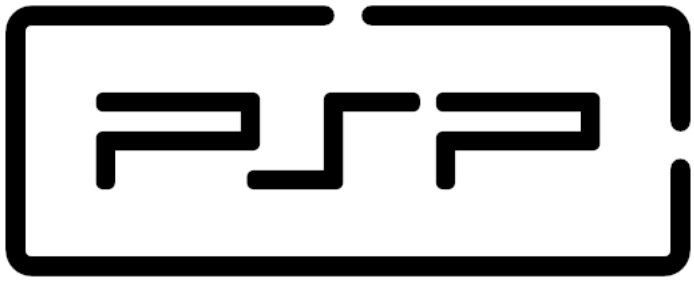


Process and Service Programming

1.1. Processes, programs, threads

1.1. Processes, programs, threads

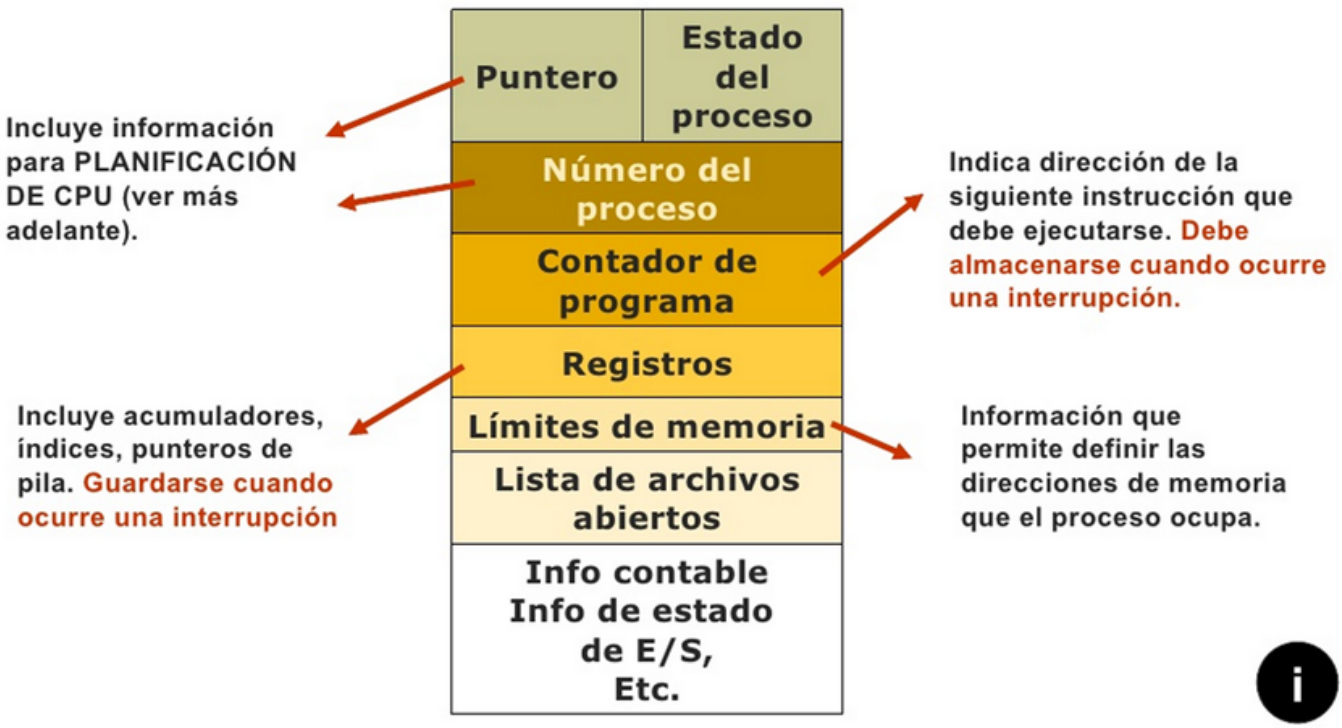
1.1.1. Processes and programs

A program and a process are related terms. A `program` is a group of instructions to carry out a specified task with some input data.

> **Black box**
>
> Black-box testing is a method of software testing that examines the functionality of an application without peering into its internal structures or workings, just by setting some input data set and checking if the output data set meets the expected one.

While a `process` can be described as an instance of a program running on a computer. A program becomes a process when loaded into memory and thus is an active entity While a program is considered to be a passive one.

A process has a high resource requirement, it needs resources like CPU, memory address, I/O during its lifetime. It has its own control block called Process Control Block where relevant information as program counter, registers, stack, *executable code*, state, ... and all it needs to be run by the OS is stored.



Each process is an independent entity. There exist a many-to-one relationship between process and program, which means one program can be invoked multiple times getting several processes in memory running the same copy of the program.

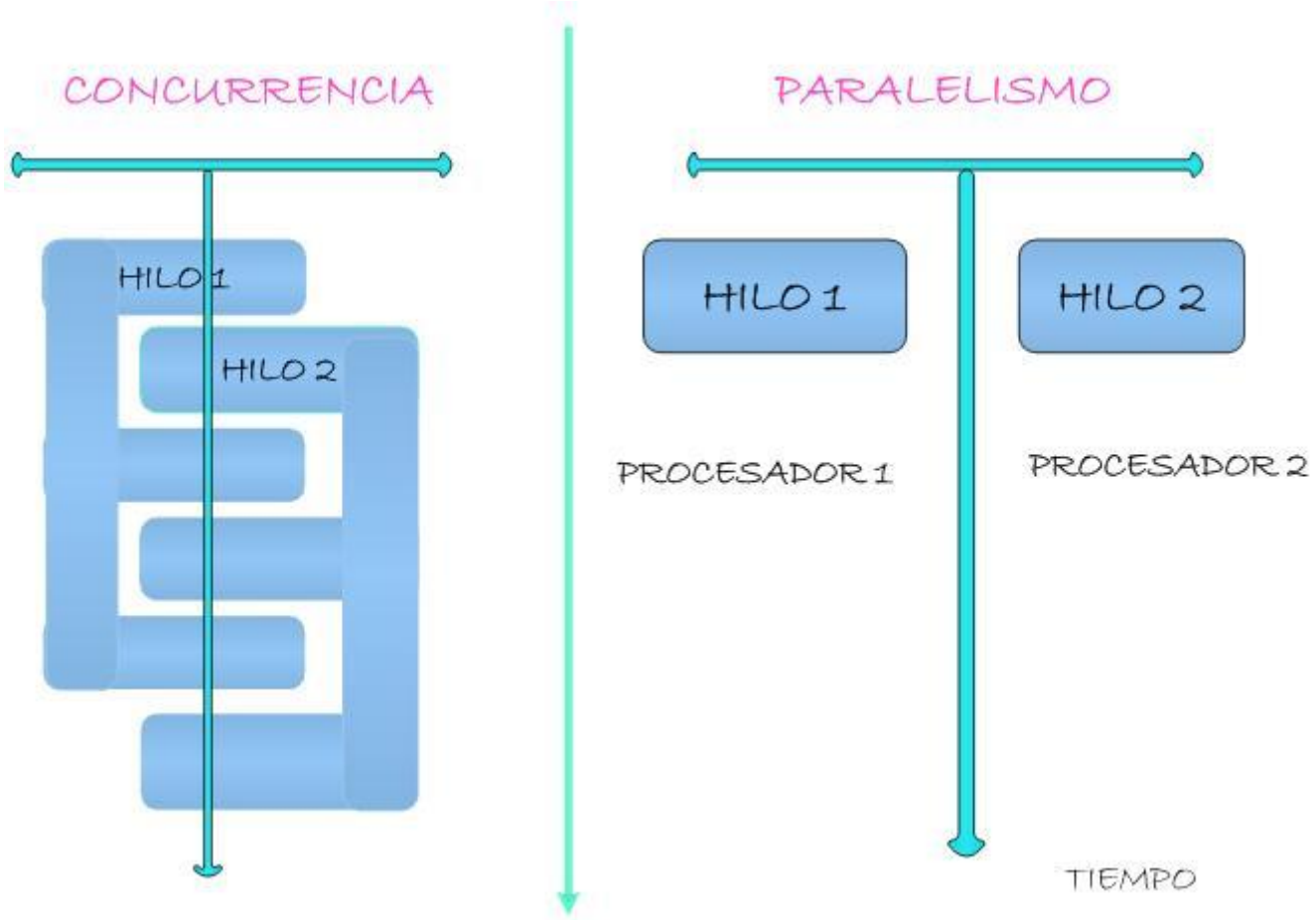
1.1.2. Concurrent programming

In computer science, concurrency is the ability of different parts or units of a program to be executed out-of-order or at the same time simultaneously.

This allows for `parallel` execution of the concurrent units, which can significantly improve overall speed of the execution in `multi-processor and multi-core` machines.

The concept of concurrent computing is frequently confused with the related but distinct concept of parallel computing, although both can be described as *multiple processes executing during the same period of time*.

- In parallel computing, execution occurs at the same physical instant: for example, on separate processors of a multi-processor machine, with the goal of speeding up computations.
- This parallel computing is impossible on a `one-core single processor`, as only one computation can occur at any instant (during any single clock cycle). By contrast, concurrent computing consists of process lifetimes overlapping, but execution need not happen at the same instant. The goal here is to model processes in the outside world that happen concurrently using `multitask`.



Concurrency

By and large, both previously described scenarios are gonna be referred to as **concurrency**.

What for?

The Real World is Massively Complex

- In the natural world, many complex, interrelated events are happening at the same time, yet within a temporal sequence.
- Compared to serial computing, parallel computing is much better suited for modeling, simulating and understanding complex, real world phenomena.

Imagine modeling these serially: climate change, rush hour traffic, weather forecast, galaxy formation, ...

Main Reasons for Using Parallel Programming

- Save time and money. In theory, throwing more resources at a task will shorten its time to completion, with potential cost savings.
  - Parallel computers can be built from cheap, commodity components.
- Solve larger and more complex problems. Many problems are so large and/or complex that it is impractical or impossible to solve them using a serial program, especially given limited computer memory.
  - *Grand Challenge Problems* (en.wikipedia.org/wiki/Grand_Challenge) requiring petaflops and petabytes of computing resources.
  - Web search engines/databases processing millions of transactions every second
- Take advantage of non-local resources. Using compute resources on a wide area network, or even the Internet when local compute resources are scarce or insufficient.
  - *SETI@home* (setiathome.berkeley.edu) has over 1.7 million users in nearly every country in the world. (May, 2018).
- Make better use of underlying parallel hardware. Modern computers, even laptops, are parallel in architecture with multiple processors/cores.
  - Parallel software is specifically intended for parallel hardware with multiple cores, threads, etc.
  - In most cases, serial programs run on modern computers "waste" potential computing power.
- Increase security. Each task can be isolated in a different process, so debug and check the security, even finishing it when it's not working properly, can be done without hanging the whole system.

Historically, parallel computing has been considered to be "the high end of computing", and has been used to model difficult problems in many areas of science and engineering. Today, commercial applications provide an equal or greater driving force in the development of faster computers. These applications require the processing of large amounts of data in sophisticated ways. For example:

New hardware environments can be classified in terms of:

- Microprocessor with many cores sharing system memory.
- Multiprocessor systems with shared memory
- Distributed systems and cloud services.

Process communication and synchronization

The concurrent running of many processes may suppose the collaboration of some of them in order to complete a common task, while they can also be competing for system resources.

In both cases it is compulsory to add communication and synchronization techniques for the processes.

Concurrent programming

> Concurrent programming and PSP is just about that, the knowledge of these **communication and synchronization techniques**.

When thinking about the way a process can communicate with each other, there are two main options:

- Message passing: It's commonly used when processes are running on different devices. They exchange information following a protocol previously set and agreed by the parts.
- Shared resources / memory: It's only available when both processes are running on the same device and allows process synchronization based on a shared resource value or state.

We can also classify the communication by the synchronization the processes use during the message passing process:

- Synchronous communication happens when messages can only be exchanged in real time. It requires that the transmitter and receiver are present in the same time and/or space. Sender is blocked until receiver gets the message. Both processes are synchronized at the reception time.
  - Examples of synchronous communication are phone calls or video meetings.
- Asynchronous communication happens when information can be exchanged independent of time. It doesn't require the recipient's immediate attention, allowing them to respond to the message at their convenience. Sender continues with it processing just after delivering the message to the receiver, not being blocked.
  - Examples of asynchronous communication are emails, online forums, and collaborative documents.

1.1.3. Services and threads

A program, as previously said, is a group of sentences (actions and checks) and a running workflow. The workflow line determines the execution order for the sentences, with dependency of the program structure and it's data.

Based on the number of workflow lines a program can have, processes are classified in terms of::

- Sequential: The have only one control workflow (monothread)
- Concurrent: They have multiple control workflows (multithread).

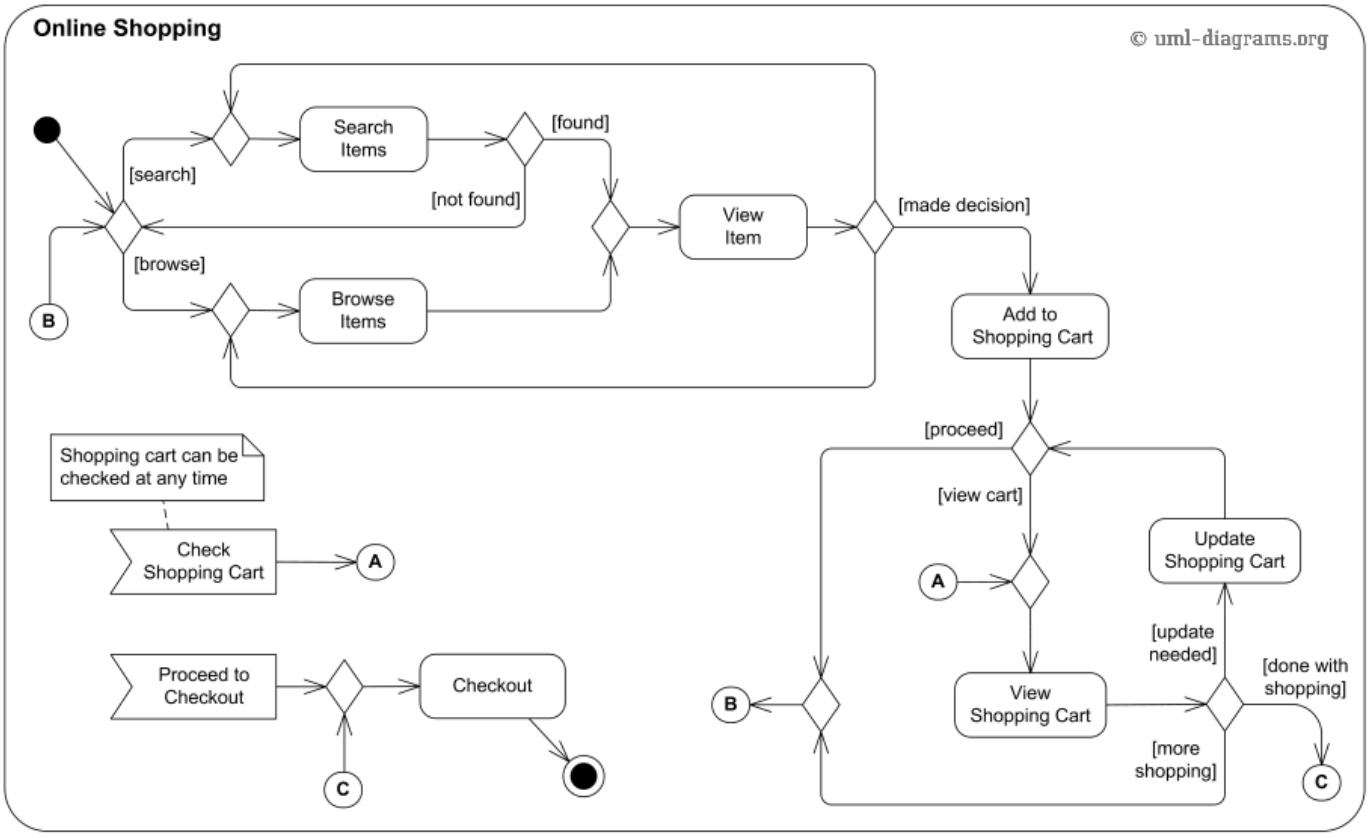
Sequential program (Von Newmann architecture)

The classical von Neumann model of computation4 is a familiar model of sequential behavior. According to this, when we start learning to code we learn the classical way, following Von Neummann's conceptual model.

Sequential programs have a single workflow line. instructions on these applications are strictly sorted as a lineal time sequence.

The program's behavior is a function of the kind of instructions it is made of and the order they are run (set by its input data)

In sequential programs the time every sentence takes to complete has no consequences on the final result.



The way to test a sequential program (`verify` or `debug`) is so easy:

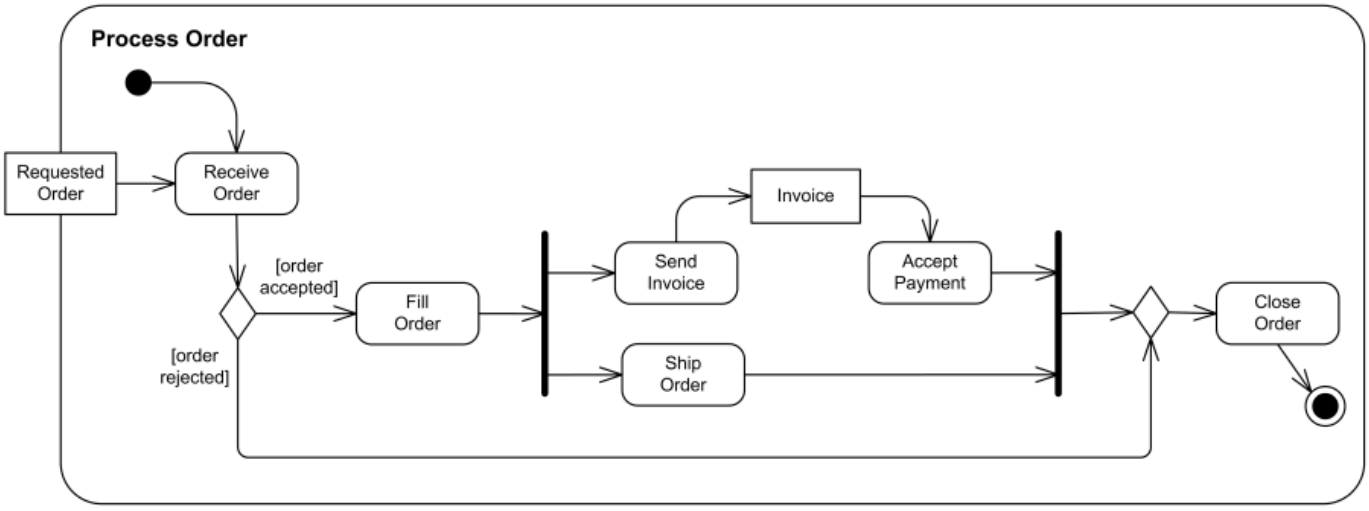
- Every sentence gives the right output.
- The sentences are executed in the expected order.

That's the basis of many basic test methods, as the "white-box" model.

Concurrent program

In concurrent programs there are many workflow lines. The sentences are not run following the same order as in a sequential program the would do.

In concurrent programs sequential order between sentences is still relevant. Nevertheless, in concurrent programs the order is only partial while in sequential programs the order is strict.



In concurrent programs the *sequencing* for concurrent processes is called **synchronization**.

The partial order implies that concurrent programs does not have to be deterministic, that is, the application results with the same input data will not always be equal.

Indeterminism

Having different outputs for the same inputs does not means that a concurrent program has any bug or malfunction.

Look at the following pseudo code example

```java
public class TestClass {
    int x;

    public void testMethod1() {
        for (int i=1; i <= 5; i++) {
            x++;
        }
    }
    public void testMethod2() {
        for (int j=1; j <= 5; j++) {
            x++;
        }
    }
    public void sequential() {
        x = 0;
        testMethod1();
        testMethod2();
        System.out.println(x);
    }
    public void parallel() {
        x = 0;
        // cobegin-coend means that both methods are run simultaneously
        // These sentences doesn't exist in Java. They are used for
        // sample purposes
        cobegin
            testMethod1();
            testMethod2();
        coend
        System.out.println(x);
    }
}
```

¿Which is the value for variable x after sequential method is run?

¿Which is the value for variable x after parallel method is run?

Historical review

The nature and models of interaction between processes of a concurrent program were studied and described by **Dijkstra** (1968), Brinch **Hansen** (1973) and **Hoare** (1974).

The academic study of concurrent algorithms started in the 1960s and they are the foundation for multiprocess operating systems in the 70's and 80's.

Concurrent programs inherent indeterminism makes its analysis and validation more complex. However, to test a concurrent program (`verify` or `debug`) the same techniques as for sequential ones are needed, adding these new ones:

- Sentences can be validated individually only if the are not engaged to shared variables.
- If shared variables are used there can be many interference effects for concurrent sentences and testing can also become very difficult. warning
- Only when sequencing between tasks is made by using explicit **synchronization** sentences, time is not relevant on the result.

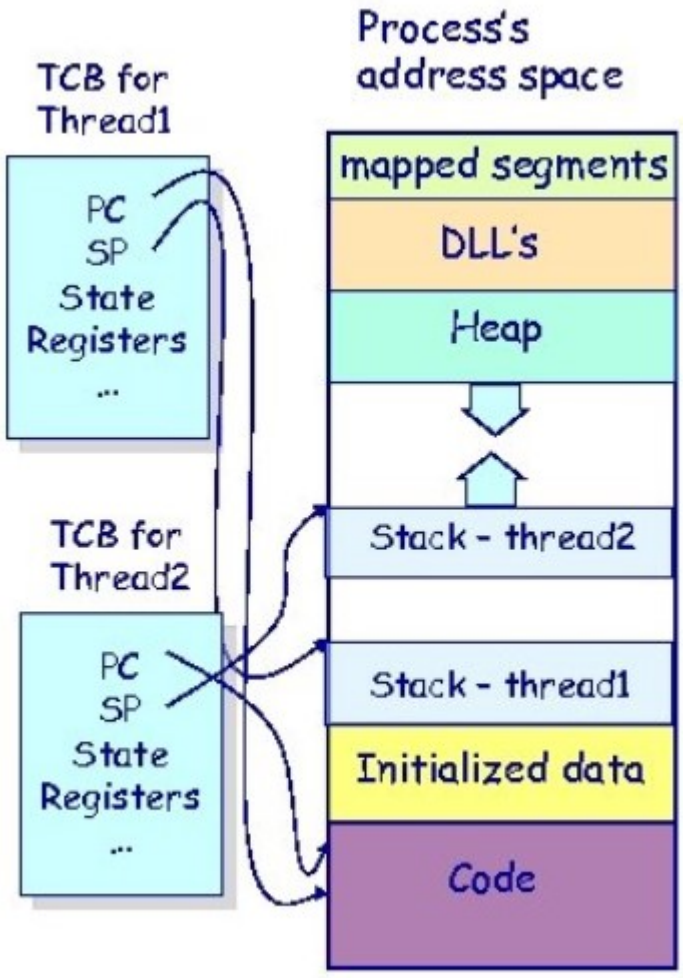
> **Important**
>
> Three previous topics described above are the basis of concurrent programming.
>
> 👁 To know them, to understand them and to apply them in the right way is all about what we are gonna learn all this course.

Threads vs processes

A thread is the unit of execution within a process. A thread is just one of the workflow lines a concurrent process can have. A process is a heavyweight running unit.

A process can have anywhere from just one thread (the main thread) to many threads. If a process has more than one thread, every thread is a lightweight running unit.

Processes	**Threads**
Have more than one thread	A thread always exists within a process
They are independent from others	They share the process resources
The OS manages them	The process manages them
they can communicate on the OS	The process manages their communication



In the above image you can see the relationship in the way a thread is created and its related process.

- The process resides in its memory address space. Threads share that memory area. In the process's address space every thread has its reserved area, but all of them can share the process's global memory and his open resources (files, sockets, etc.)..
- We have already described a process PCD with the process information..
- In a similar way, have their TCB (Thread Control Block) where the threads store their specific information (program counter, stack pointer, thread status, registers and a PCB pointer).

Services

A service is a process commonly started during OS boot. As it does not need user interaction services are run as **daemons** run in *background mode*.

They are called services because once started they are waiting for a process to ask them to do a task. As they have to manage request from several processes they usually are multithread programs.