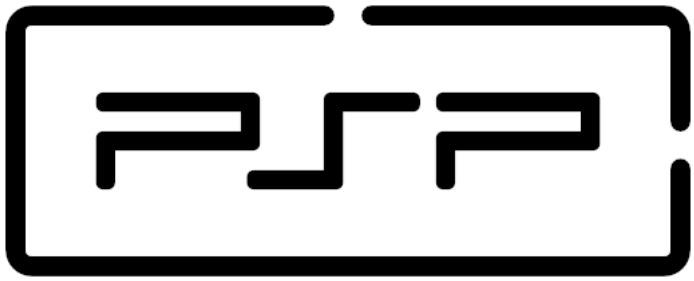


Process and Service Programming

1.2. Concurrency

1.2. Concurrency

According to the Collins dictionary some of the senses os the word concurrency are

> Ccooperation or combination. Simultaneous occurrence; coincidence.

If we change occurrence to `process`, we get a definition closer to its sense in the computer science.

This is not the first time the word `process` appears, that is because they are one of the most important concepts in programming.

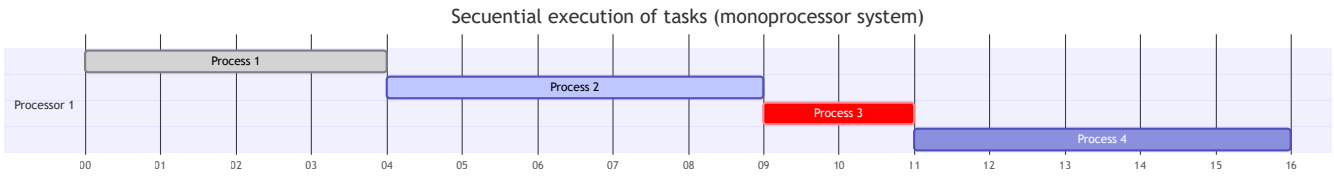
1.2.1. Concurrency vs Parallelism

Now that we already know what a process is, let's look at its relationship with the hardware where they are run.

Monoprocess

Maybe we want to run many processes at the same time, but if we only have one processor unit, it's absolutely impossible to have more than one task running at the same time.

One possibility is to run the tasks in sequence. The system starts running one process and it doesn't start with the next one until the current task has completely finished. That is what happens in systems that are able to do one task at a time, something really strange nowadays.

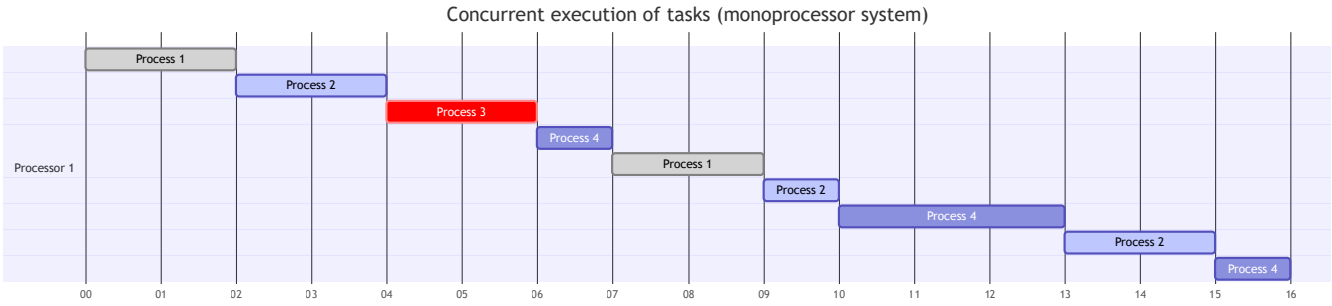


Multitasking

In a multiprogramming system there are one or more programs loaded in main memory which are ready to execute. Only one program at a time is able to get the CPU for executing its instructions while all the others are waiting their turn.

The main idea of multiprogramming is to maximize the use of CPU time. Indeed, suppose the currently running process is performing an I/O task (which, by definition, does not need the CPU to be accomplished). Then, the OS may interrupt that process and give the control to one of the other in-main-memory programs that are ready to execute (i.e. process context switching). It is said that running processes are multiplexed on time.

This way, the OS gives as the illusion that many processes are running simultaneously. That is commonly called `multitasking`.

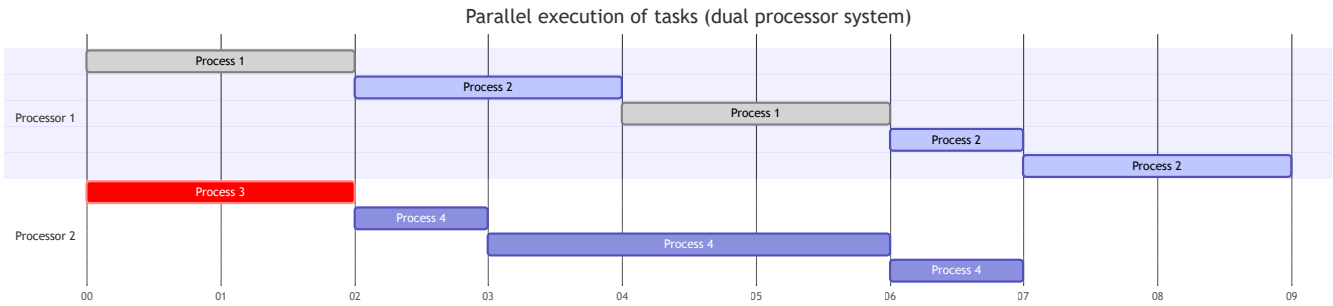


On both previous images can be observed how the total CPU time to complete all processes is the same in both models. Nonetheless, on the second model the user has the sensation that all tasks are running at the same time.

Parallelism

Multiprocessing sometimes refers to executing multiple processes (programs) at the same time. This might be misleading because we have already introduced the term “multiprogramming” to describe that before. In fact, multiprocessing refers to the hardware (i.e., the CPU units) rather than the software (i.e., running processes). If the underlying hardware provides more than one processor then that is multiprocessing. Several variations on the basic scheme exist, e.g., multiple cores on one die or multiple dies in one package or multiple packages in one system. Anyway, a system can be both multiprogrammed by having multiple programs running at the same time and multiprocessing by having more than one physical processor.

Nowadays most devices, from desktop to laptops through mobile devices and IoT, all of them offer multiprocess capabilities, that is, they have mor than one processing unit to really do many tasks at the same time, no simulate it. This kind of execution is called `parallelism`.



The bigger the processor units number is, the less time the tasks it takes to run and the user will have a better experience. This is one of the goals of operating systems, schedule properly the tasks to minimize running times, wait times and to maximize the resources use , mainly the processors.

cores vs threads

have you bought a microprocessor recently? Are you up-to-date in the state of the art of hardware? Then, you'll probably know that one of the main characteristics of a microprocessor are its **cores number** (4, 8, 16).

Moreover, the number of cores is completed with another configuration, **treads number**, that usually is twice the number of cores.

¿What is the relationship between processor threads and concurrency? ¿A computer system with 8 cores / 16 threads means that it can run up to 16 process in parallel?

1.2.2. Distributed systems

> "A collection of independent computers that appears to its users as a single. coherent system"
> "Andrew S. Tanembaum"

This definition has several important aspects:

- The first one is that a distributed system consists of components (i.e., computers) that are autonomous.
- A second aspect is that users (people or programs) think they are dealing with a single system. This means that one way or the other the autonomous components need to collaborate. How to establish this collaboration lies at the heart of developing distributed systems

The most known and famous example of distributed system is `the Internet`. The Internet is seen by the users as a single huge documents repository, that is, a single system able to provide almost any information or service. Notwithstanding the above, we know that is made up of millions of devices located all over the world and interconnected.

It began with the need to share resources. Actually the state-of-the-art on these systems are `Cloud Computing` or cloud services. It's said that a distributed system is where software components are distributed on a network and they communicate and coordinate with each other by using message passing.

Let's concentrate on important characteristics of distributed systems:

- Concurrency. Allows running multiple process in parallel.
- Global watch independency. Implies synchronization using messaging.
- Scalability: distributed systems should also be relatively easy to expand or scale. This characteristic is a direct consequence of having independent computers, but at the same time, hiding how these computers actually take part in the system as a whole.
- Fault tolerance: A distributed system will normally be continuously available, although perhaps some parts may be temporarily out of order.

1.2.3. Advantages and disadvantages

Pros of parallel processing:

- Simultaneous running of tasks
- Reduce total running time
- Helps to solve big and complex problems
- Use of non local resources on the network
- Reduce expenses by taking advantage of shared resources. It's not necessary to invest on a supercomputer because it's possible to have the same processing power with smaller computers distributed

Cons of parallel processing:

- Compilers and development environments are more complex to develop.
- Parallel programs are more difficult to write
- Higher power consumption
- Bigger data access complexity
- High communication and synchronization complexity on subtasks warning

Pros of distributed programming

- Resource & data sharing
- Scale under demand
- Bigger flexibility to distribute processing load
- High availability
- Support for distributed applications
- Open philosophy and heterogeneous development

> **Scalability**
>
> Scalability means the possibility of increase the processing, storage, and capabilities of a system
>
> Research about differences, pros and cons of `vertical scaling` vs `horizontal scaling`.

Cons of distributed programming

- Increase system complexity
- New specialized software is needed
- Communication problems (data lost, overflows, saturation, etc.)
- Security problems, DDoS attacks

Distributed and parallel programming examples

- Weather forecast analysis and research

- Human genome research
- Biosphere modelling
- Seismic predictions
- Molecule simulation

Example of parallel and distributed programming

Allien intelligence research- SETI Project

1.2.4. Bernstein's conditions

Once we know what a concurrent program is and the different hardware architectures that support them, let's try to identify which program parts can be executed concurrently.

If we look at the following code we can determine that the first sentence must be run before the second one in order to get always the same result (for the same input data set).

```java
x = x + 1;
y = x + 1;
```

Though in the following code the order isn't important at all and doesn't change the final result (output data set). In this situation all of them cam be run at the same time increasing processing speed.

```java
x = 1;
y = 2;
z = 3;
```

A.J. Bernstein's Conditions are the conditions applied on two statements S1 and S2 that are to be executed in the processor. It states that three conditions that are explained below must be satisfied for two successive statements S_i y S_j to be executed concurrently and still produce the same result

Bernstein conditions are rely on the subsequent two sets of variables:

- $R(S_k) = \{a_1, a_2, a_3, ...\}$ read set consists of all variables that are read during exceution of k statements set
- $W(S_k) = \{b_1, b_2, b_3, ...\}$ write set consists of all variables that are written (updated) during execution of k statements set.

To run concurrently both statement sets S_i y S_j these three conditions must be matched simultaneously:

- $R(S_i) \cap W(S_j)$
- $W(S_i) \cap R(S_j)$
- $W(S_i) \cap W(S_j)$

Which of these instructions can be run concurrently?

```java
a = x + y;
b = z - 1;
c = a - b;
w = c + 1;
```

First of all we must get L & E sets for each sentence

> $R(S_1) = \{x, y\}$
> $W(S_1) = \{a\}$

> $R(S_2) = \{z\}$
> $W(S_2) = \{b\}$

> $R(S_3) = \{a, b\}$
> $W(S_3) = \{c\}$

> $R(S_4) = \{c\}$
> $W(S_4) = \{w\}$

And now let's apply the rules for each pair of sentences

> $R(S_1) \cap W(S_2) = \emptyset$
> $W(S_1) \cap R(S_2) = \emptyset$
> $W(S_1) \cap W(S_2) = \emptyset$ // They can be run simultaneously

> $R(S_1) \cap W(S_3) = \emptyset$
> $W(S_1) \cap R(S_3) = \{a\} \neq \emptyset$
> $W(S_1) \cap W(S_3) = \emptyset$ // NO parallelism without problems

> $R(S_1) \cap W(S_4) = \emptyset$
> $W(S_1) \cap R(S_4) = \emptyset$
> $W(S_1) \cap W(S_4) = \emptyset$ // They can be run simultaneously

> $R(S_2) \cap W(S_3) = \emptyset$
> $W(S_2) \cap R(S_3) = \{b] \neq \emptyset$
> $W(S_2) \cap W(S_3) = \emptyset$ // NO parallelism without problems

> $R(S_2) \cap W(S_4) = \emptyset$
> $W(S_2) \cap R(S_4) = \emptyset$
> $W(S_2) \cap W(S_4) = \emptyset$ // They can be run simultaneously

> $R(S_3) \cap W(S_4) = \emptyset$
> $W(S_3) \cap R(S_4) = \{c\} \neq \emptyset$
> $W(S_3) \cap W(S_4) = \emptyset$ // NO parallelism without problems