Operating Systems

Lecture 2

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.

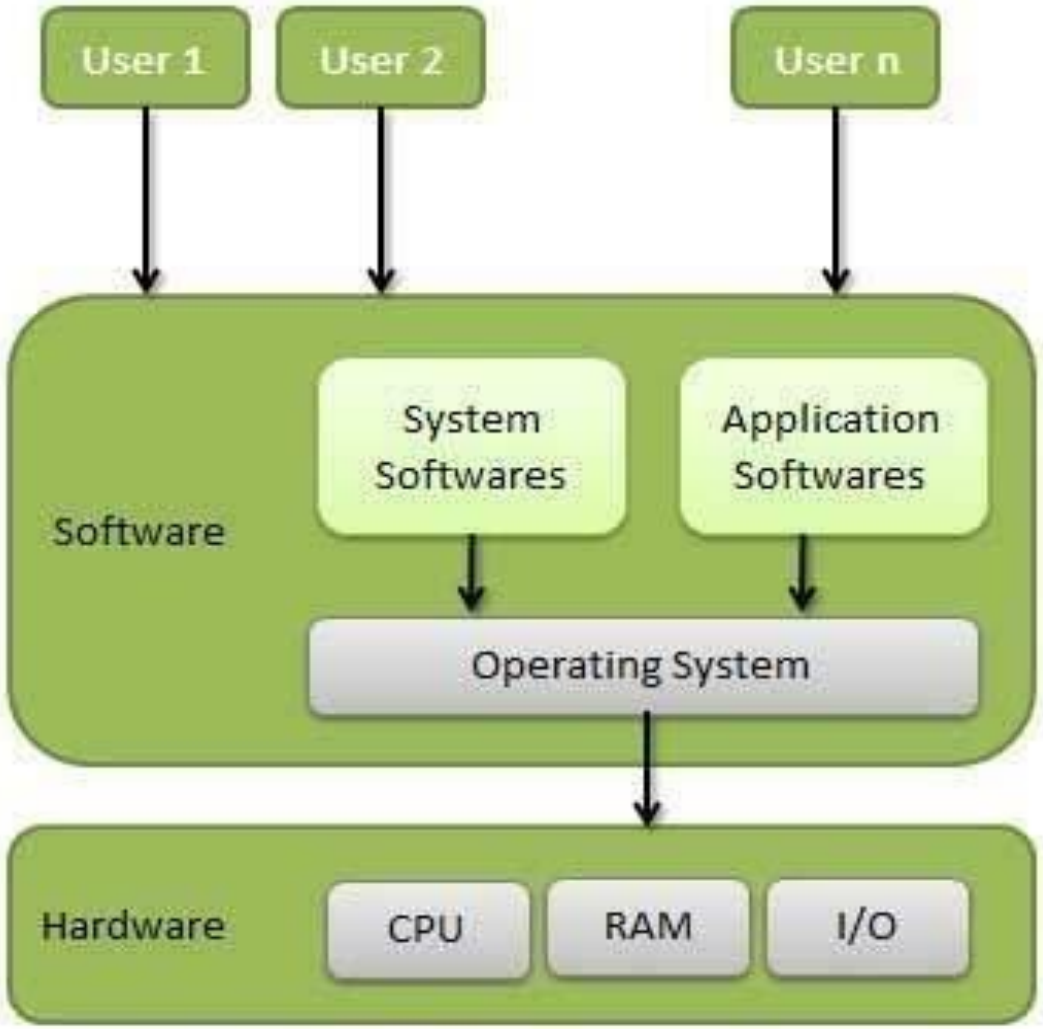
Responsibilities of an OS (cont.)

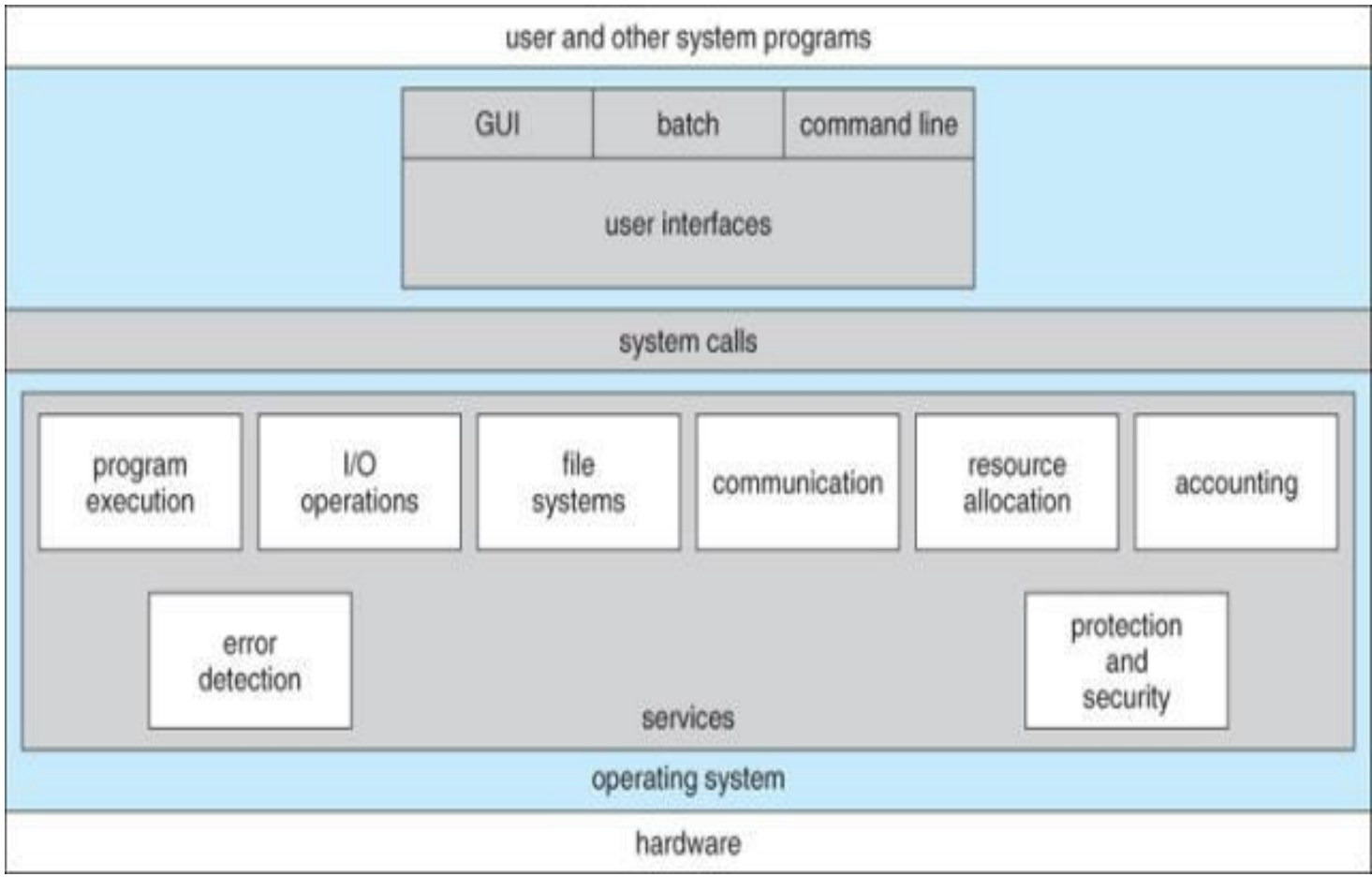
Requirements	Solution
There may be a need for the user or applications to save and read back contents from the **storage**.	Most OSs have a **directory and file system** that handles the storage and retrieval of contents on the disk.
It is important to perform all of the core operations listed in the 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





Services Provided by the OS

- One set of operating-system services provides functions that are helpful to the user:

  - **User interface** - Almost all operating systems have a user interface (UI).
    - Varies between **Command-Line (CLI)**, **Graphics User Interface (GUI)**, **Batch.**

  - **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error).

  - **I/O operations** - A running program may require I/O, which may involve a file or an I/O device.

Services Provided by the OS

- One set of operating-system services provides functions that are helpful to the user:
  - **File-system manipulation** – The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.
  - **Communications** – Processes may exchange information, on the same computer or between computers over a network.
    - Communications may be via shared memory or through message passing (packets moved by the OS).

Services Provided by the OS

- One set of operating-system services provides functions that are helpful to the user:
  - **Error detection** – OS needs to be constantly aware of possible errors.
    - May occur in the CPU and memory hardware, in I/O devices, in user program.
    - For each type of error, OS should take the appropriate action to ensure correct and consistent computing.
    - Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system.

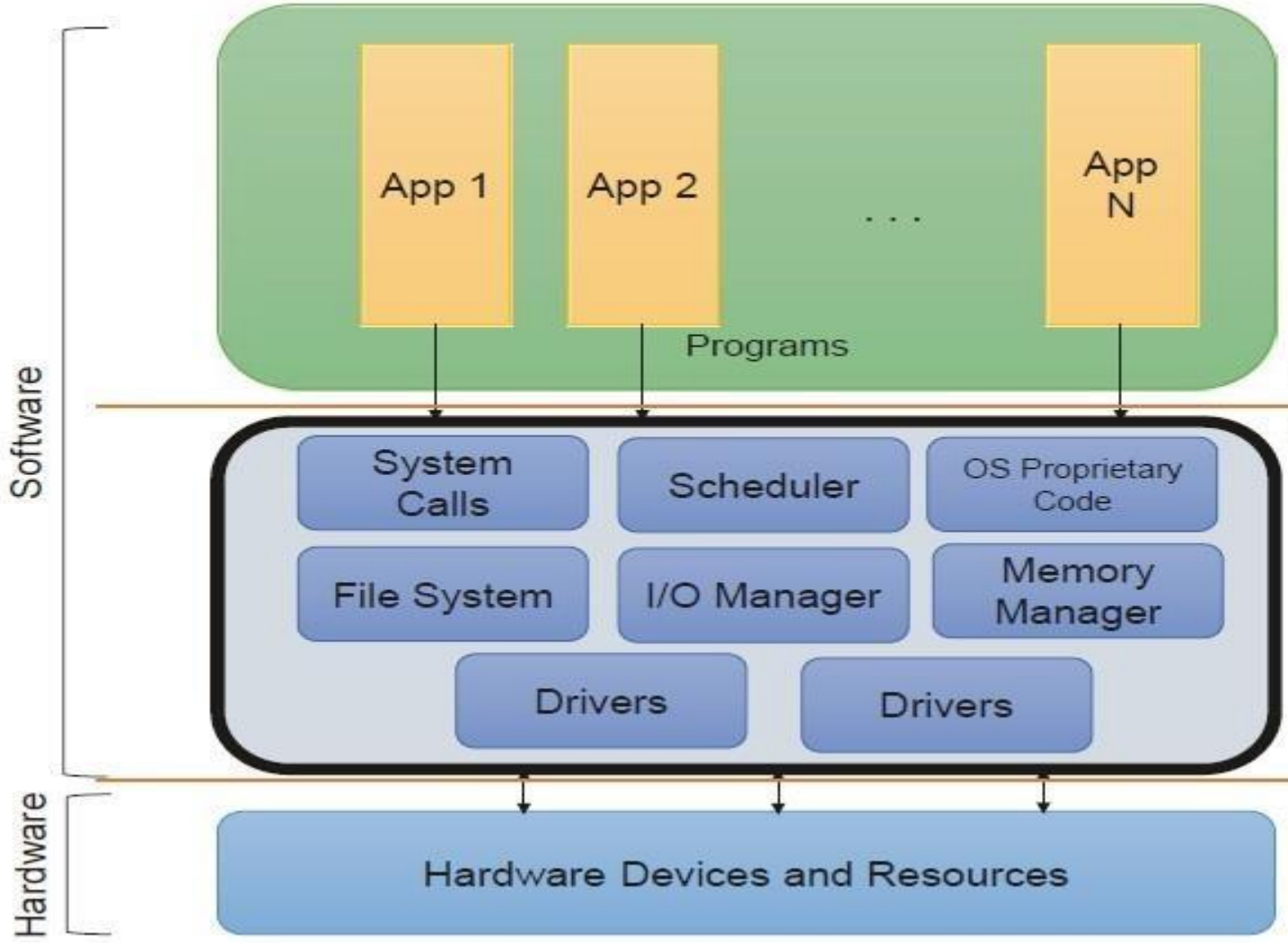
Services Provided by the OS

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing: (1/2)
  - **Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them.
    - Many types of resources - CPU cycles, main memory, file storage, I/O devices.
  - **Accounting** - To keep track of which users use how much and what kinds of computer resources.

Services Provided by the OS

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing: (2/2)
  - **Protection and security** - The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other.
    - **Protection** involves ensuring that all access to system resources is controlled.
    - **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts.

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***.

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

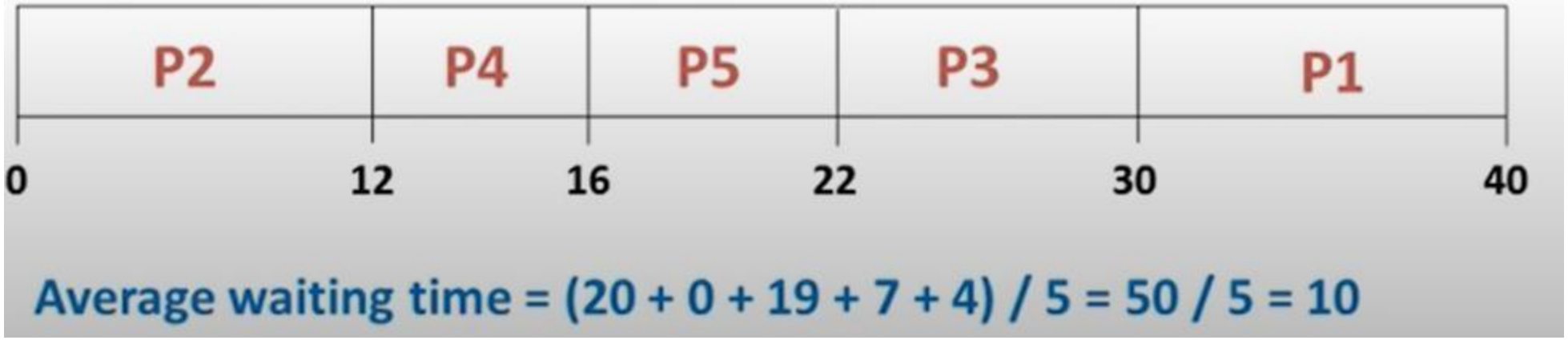
P1 = 30 – 10 = 20

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

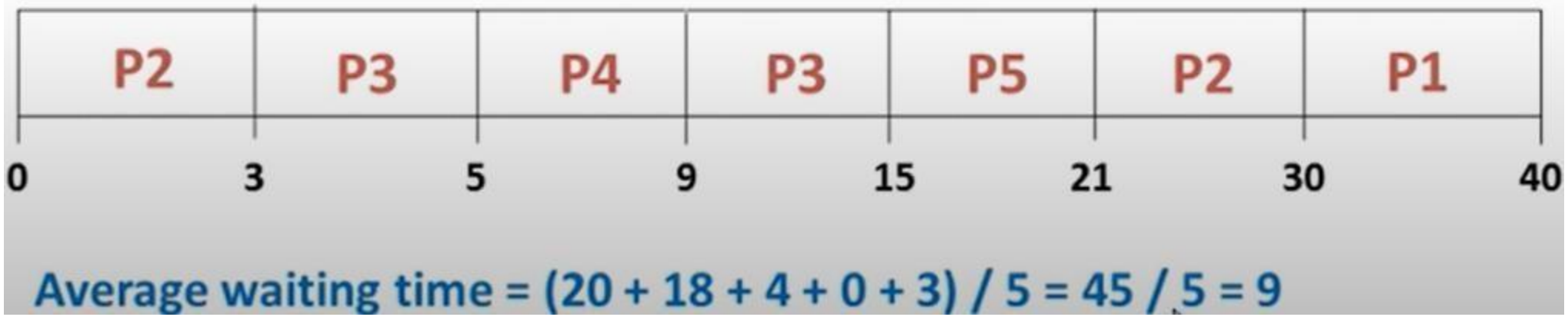
P1 = 30 – 10 = 20

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

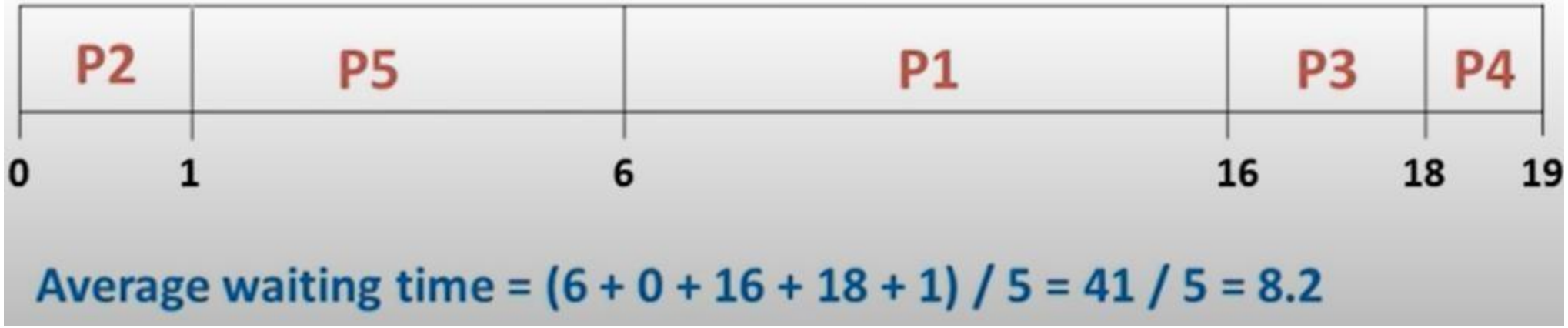
Shortest Job First Preemptive Scheduling is also known as Shortest remaining Time(SRT)

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



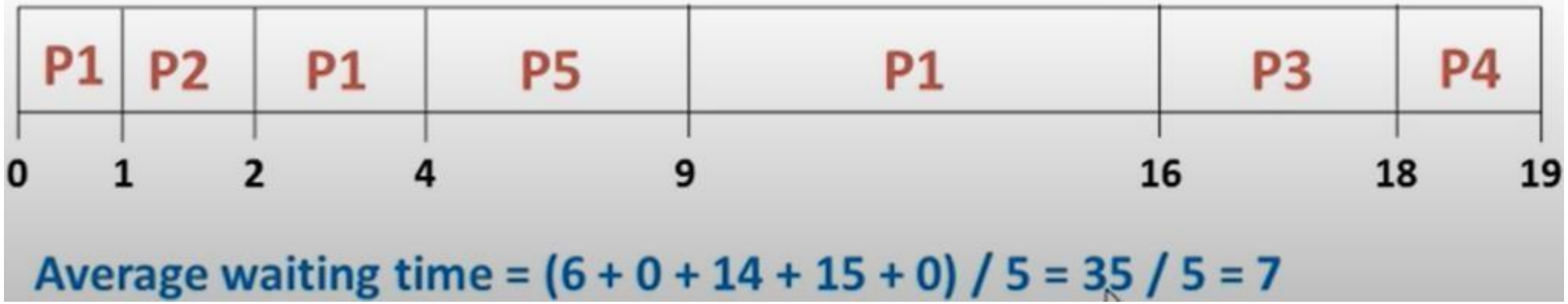
Average waiting time = (6 + 0 + 16 + 18 + 1) / 5 = 41 / 5 = 8.2

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

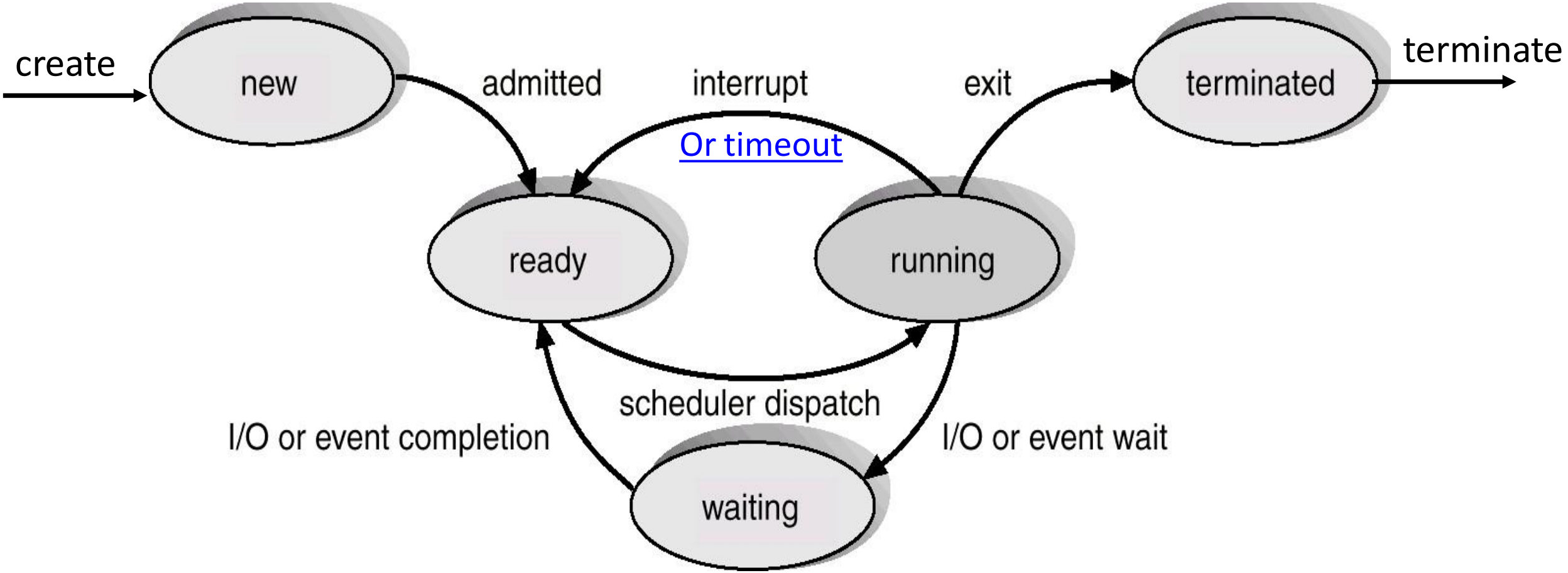
Waiting time :
start time – arrival time

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



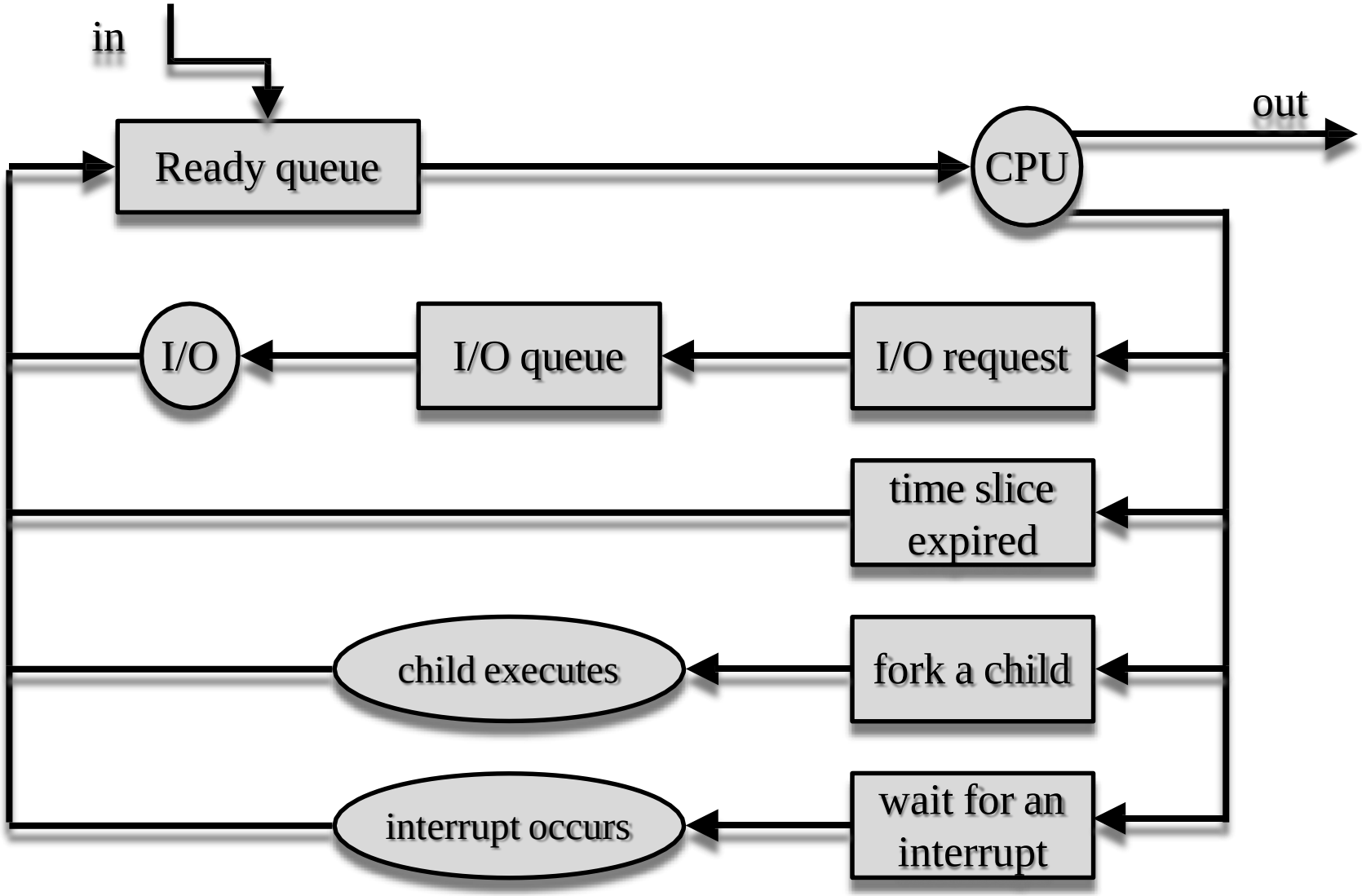
Process States

- 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.

Scheduling Queues



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.

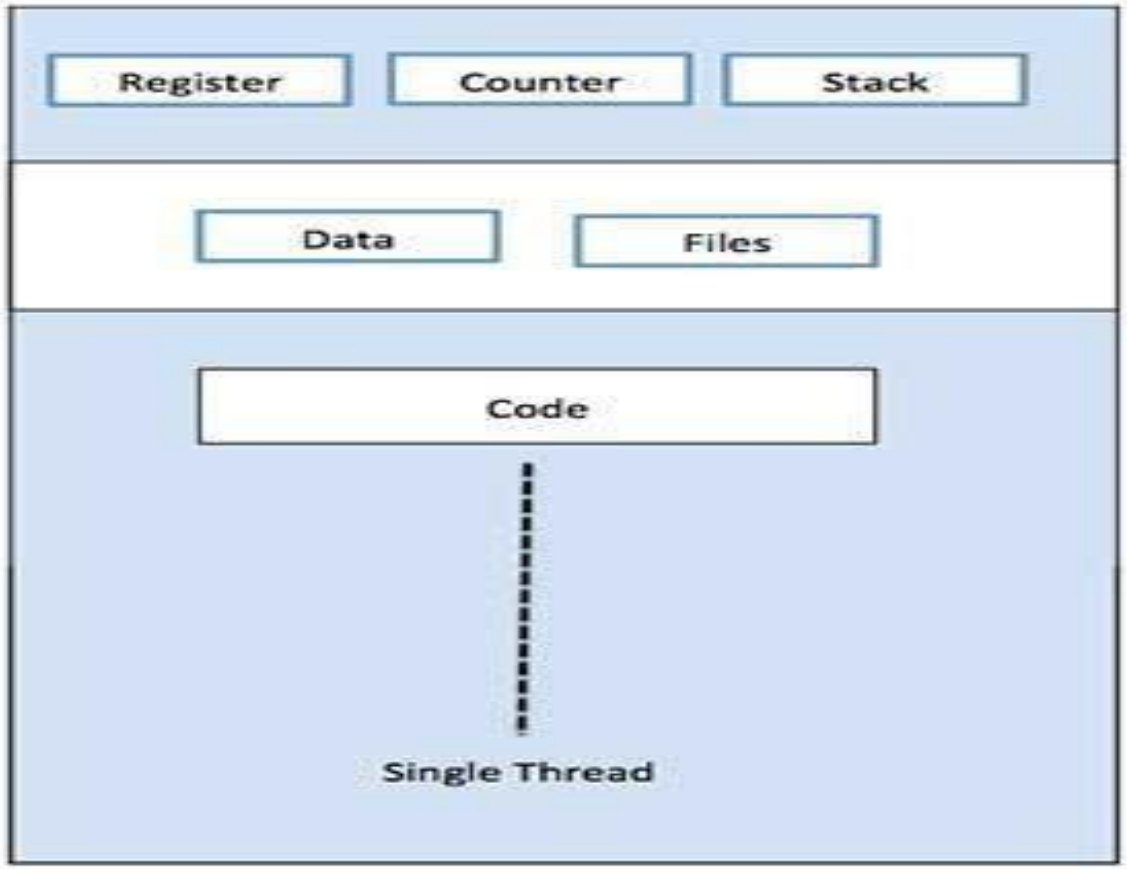
What is Thread?

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

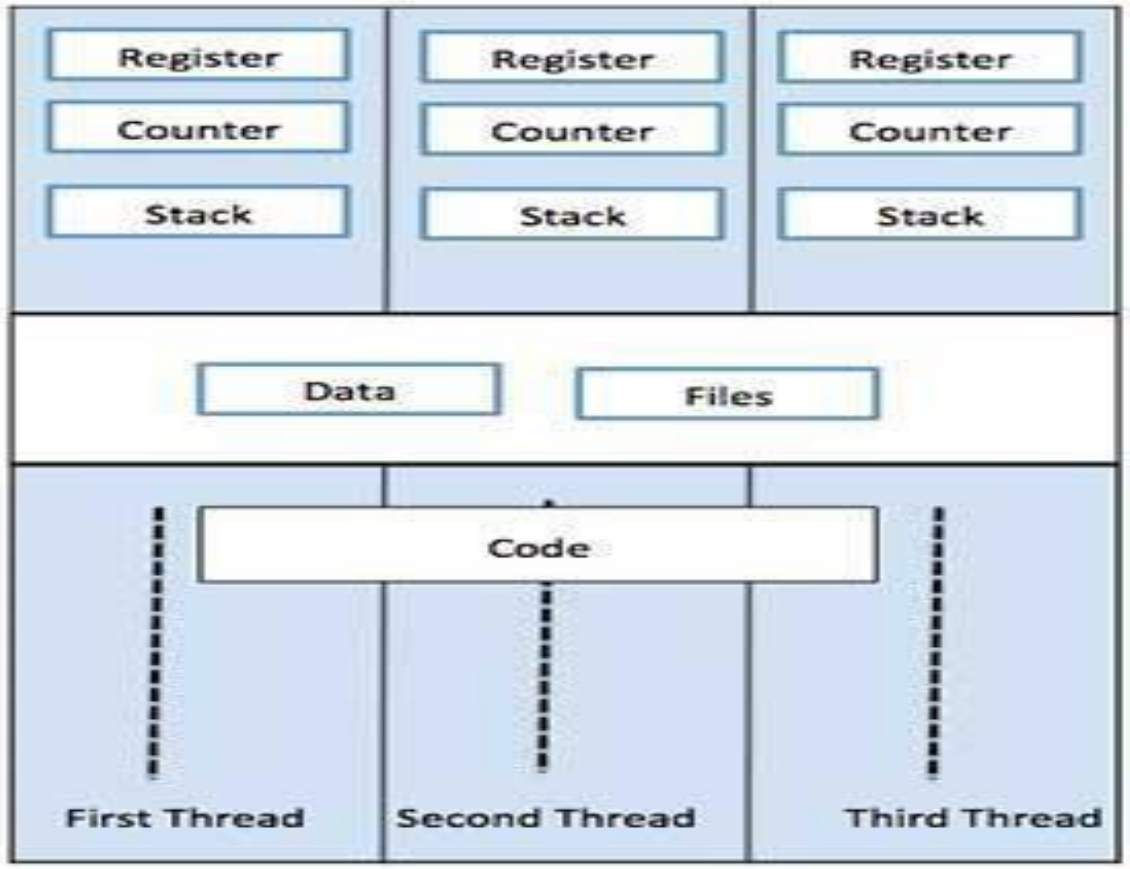
- When a process gets executed, **it could create one or more threads** internally that can be executed on the processor.

- **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.
- **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

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

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

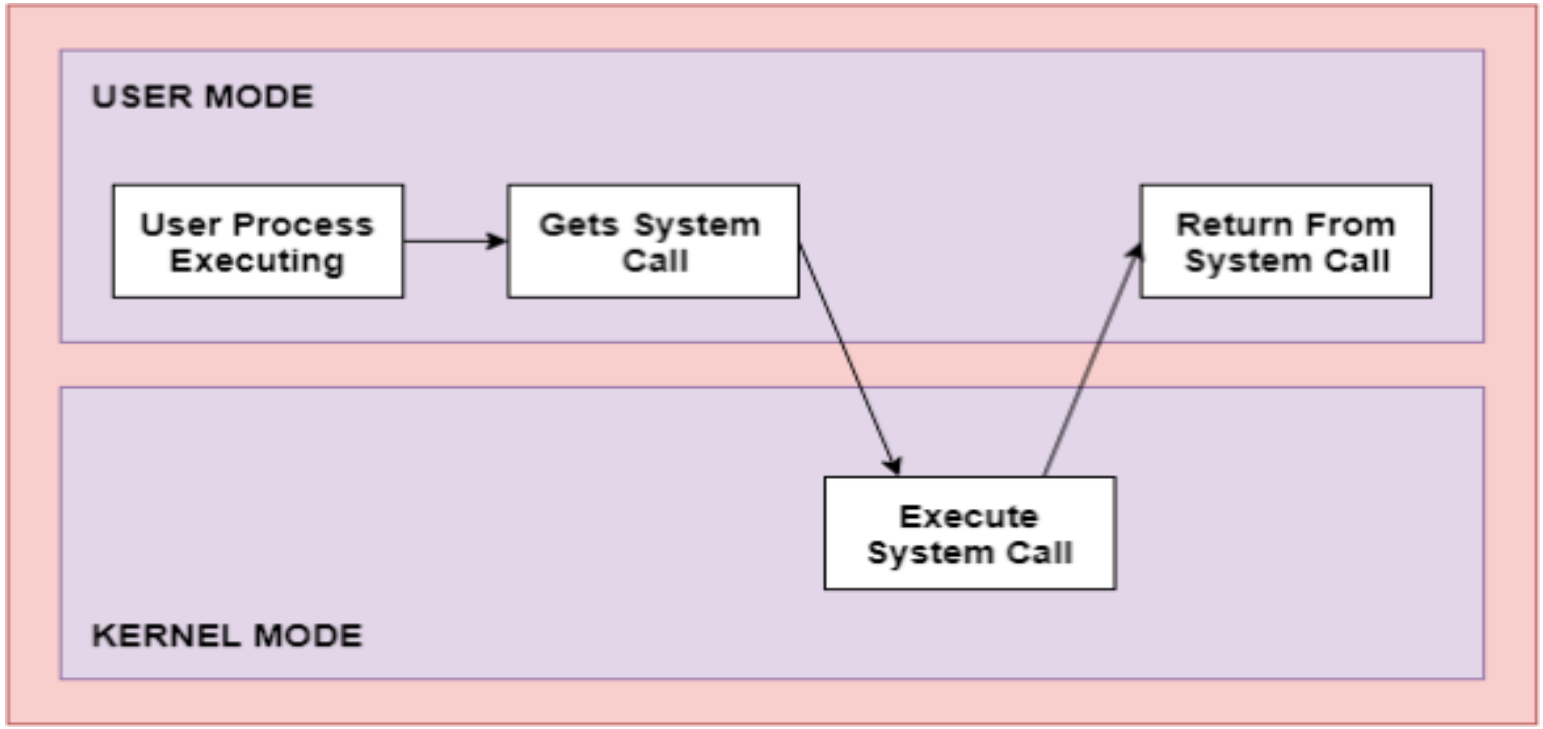
The **interface between a process and an operating system** is provided by **system calls**.

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



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.

Thank You