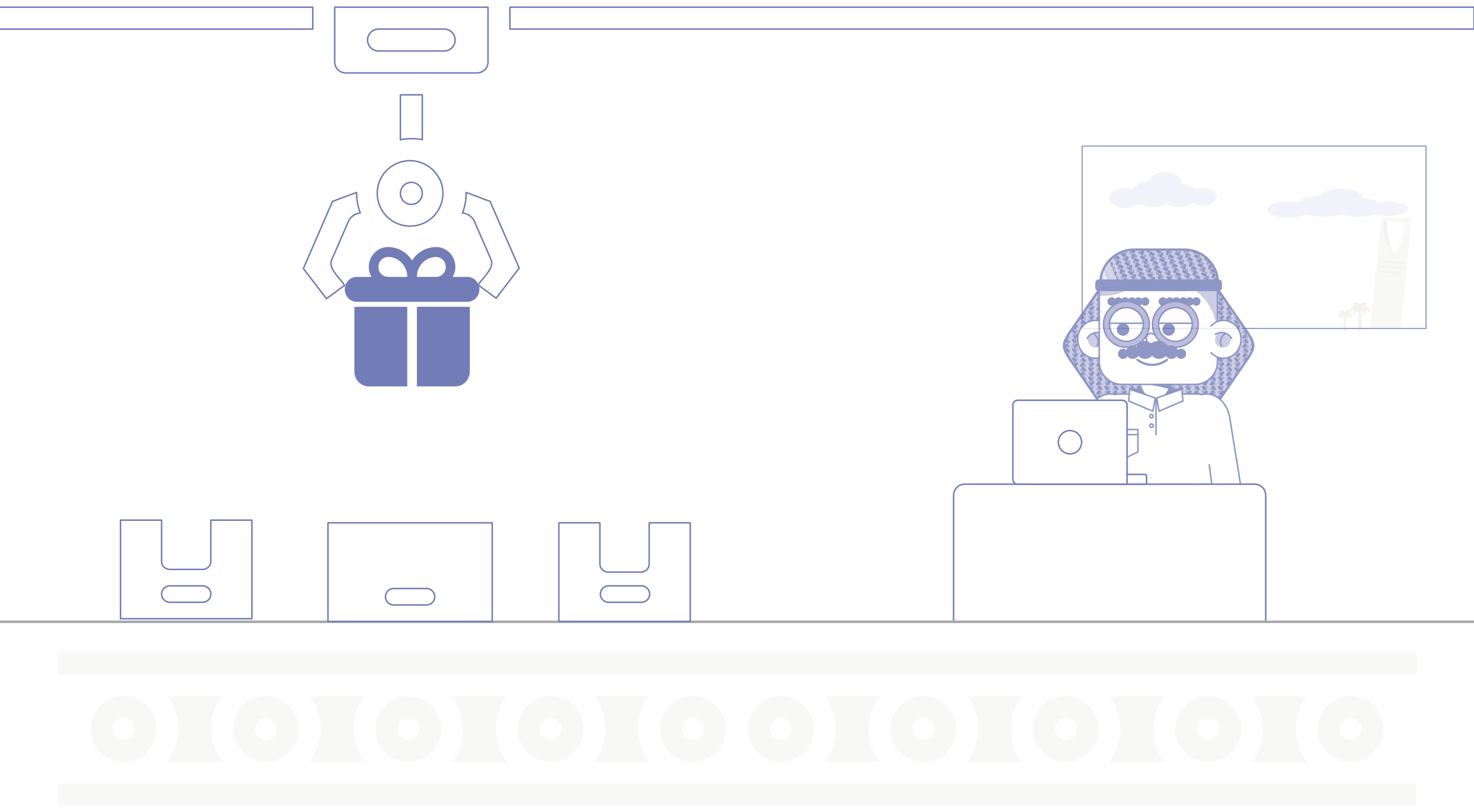
OPERATING SYSTEM

Introduction



Introduction to Operating System

Step 01 Introduction

Besides anything you might think of as an Operating System, let us keep it and see an example before everything else.

In a company, there is an employee called **A** who would like to use the company's resources to complete his task. He needs a calculator, a document writer, and a printer.

- **Calculator:** Calculate a mathematical operation.
- **Writer:** Write the result in a document.
- **Printer:** Print a document.

All these resources are available in the **office room** and he can have full control of it at any time since there is no one else needs it.

However, the demand for the company's services increased, and more team members joined. After C and B joined, there was a conflict in using these resources.

A, **B**, and **C**, each have their own tasks to complete, but to do so they need the resources in the office room.

A, has a problem to solve, write the solution, and then print it.

```
A needs: calculator -> writer -> printer.
```

B, has a ready-made document that needs to be printed.

```
B needs: printer.
```

C, has a problem to solve and to write it as a document.

```
C needs: calculator -> writer.
```

What do you think will happen in this case?
The team had a conflict about who can use the resource first. Each member takes his own task as a priority and wants to complete it.

What do you think should happen?

In this case, the team leader/office manager should manage the use of resources by assigning each member to a resource.

- Calculator: A then C
- Writer: None
- Printer: B

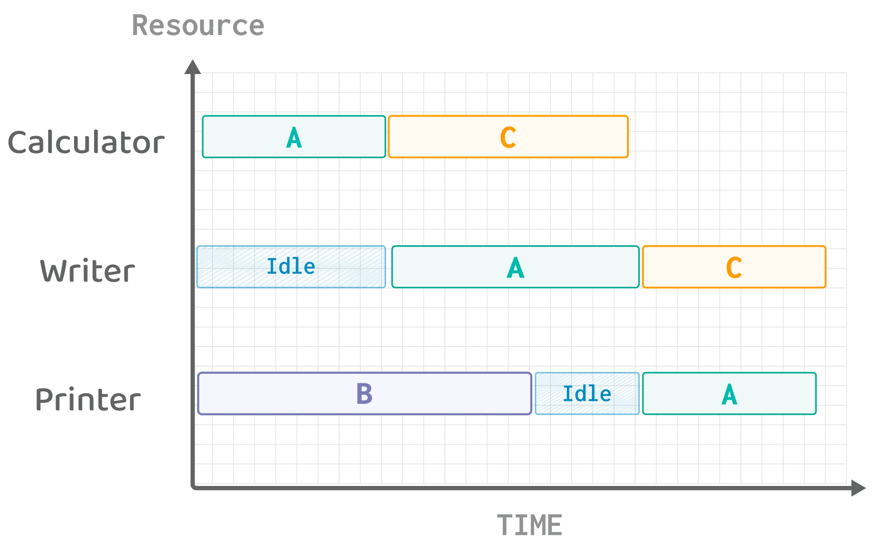
01 A can use the calculator first and C should wait until it becomes available.

02 Once A has finished from the calculator he can directly use the writer since it is available.

03 C can use the calculator since A has finished.

04 B can directly use the printer since no one needs it.

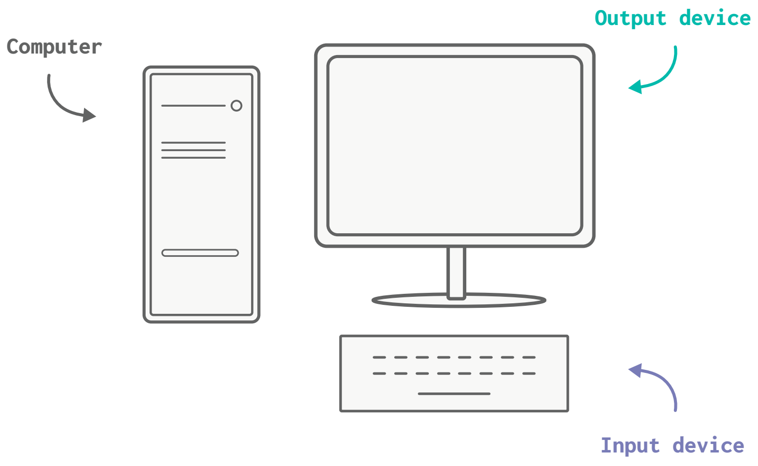
05 A must wait for B to finish from the printer and use it to complete his job.



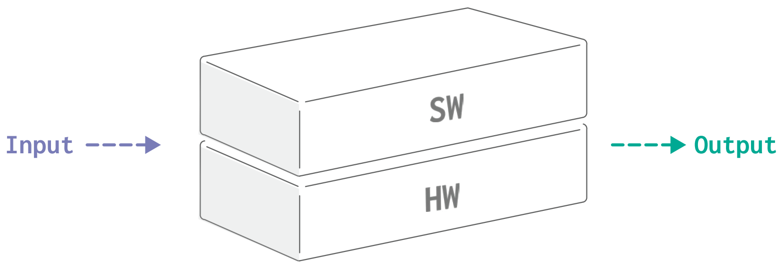
The team leader can decide if C or A should use the calculator based on the strategy he follows. If based on priority, C might use the calculator first if he has a higher priority task. If based on who came first, A should use the calculator first.

In the end, the company's services are provided. But how it was managed internally depends on its structure & organization.

The company takes an input which is the request, handles it, and then produces an output. Same as the computer.



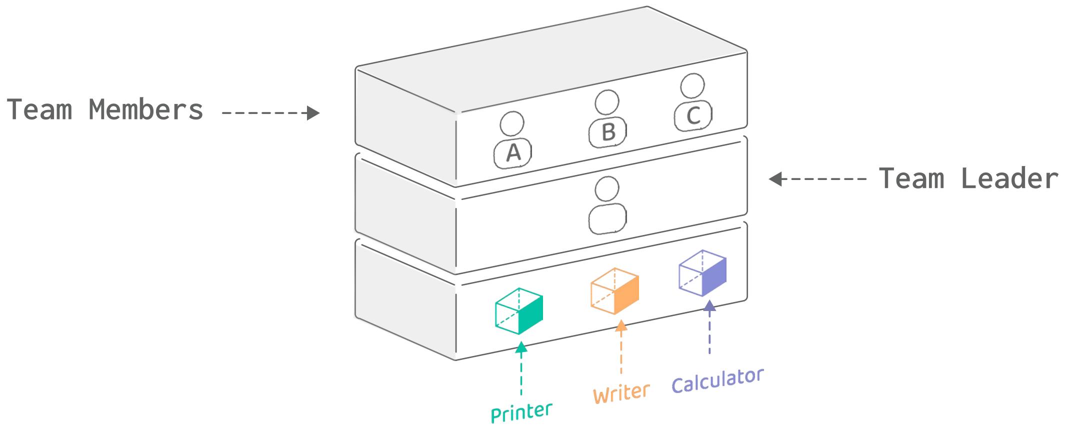
The computer internally is composed of two parts, the software and the hardware.

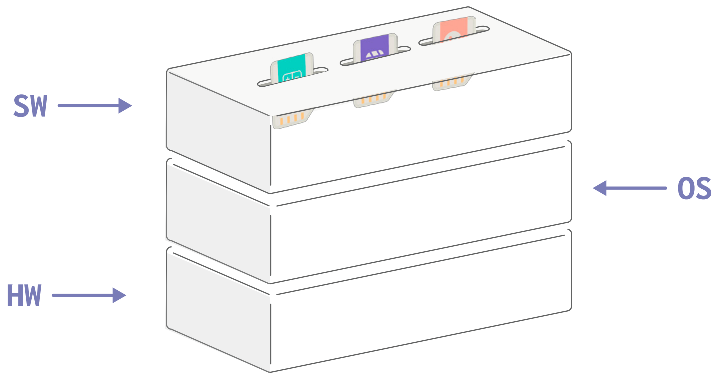


In early computers, there was only one software running on the hardware. The software had full control over the hardware, same as the first case with A being the only employee who needed the resources.

However, with the evolution of computer systems, computer manufacturers wanted to make use and utilize the hardware more efficiently by running several programs at the same time. To do so, a specialized software was developed which is the operating system.

The operating system acts as a mediator just like the team leader from the example between software programs and hardware resources to manage its usage.



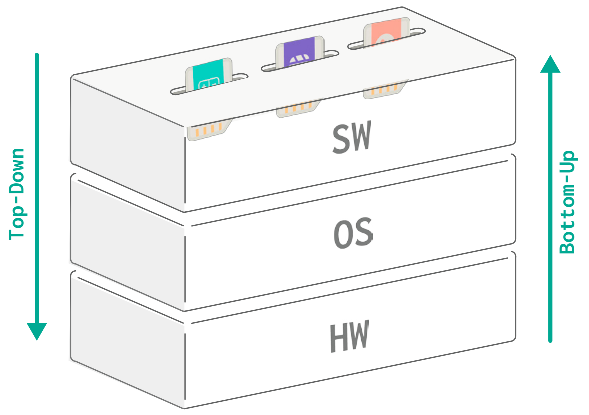


By being a mediator, the operating system provides the programs with a list of services it can perform for them and manage the usage of the hardware resources.

Step 02 Operating System Purpose

The purpose of operating systems can be viewed from two directions:

- **Top-Down:** Provide simple APIs for software programs to use.
- **Bottom-up:** Manages resources and schedules their usage.



Step 03 Termonologies

01 Software

Software (aka. program) is a set of instructions that drives the hardware to perform. Sometimes software programs are categorized into **System software** and **Application software**. Both are fundamentally the same, but they are used for different purposes.

- Application Software

It is a common type of software program. Application software is written for normal users to perform their tasks. For example, games and text editors.

- System Software

System software is written for the system to enable it to perform its job and help application software to do its intended tasks. For example, compilers and operating systems.

02 Hardware

Hardware refers to the physical resources in a computer. It contains the resources needed by the computer to perform. The essential hardware resources are the following,

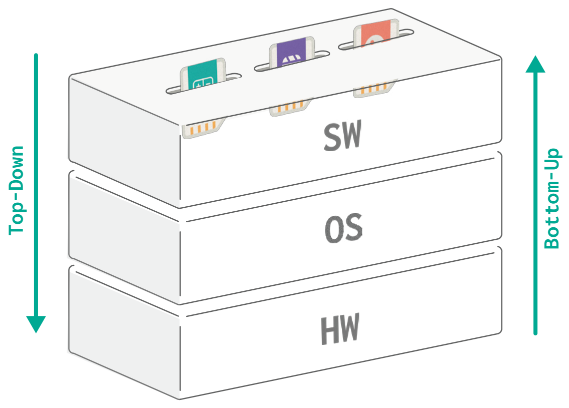
- CPU: aka. processor is responsible for executing instructions.
- Main Memory: used for storing data temporarily.
- I/O devices: used to get input and produce output.

03 Operating System

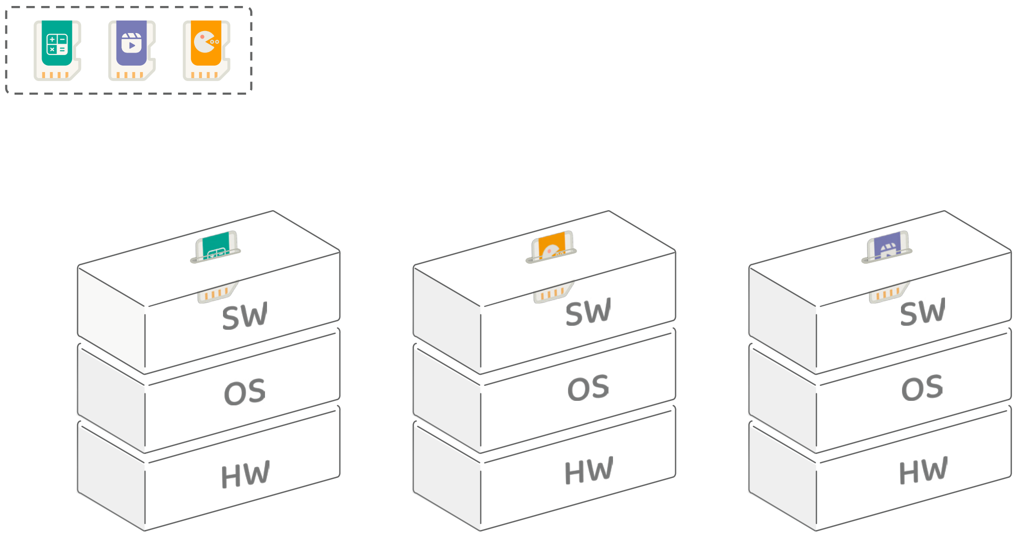
Now we know the key concepts of computer systems, we can dive deep into the operating system and its purpose.
Operating System is widely known as a system software that manages both hardware and software resources of a computer.

Step 04 Operating System Perspective

Operating systems as described earlier, act as a mediator between hardware resources and software programs for two reasons. First, to provide programs with an interface of services they can request. Second, to manage hardware resources for better utilization.



Since early computers couldn't run two programs at the same time because programs needed to dominate the hardware resources to execute, Operating systems managed to run several programs at the same time by misleading the programs into believing that each of them had the computer resources for itself. But the truth is that OS manages programs by suspending some and executing others so quickly that they seem to run at the same time. However, that is just an illusion given by the OS.



The image above represents the illusion given by operating systems, each program thinks the computer is for itself.

OS wouldn't manage programs and evolve as we see them today without the concept of **process**.

Step 05 Process

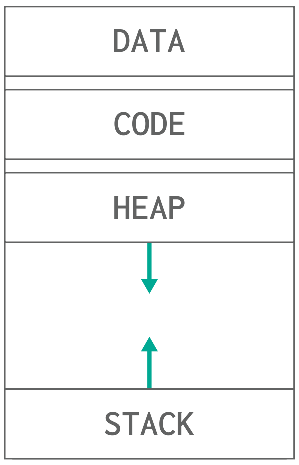
What is a process? The **Process** is a program in execution. When you run a program by clicking the program icon, operating systems map that program into a process like filling a form of data. Answering how much storage it needs? Assign it an id and status etc... By doing this, the operating system is able to manage the program execution and assign it the resources it needs.

When clicking a program icon to run it, the operating system loads the program from the disk (which stores all programs permanently) to the main memory with a specific structure.
The program is then called a process. A **process** is a program in execution.

What happens in the background when you run a program?

At first, a program is just an execution File sitting on the disk storage. When you click the program icon, the operating system loads the program into the main memory as a process, and then the CPU starts reading the program instructions and executing them.

The process has a uniform structure that it applies to when loaded to the main memory as shown in the image below.



Operating systems handle programs as processes to unify how they execute and manage programs. If each program has its own structure, the operating system won't be able to handle all programs and will only support a limited range of them. This conflicts with the purpose of the operating systems.

You can think about it like in restaurants, when you give the waiter your order, you need to use a language that he understands so he can serve you. After taking your order, he will pass it to the kitchen and let them perform the order, and deliver it when finished. We conclude that in order to perform your request, there is a set of rules of rules you need to follow. The same goes with the operating system which acts as the waiter. Programs should talk in a language the operating system understands to perform their requests, which is the language of **processes**.

Step 06 Conclusion

We have introduced the concept of operating systems and their purposes. We will learn on the next topic how to interact with the operating system.

#

Interacting With Operating Systems

Step 01 Interacting with Operating Systems

As normal users, we can interact with the OS by user interfaces such as a Graphical User Interface GUI when clicking programs and closing them or using the Command Line Interface CLI. However, software programs communicate with the OS using System Calls aka syscalls. The operating system provides software programs with a list of services they can request to access a resource or do an action which is syscalls.

Syscalls are used to manage processes, memory, files, and much more. As programmers, we develop software programs. Therefore, we need to understand how to interact with the operating system through the use of syscalls in the development phase of a program.

In the next topics, we will focus on syscalls that are related to memory management, process management, and I/O syscalls. But first, we will walk through the big picture of how operating systems handle processes.

> **NOTE**
> Syscalls allow software programs to access and utilize hardware resources without needing to know the underlying details.

Step 02 Program Life-Cycle

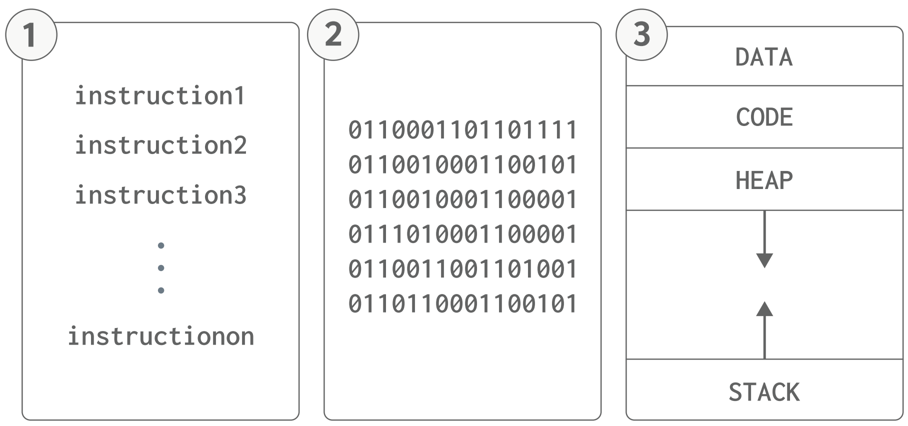
We are aiming to understand operating systems from the programmer's perspective. Hence, we are required to understand the phases our programs go through and how operating systems interact with and manage them.

The program in its lifetime, is going to walk through several phases.

01 Source Code

02 Executable File

03 Running Program (Process)



At the start of creating a program, it will be just a source code which is a set of instructions written in a specific programming language. When the program is compiled, the executable file will be generated. The executable file is stored on the disk permanently until the user clicks on it to run. When a program is clicked, it will be mapped into the main memory and will start running, at this moment, we call it a Process.

> NOTE
> The process is a program in execution.

Step 03 Example

Hello World program in C.

```
#include <stdio.h>

int main(){
    puts(“Hello World”);
    return 0;
}
```

The program at first is a set of instructions to tell the computer to print **`Hello World`** when it runs. However, the computer can not understand high-level programming languages directly, we need to compile the program file to get an executable file and run it for execution.

compile C program

```
gcc hello.c -o out
```

Once we run the command above, the compiled **`out`** file will appear.

run the executable file

```
./out
```

Running the executable file will produce the following output.

OUTPUT

```
Hello World
```

Process Anatomy

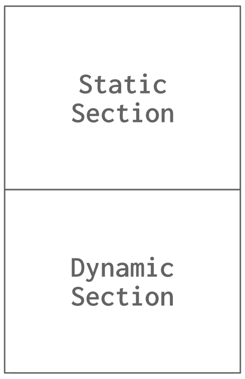
Step 01 Process Anatomy

A process is essentially a program that is currently being executed. When a program runs, it is transformed into a process, which is assigned an identifier (ID), a state,
memory space, and various other attributes. In this lesson, we will take a closer look at processes and the related data associated with them.

01 Process

You can think of a process as a task assigned to the computer that it needs to execute. For the computer to perform this task effectively, it must understand the information and resources required by the program. This necessity is why processes are organized in a standardized format that allows the operating system to know how to interact with them.

When a process is loaded into memory, it is divided into two main sections: the static section and the dynamic section.



02 Static Section

The static section contains data that is known at compile time, such as:

Global Variables: These variables are accessible from any part of the program and retain their values throughout the program's execution.

Program Instructions: This is the executable code/instructions written by the programmer that the CPU processes or executes.

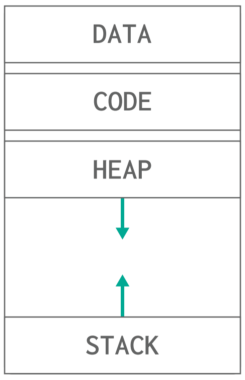
The size and layout of this static section are determined during the compilation of the program, meaning that the necessary space for these elements is allocated before the program runs.

03 Dynamic Section

In contrast, the dynamic section is responsible for data that is defined at runtime. This includes:

Heap: This area is used for dynamically allocated memory, such as when new objects are created. The heap allows for flexible memory usage as resources can be allocated and deallocated as needed while the program is running.

Stack: The stack is used for managing function calls and local variables. Each time a function is invoked, a new stack frame is created to hold its local variables and return address; this frame is removed when the function completes.



Step 02 Process Metadata

Processes are managed by the operating system, which handles their creation, scheduling, and termination. To do so, the operating system stores metadata about the process for management purposes.

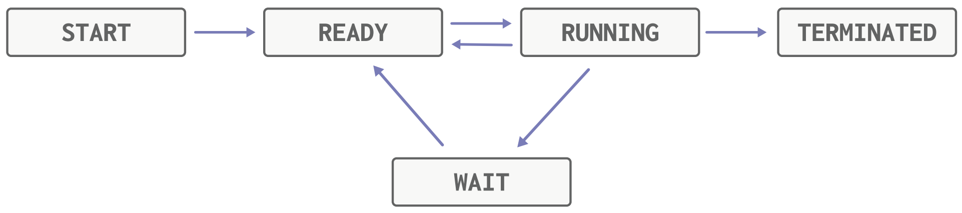
DATA	DESCRIPTION
PID	Process ID uniquely identifies a process
State	The current state of the process
PC	A pointer to the next instruction to be executed in the program/process
User	The user who started the process

There is more information some of them are specific to the operating system needs for example the process priority to understand when will it be executed.

Step 03 Process State Life-Cycle

Each process has the information of its state, the states the process can be in are as follows.

- **New/Started:** The process has just been created.
- **Ready:** The process is ready to be executed.
- **Running:** The process is currently being executed.
- **Blocked/Waiting:** The process is waiting for an external event to happen such as I/O device availability or operation.
- **Terminated:** The process has completed its execution.



Step 04 Summary

We have introduced the overall picture of how operating systems manage processes and their life cycle. But keep in mind that it was only introduced at a high level to better understand future topics and discussions.

Later on, we will deep dive into how processes are mapped and managed, and how can we request services from the operating system using syscalls.