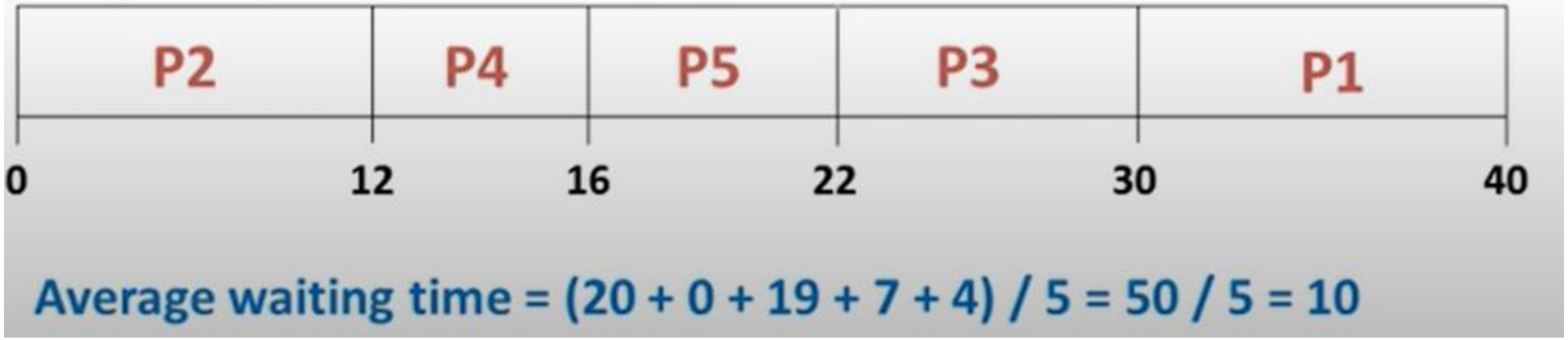
P1 = 30 – 10 = 20

P2 = 0 – 0 = 0

P3 = 22 – 3 = 19

P4 = 12 – 5 = 7

P5 = 16 – 12 = 4



Average waiting time = (20 + 0 + 19 + 7 + 4) / 5 = 50 / 5 = 10

scheduling algorithms (Examples) – SJF preemptive

Process	Burst Time	Arrival Time
P2	12 9	0
P3	8 6	3
P4	4	5
P1	10	10
P5	6	12

Waiting time : start time – arrival time

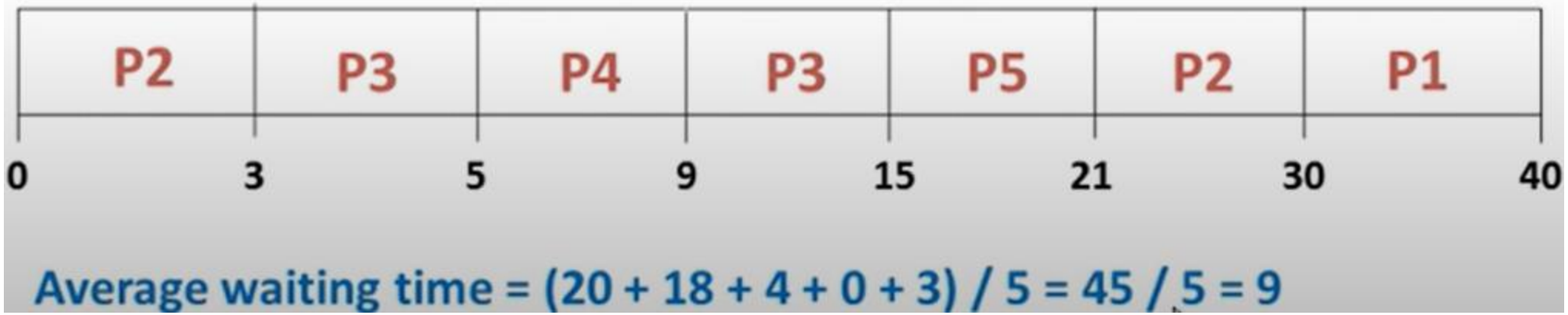
P1 = 30 – 10 = 20

P2 = (0 – 0) + (21 - 3) = 18

P3 = (3 – 3) + (9 - 5) = 4

P4 = (5 – 5) = 0

P5 = 15 – 12 = 3



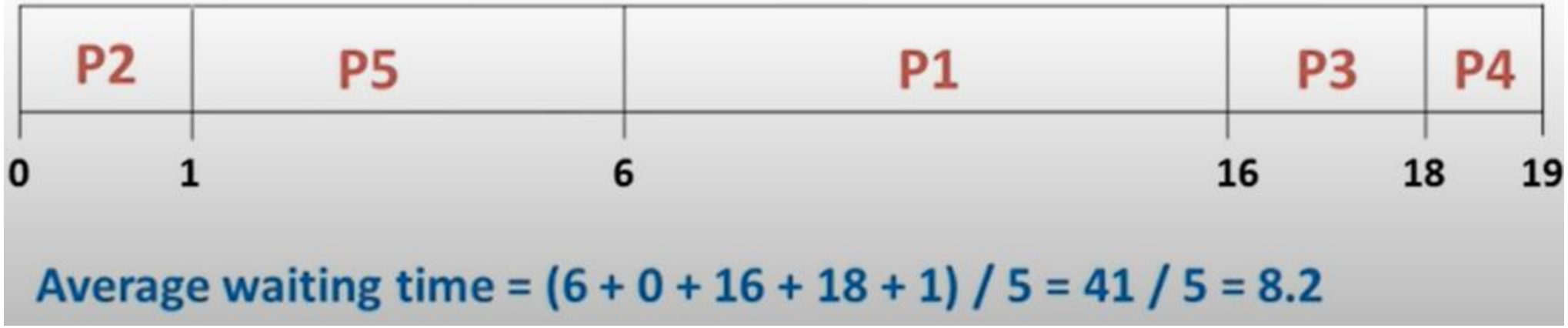
Average waiting time = (20 + 18 + 4 + 0 + 3) / 5 = 45 / 5 = 9

scheduling algorithms (Examples) – Priority Non- preemptive

Process	Burst Time	Priority	Arrival Time
P1	10	3	All Processes Arrived at The Same Time
P2	1	1	
P3	2	4	
P4	1	5	
P5	5	2	

Waiting time :
start time – arrival time

P1 = 6
P2 = 0
P3 = 16
P4 = 18
P5 = 1



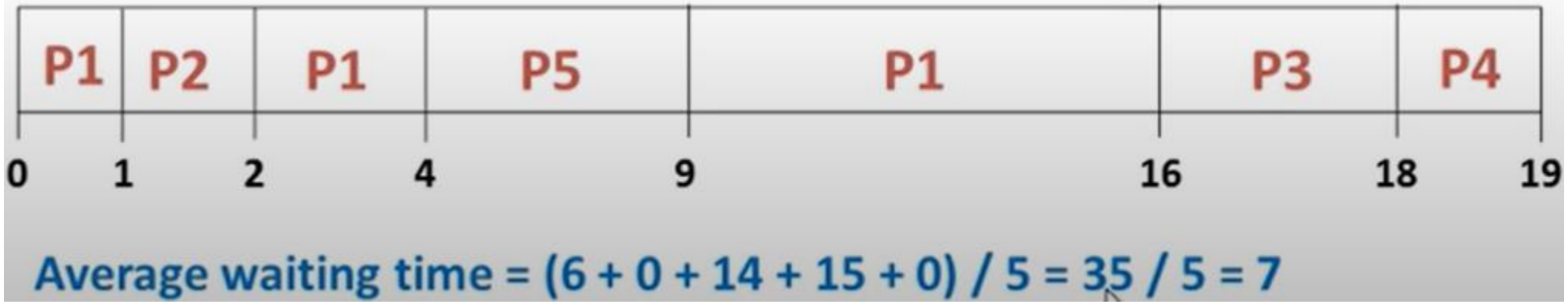
Average waiting time = (6 + 0 + 16 + 18 + 1) / 5 = 41 / 5 = 8.2

scheduling algorithms (Examples) – Priority preemptive

Process	Burst Time	Priority	Arrival Time
~~P1~~	~~10~~ 9 7	~~3~~	~~0.0~~
P2	1	1	1.0
P3	2	4	2.0
P4	1	5	3.0
P5	5	2	4.0

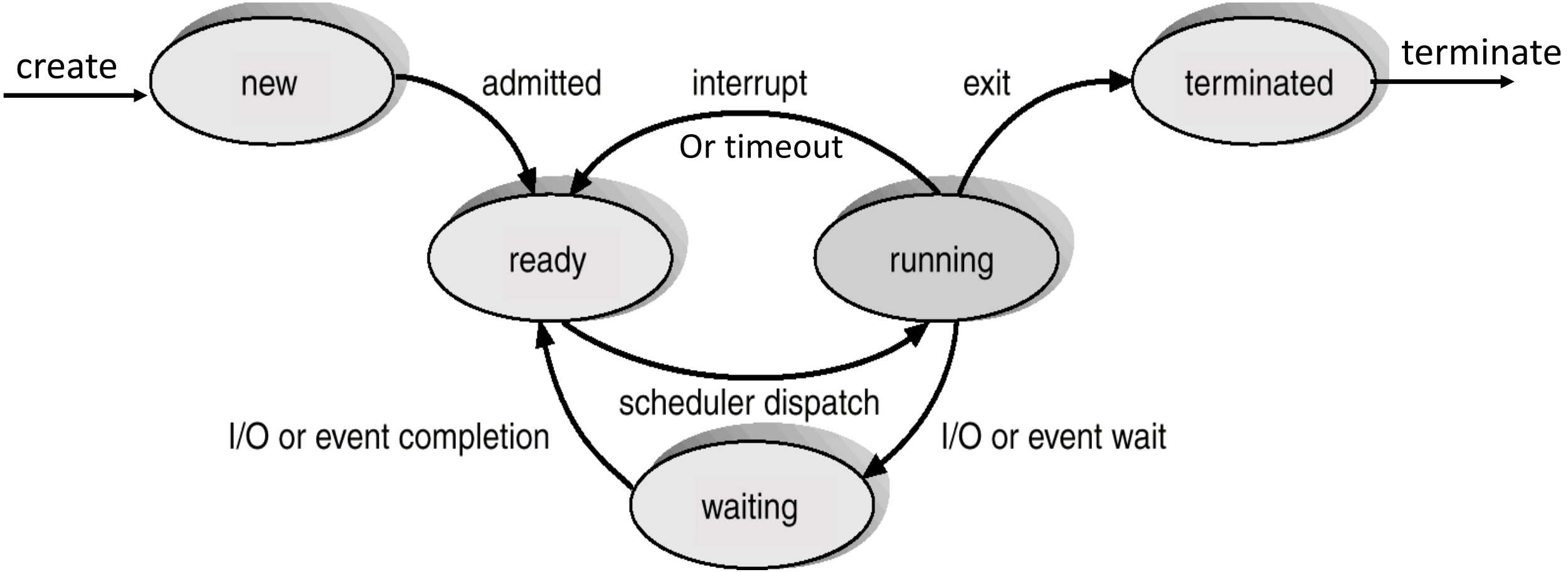
Waiting time :
start time – arrival time

P1 = (0 - 0)+(2 - 1)+(9 - 4) = 6
P2 = 1 – 1 = 0
P3 = 16 – 2 = 14
P4 = 18 – 3 = 15
P5 = 4 – 4 = 0



Average waiting time = (6 + 0 + 14 + 15 + 0) / 5 = 35 / 5 = 7

Process States



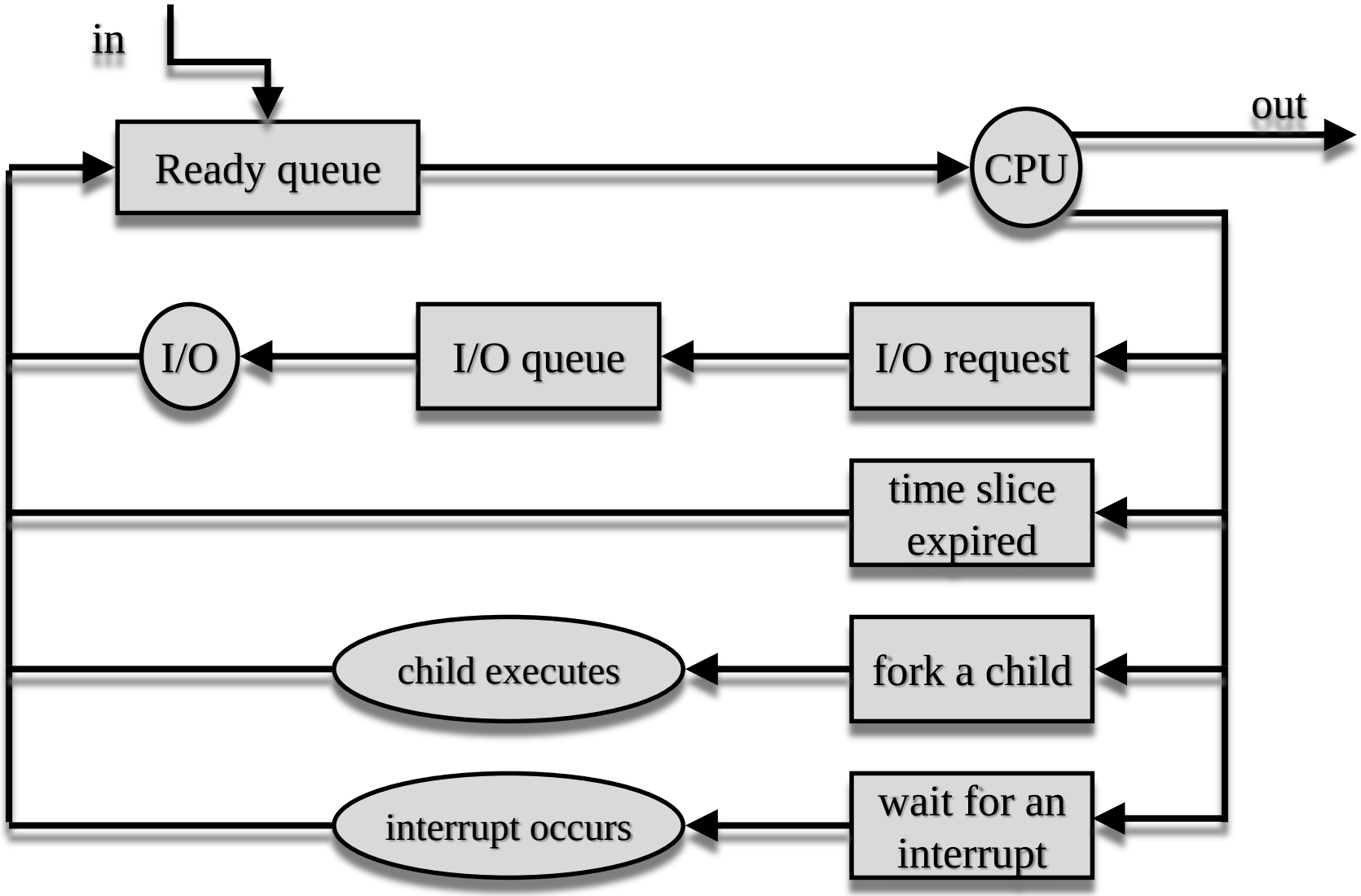
Process States

- When a program gets triggered for execution, typically say using a double click of the EXE (or using a CreateProcess() API in Windows), a new process is created.
- process typically supports multiple states of readiness in its lifecycle:
  - **New**: The process is being created.
  - **Running**: Instructions are being executed.
  - **Waiting**: The process is waiting for some event to occur.
  - **Ready**: The process is waiting to be assigned to a processor.
  - **Terminated or Exit**: The process has finished execution.
- There could be more than one CPU core on the system and hence the OS could schedule on any of the available cores.
- In order to avoid switching of context between CPU cores every time, the OS tries to limit such frequent transitions.
- The OS monitors and manages the transition of these states seamlessly and maintains the states of all such processes running on the system.

Scheduling

- The most frequent process states are the Ready, Waiting, and Running states. The operating system will receive requests to run multiple processes at the same time and may need to streamline the execution. It uses process scheduling queues to perform this:
  1. **Ready Queue**: When a new process is created, it transitions from New to the Ready state. It enters this queue indicating that it is ready to be scheduled.
  2. **Waiting Queue**: When a process gets blocked by a dependent I/O or device or needs to be suspended temporarily, it moves to the Blocked state since it is waiting for a resource. At this point, the OS pushes such process to the Waiting queue.
  3. **Job queue** that maintains all the processes in the system at any point in time. This is usually needed for bookkeeping purposes.

Scheduling Queues



Scheduling Criteria

- Some of the typical metrics that the OS may use to determine scheduling priorities are listed in the following:
  - **CPU Utilization and Execution Runtime**: The total amount of time the process is making use of the CPU excluding NOP (no-operation) idle cycles.
  - **Volume/Execution Throughput**: Some OSs may need to support certain execution rates for a given duration.
  - **Responsiveness**: The time taken for completion of a process and the average time spent in different queues.
  - **Resource Waiting Time**: The average time taken on external I/Os on the system.

- ***Note*** *Most OSs try to ensure there is fairness and liveness in scheduling. There are various scheduling algorithms like First Come, First Serve (FCFS), Shortest Job First (SJF), Shortest Remaining Time First (SRT F), Round-Robin, Static/Dynamic Priority, and so on that the OS uses for scheduling of processes.*

Context Switching

- The operating system may need to swap the currently executing process with another process to allow other applications to run, it does so with the help of context switching.
- When a process is executing on the CPU, the process context is determined by the program counter (instruction currently run), the processor status, register states, and various other metrics.
- When the OS needs to swap a currently executing process with another process, it must do the following steps:
  1. Pause the currently executing process and save the context.
  2. Switch to the new process.
  3. When starting a new process, the OS must set the context appropriately for that process.
- This ensures that the process executes exactly from where it was swapped.

Process Control Block (PCB)

Pointer	Process state
Priority	
Program counter	
CPU registers	
Memory management info	
I/O status information	
Accounting Information	

Ref: https://www.javatpoint.com/os-attributes-of-a-process

Process Control Block (PCB)

- The **process ID** is a unique identifier for the instance of the process that is to be created or currently running.
- The **process state** determines the current state of the process, described in the preceding section.
- The **pointer** could refer to the hierarchy of processes (e.g., if there was a parent process that triggered this process).
- The **priority** refers to the priority level (e.g., high, medium, low, critical, real time, etc.) that the OS may need to use to determine the scheduling.
- **Affinity and CPU register** details include if there is a need to run a process on a specific core. It may also hold other register and memory details that are needed to execute the process.

Process Control Block (PCB)

- The **program counter** usually refers to the next instruction that needs to be run.
- The **I/O status information,** like which devices assigned, limits, and so on that is used to monitor each process is also included in the structure.
- The **accounting information** such as paging requirements from memory, timers, how many time unit remaining to finish, … etc

- There could be some modifications to how the PCB looks on different OSs. However, most of the preceding are commonly represented in the PCB.

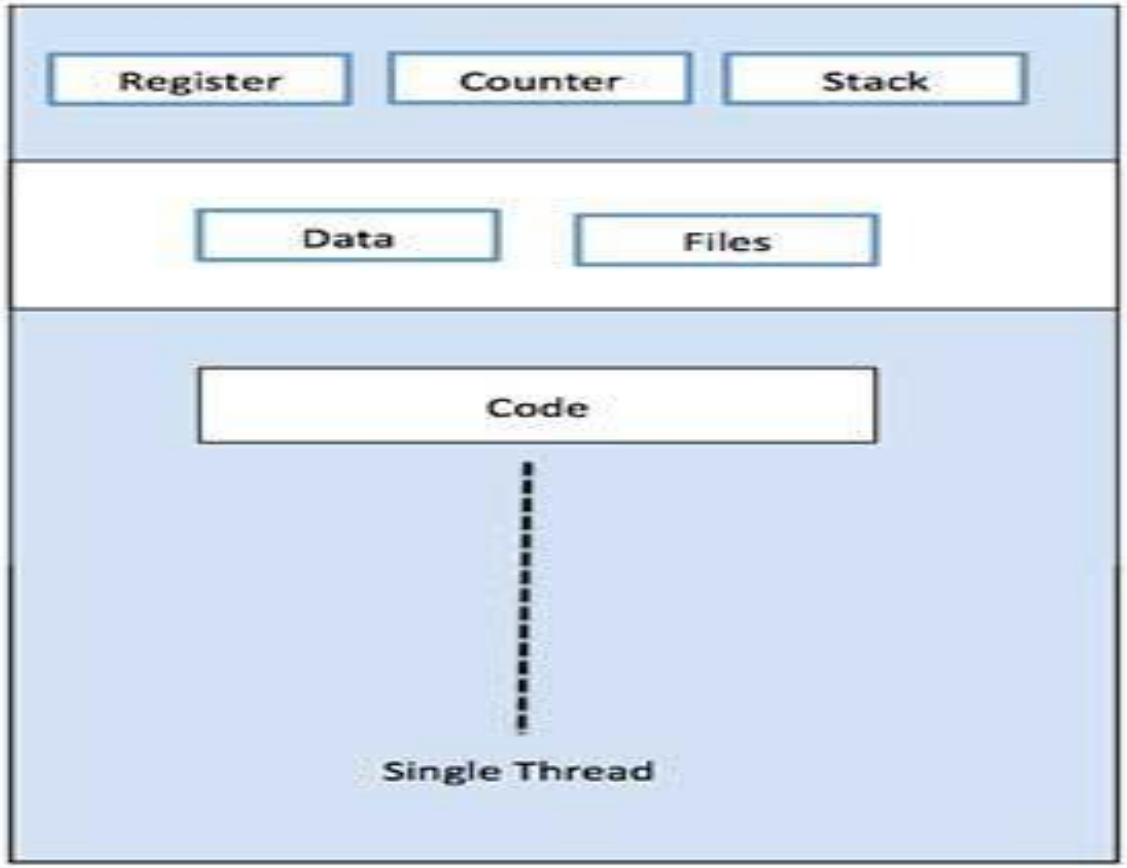
What is Thread?

- A thread is also called a **lightweight process**. Threads provide a way to improve application performance through parallelism.

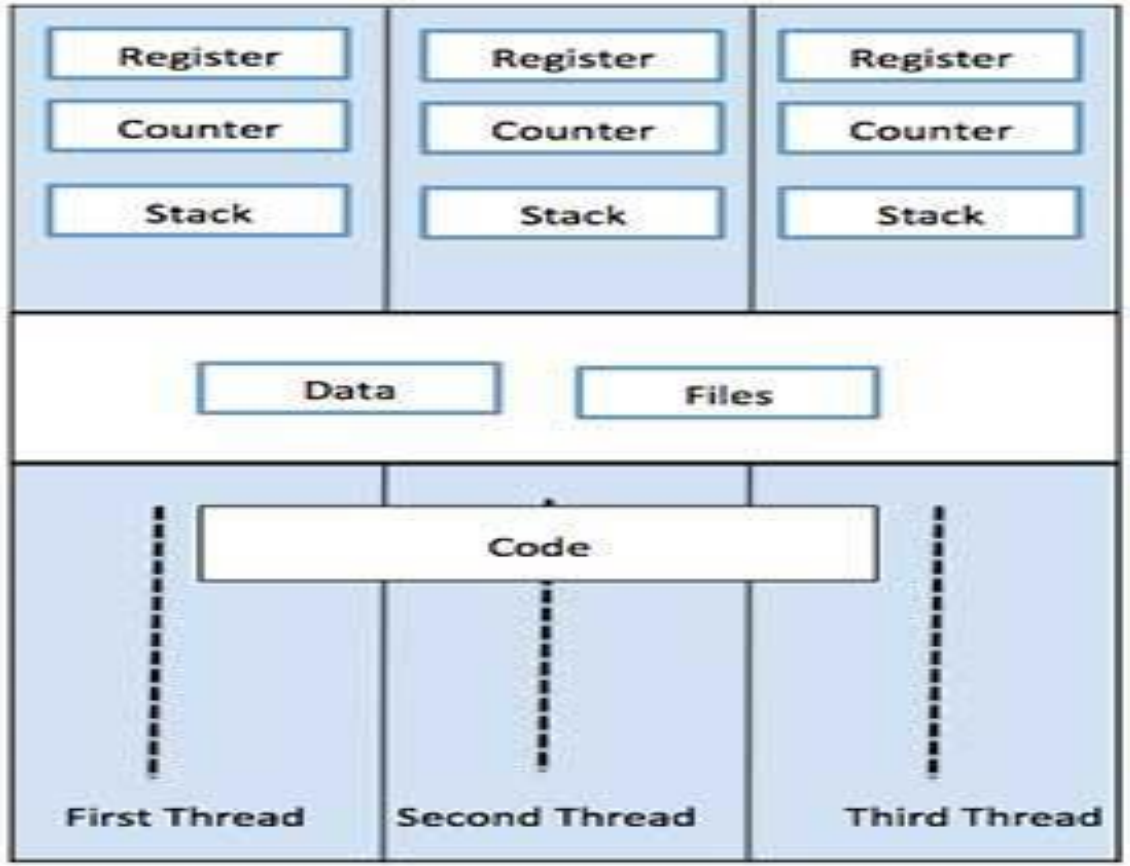
- When a process gets executed, **it could create one or more threads** internally that can be executed on the processor.

- **Each thread belongs to exactly one process** and **no thread can exist outside a process**. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server.

single-threaded & a multithreaded process.



Single Process P with single thread



Single Process P with three threads

Thread Concepts

- Examples:
  - Chatting program: the sending and receiving operations are independent, using threads
  - Strategic games: there are many actions happened at the same time independently, using threads

Comparison between Process & Thread

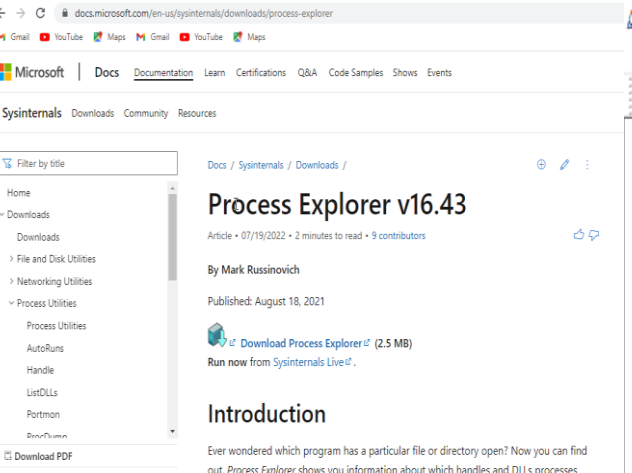
S/N	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's

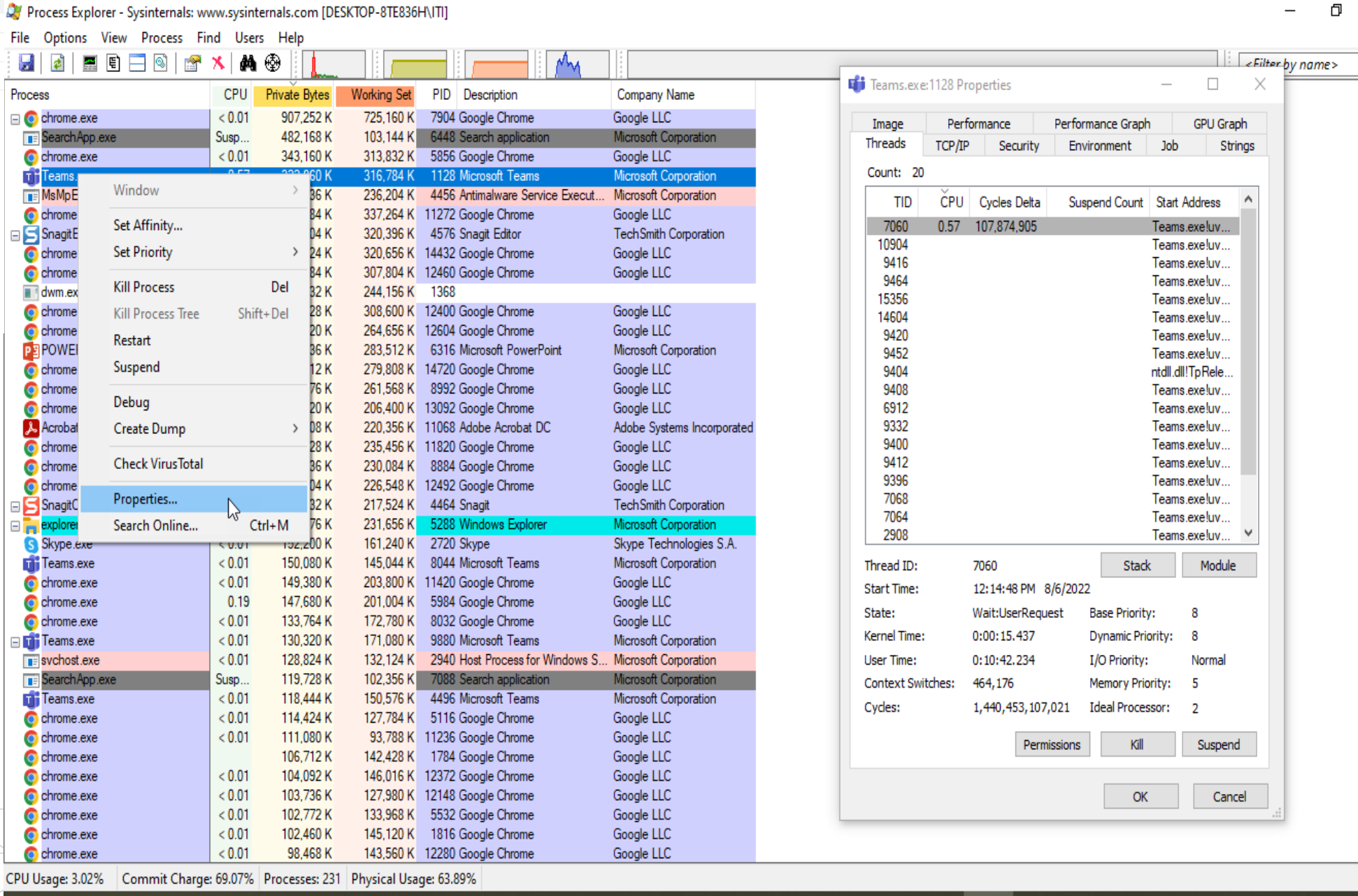
Benefits of Threads

- **Effective Utilization of Multiprocessor system:** When you have more than one thread in one process, you can schedule more than one thread in more than one processor.
- **Faster context switch:** The context switching period between threads is less than the process context switching. The process context switch means more overhead for the CPU.
- **Enhanced throughput of the system:** When the process is split into many threads, and each thread is treated as a job, the number of jobs done in the unit time increases. That is why the throughput of the system also increases.
- **Communication:** Multiple-thread communication is simpler than process communication because the threads share the same address space, while in process.
- **Resource sharing:** Resources can be shared between all threads within a process, such as code, data, and files.

Ref: https://www.javatpoint.com/threads-in-operating-system

Thread Concepts

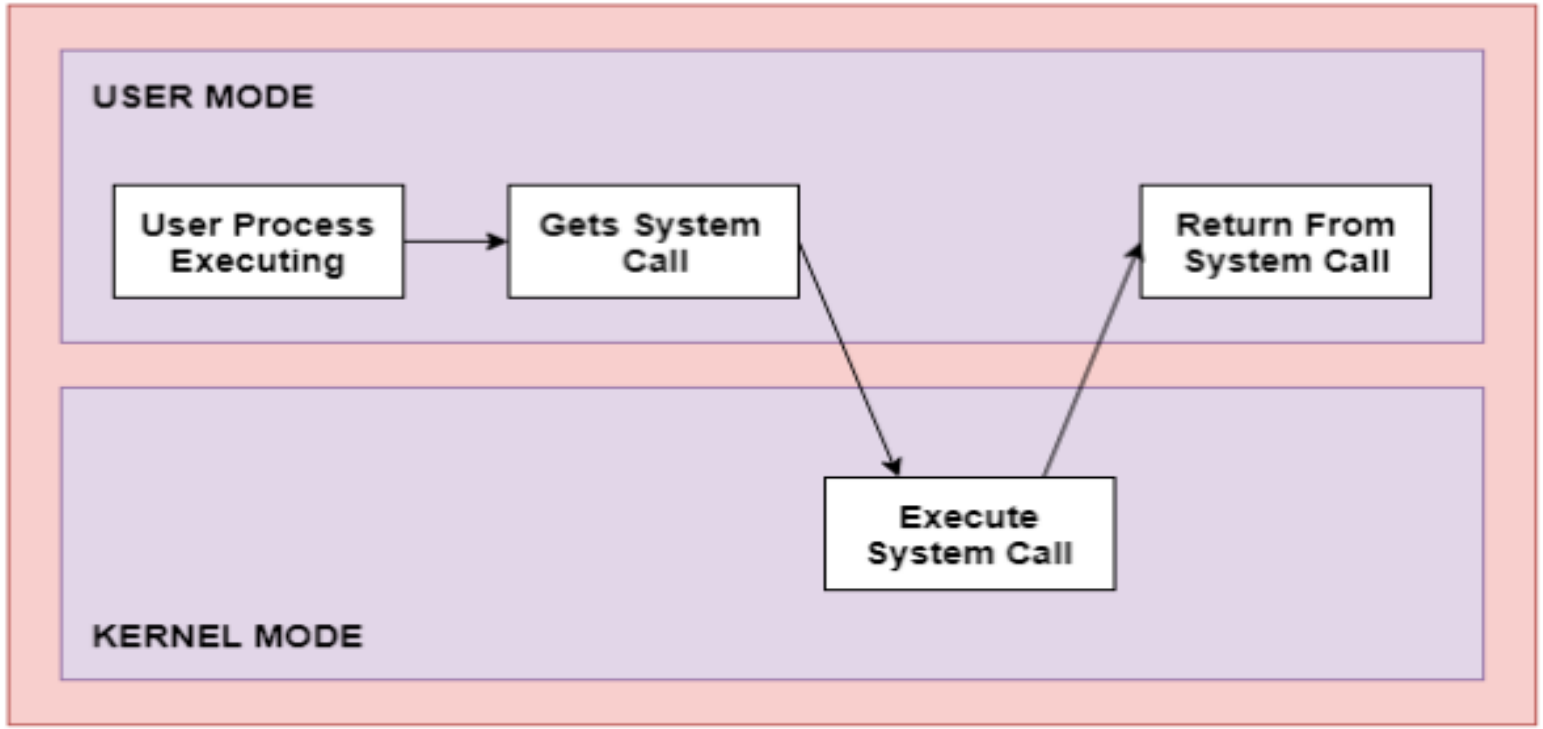




Thread Concepts

- *The OS may employ different types of threads, depending on whether they are run (**user-mode** threads), or (**kernel-mode** threads).*

Execution of the system call



the processes execute normally in the user mode until a system call interrupts this. Then the system call is executed in the kernel mode. After the execution of the system call, the control returns to the user mode and execution of user processes can be resumed.

User Mode vs Kernel Mode

- There are two modes of operation in the operating system to make sure it works correctly. These are **user mode** and **kernel mode**.

- **User mode** − Cannot access any hardware resources, which perform only the user operations.
- **Kernel-mode** − Can access hardware resources like RAM, Printer.

The system is in user mode when the operating system is running a user application such as handling a text editor. The transition from user mode to **kernel mode** occurs when the application requests the help of operating system or an interrupt or a **system call occurs**.

Ref: https://www.tutorialspoint.com/how-are-system-calls-connected-to-the-operating-system

SYSTEM CALL

The interface between a process and an operating system is provided by **system calls**.

In general, system calls are available as assembly language instructions. They are also included in the manuals used by the assembly level programmers.

System calls are usually made when a process (in user mode) requires access to a resource. Then it requests the kernel to provide the resource via a system call.

execution of the system call

Difference between 32-bit and 64-bit operating systems

- there are two types of processors existing, i.e., 32-bit and 64-bit processors.
- *A 32-bit system can access 2^{32} different memory addresses, i.e 4 GB of RAM or physical memory.*
- *A 64-bit system can access 2^{64} different memory addresses, i.e actually 18-Quintillion bytes* (18 exabytes) *of RAM.*
- A computer with a 64-bit processor can have a 64-bit or 32-bit version of an operating system installed. However, with a 32-bit operating system, the 64-bit processor would not run at its full capability.

Ref: https://www.geeksforgeeks.org/difference-32-bit-64-bit-operating-systems/?ref=lbp

Ref: https://www.javatpoint.com/32-bit-vs-64-bit-operating-system

Thank You