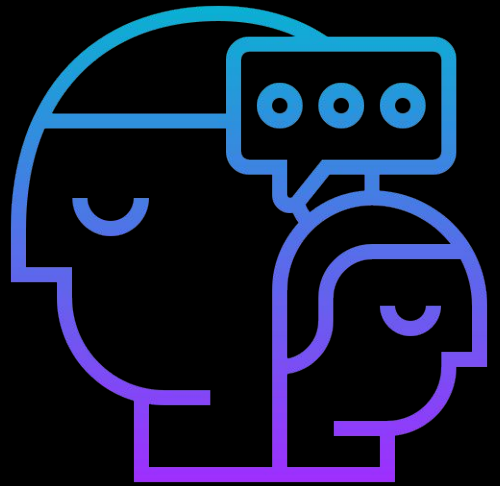
Programación concurrente

Johan Sebastian Salamanca Gonzalez
Mauricio Meza Burbano
Cesar Augusto Torres Ardila

Contenido

- Filosofía
- Conceptos claves
- Ventajas y desventajas
- Lenguajes de programación
- Ejemplos
- Aplicaciones
- Referencias

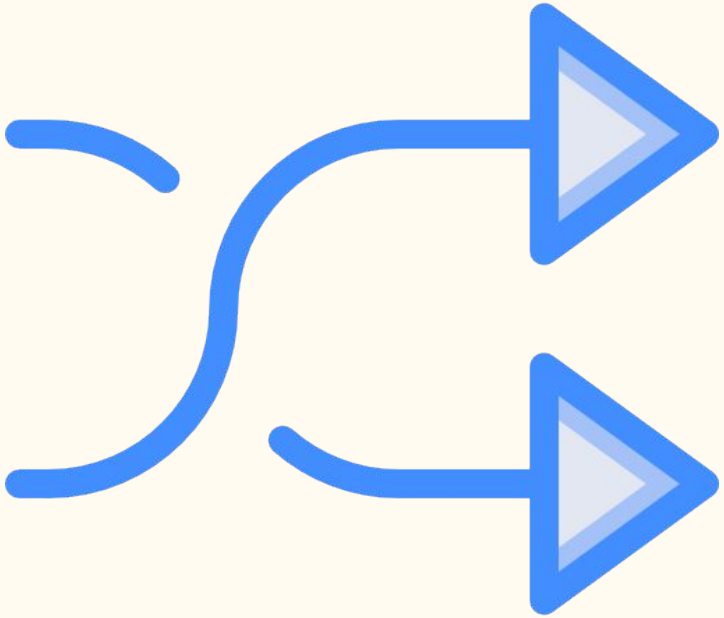
Filosofía



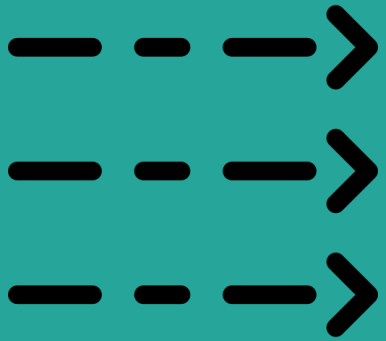
Ejecución simultánea de tareas interactivas

Definición

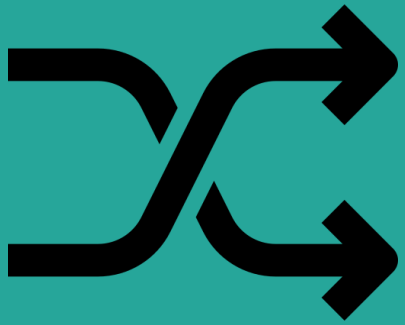
Es un conjunto de técnicas que permiten la ejecución concurrente entre tareas que pueden interactuar entre sí, de esta forma se tiene un conjunto de procesos o hilos que se sincronizan y comunican entre sí.



Programación paralela



Programación concurrente



Programación secuencial

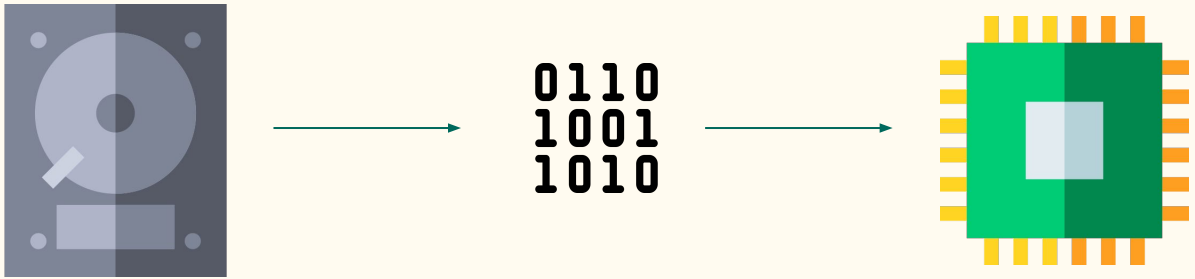

Cola de Procesador
Proceso 1
Proceso 2
Proceso 3
Proceso 1
Proceso 2
Proceso 3
Proceso 1
Proceso 2
Proceso 3
Resultado Proceso 1
Resultado Proceso 2
Resultado Proceso 3

Conceptos claves

Programas y Procesos

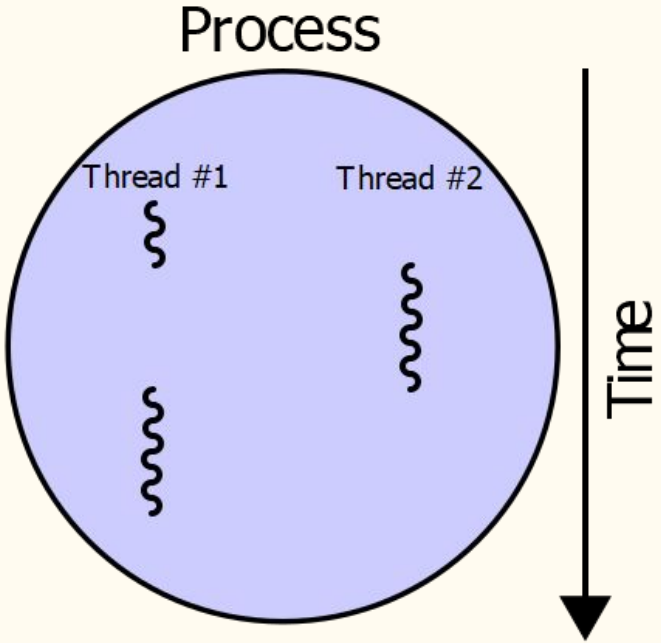
Un programa es un conjunto de instrucciones guardadas en disco que se deben realizar en una CPU.

Cuando una instancia de este programa se carga en memoria y es ejecutado en un procesador pasa a ser un proceso.



Hilos

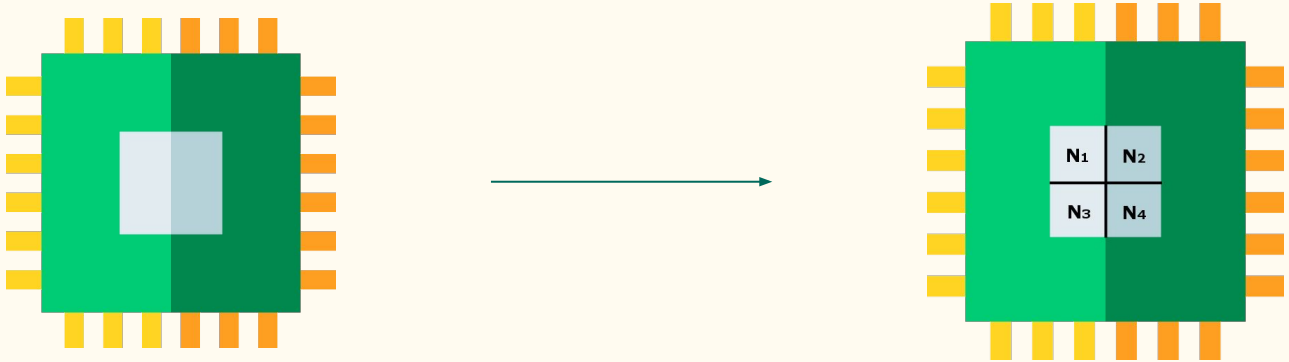
Representa la unidad básica de uso de CPU donde un proceso se puede componer de uno o más hilos que ejecutan instrucciones del programa y se usan para poder utilizar concurrencia en un proceso singular.



Procesadores y Núcleos

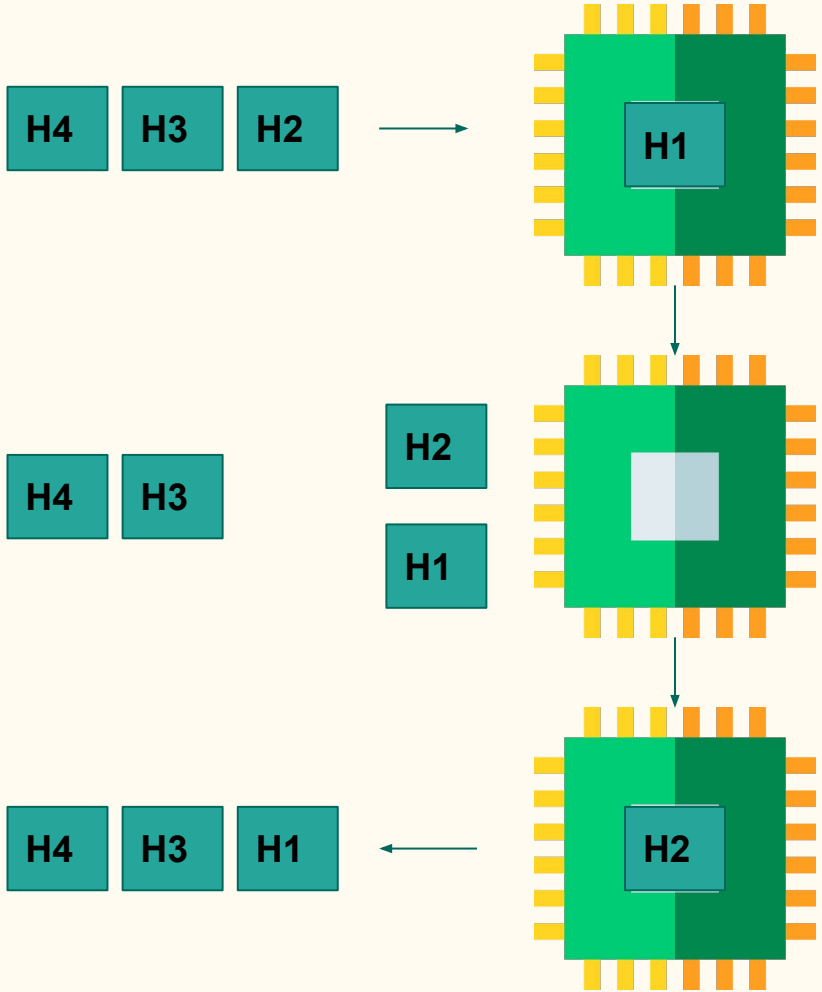
Un procesador es el chip fisico que hace parte del computador que contiene la CPU que se define como el hardware que permite realizar instrucciones

Cada CPU puede contener uno o más núcleos los cuales son los encargados de ejecutar las instrucciones de cada proceso que llega a la CPU



Cambio de Contexto

Consiste en el proceso realizado por el Sistema Operativo de tomar un proceso o hilo ejecutándose en un núcleo de CPU, guardar el estado de la última instrucción, detener su ejecución, y finalmente colocar otro hilo o proceso en el núcleo para su ejecución.



Condición de carrera

También llamada condición de secuencias una expresión que se usa cuando el resultado de la ejecución de un proceso, depende del orden en que se sucedieron eventos anteriores.

Sección crítica

Secciones del código de los procesos que no pueden ser ejecutadas de forma concurrente ya que causaría errores o funcionamientos inesperado al tener variables o recursos que son accedidos por más de un proceso a la vez.

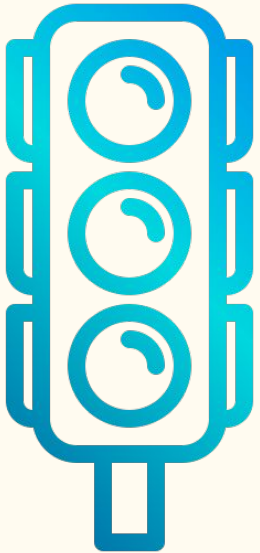
Mutua exclusión

También llamada Mutex, consiste en inhabilitar las secciones críticas del código cuando la ejecución de un proceso ha llegado a dicha seccion de codigo y volver a habilitarla cuando se termine de ejecutar el código en dicha sección

Semáforos

Son indicadores de condición que indican si un recurso está disponible o no antes de ser accedido

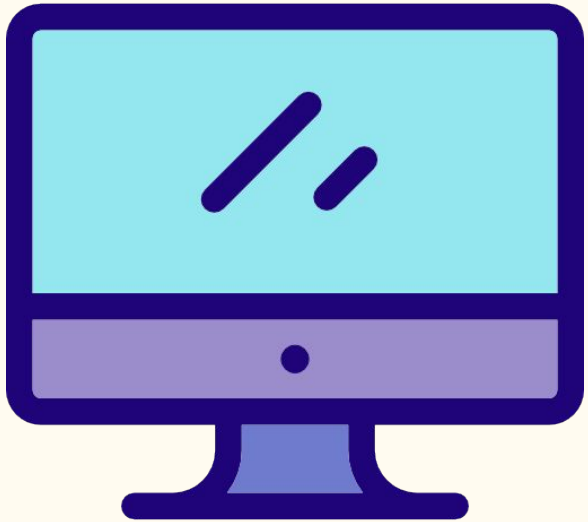
- Semáforos binarios
- Semáforos generales



Monitores

Estructura de dato abstracta compuesta de un identificador, un conjunto de variables globales, conjunto de procedimientos y un bloque de inicialización.

Pueden ser utilizadas por más de un hilo ya que los procedimientos se ejecutan con exclusión mutua.



Mensajes

Son cadenas de texto transmitidas entre procesos que permiten la sincronización y comunicación entre procesos.

- Mensajes directos
- Mensajes indirectos
- Mensajes implícitos

Buzones

Es una estructura de datos dedicada a almacenar mensajes, la cual tiene un tamaño definido.

- Buzones propios de un proceso
- Buzones del Sistema operativo

Ventajas y desventajas

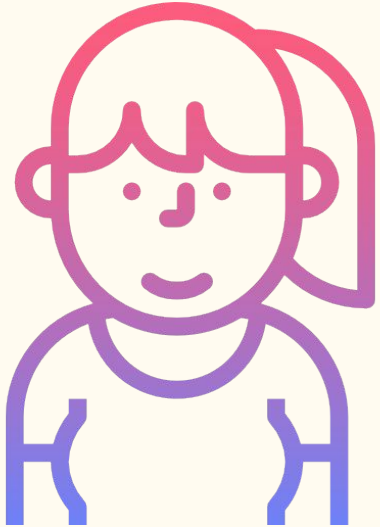
Ventajas

Optimización

- Optimiza los recursos computacionales de procesamiento aprovechando la CPU y cada uno de sus núcleos al máximo.
- Esto permite un mayor tiempo de ejecución que un programa ejecutado secuencialmente

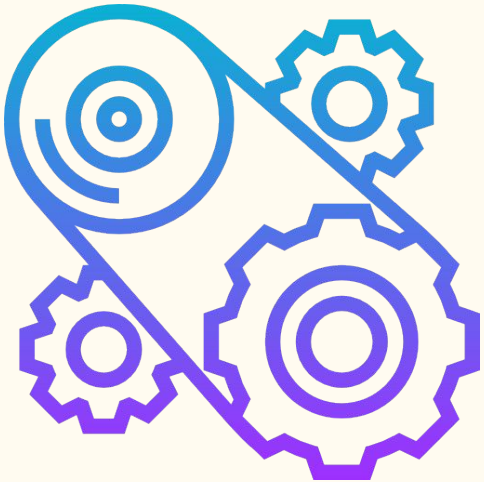
Responsividad al usuario

- La concurrencia le permite a un programa seguir ejecutándose incluso si parte de él está realizando una operación que lleva tiempo.
- Así el programa esta ocupado pero sigue estando responsivo al usuario



Escalabilidad

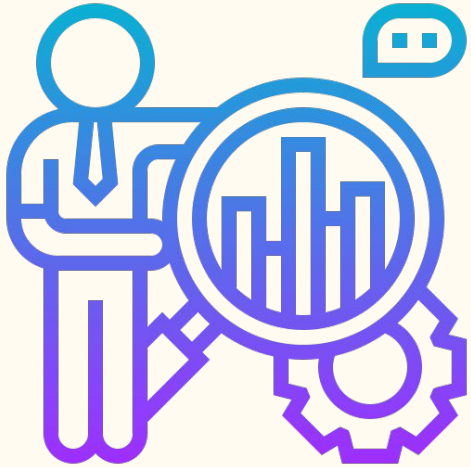
- En un sistema multiprocesador las ventajas se multiplican ya que se puede combinar la concurrencia con paralelización y permitir a varios hilos ejecutarse paralelamente en varios núcleos sin necesidad de cambiar demasiado la implementación de los hilos



Desventajas

Implementación

- Es necesario examinar cuidadosamente el programa para encontrar qué tareas pueden dividirse en hilos concurrentes
- Una concurrencia mal implementada podría aumentar el tiempo de ejecución de un programa

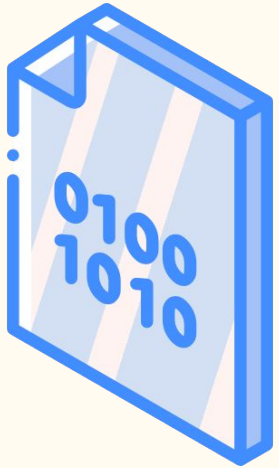


Pruebas y Depuración

- Un programa que se ejecuta concurrentemente puede ejecutarse en diferentes órdenes generando muchas posibilidades de ejecución.
- Esto dificulta el trabajo del programador a la hora de realizar testing y depuración del código

Manejo de Datos

- Se pueden generar error cuando varios hilos comparten los mismos datos y estos intentan acceder a estos al mismo tiempo.
- Asimismo la modificación de datos combinado con el orden en el que se modifican pueden generar errores no esperados.



Lenguajes de programación y ejemplos

Java

- Java permite realizar programación concurrente a través de hilos (threads).
- Los hilos son procesos ligeros, que se ejecutan fuera de la línea de flujo actual.
- Los cambios de contexto son menos costosos en tiempo de ejecución.

```java
package javaapplication4;

import static java.lang.Thread.sleep;

public class ParalelismExample {

    // Procesos a ejecutar
    public static void processes(){
        try {
            for (int i = 0; i < 10; i++) {
                System.out.println("Hilo " + Thread.currentThread().getName() + ": proceso " + i);
                sleep((long) (1000*Math.random()));
            }
        } catch (InterruptedException ex) {
        }
    }

    public static void main(String[] args) {

        // Crear hilos
        Thread treads[] = new Thread[10];
        for (int i = 0; i < treads.length; i++) {
            treads[i] = new Thread(String.valueOf(i)) {
                public void run(){
                    processes();
                }
            };
        }

        // Iniciar hilos
        for (int i = 0; i < treads.length; i++) {
            treads[i].start();
        }

    }

}
```

Paralelismo

Hilo 0: proceso 0
Hilo 2: proceso 0
Hilo 1: proceso 0
...
Hilo 3: proceso 1
Hilo 1: proceso 1
Hilo 8: proceso 2
Hilo 6: proceso 1
....
Hilo 5: proceso 9
Hilo 4: proceso 8
Hilo 3: proceso 9
Hilo 4: proceso 9

```java
package javaapplication4;

public class ConcurrenceExample extends Thread {

    // Procesos a ejecutar
    public static void processes(){
        try {
            for (int i = 0; i < 10; i++) {
                System.out.println("Hilo " + Thread.currentThread().getName() + ": proceso " + i);
                sleep((long) (1000*Math.random()));
                synchronized(Thread.currentThread()){ Thread.currentThread().wait();}
            }
        } catch (InterruptedException ex) {
        }
    }

    public static void main(String[] args) {

        // Crear hilos
        Thread treads[] = new Thread[10];
        for (int i = 0; i < treads.length; i++) {
            treads[i] = new Thread(String.valueOf(i)) {
                public void run(){
                    processes();
                }
            };
        }

        // Iniciar hilos
        for (int i = 0; i < treads.length; i++) {
            treads[i].start();
        }
```

Concurrencia

Hilo 0: proceso 0
Hilo 2: proceso 0
Hilo 1: proceso 0
...
Hilo 9: proceso 5
Hilo 8: proceso 5
Hilo 2: proceso 6
Hilo 6: proceso 6
....
Hilo 3: proceso 9
Hilo 1: proceso 9
Hilo 2: proceso 9
Hilo 8: proceso 9

```
        // Sincronizar hilos
        while (true) {

            // Verificar si todos los hilos han terminado un proceso
            boolean processedAll = true;
            for (int i = 0; i < treads.length; i++) {
                if (treads[i].getState() != Thread.State.WAITING) {
                    processedAll = false;
                    break;
                }
            }

            // Si todos han terminado un proceso continuar con el siguiente
            if (processedAll) {
                for (int i = 0; i < treads.length; i++) {
                    synchronized(treads[i]){ treads[i].notify();}
                }
            }

            // Si ya se han acabado todos los procesos
            if (treads[0].getState() == Thread.State.TERMINATED) {
                break;
            }

        }

    }
}
```

Concurrencia

Hilo 0: proceso 0
Hilo 2: proceso 0
Hilo 1: proceso 0
...
Hilo 9: proceso 5
Hilo 8: proceso 5
Hilo 2: proceso 6
Hilo 6: proceso 6
....
Hilo 3: proceso 9
Hilo 1: proceso 9
Hilo 2: proceso 9
Hilo 8: proceso 9

C++

- Al igual que java puede realizar concurrencia por medio de hilos, aunque no de forma nativa.
- Cuando se creó el estándar inicial de C++, el mecanismo de concurrencia fue excluído de forma explícita porque C no tenía uno.
- Actualmente existe una librería “thread” que permite programar concurrentemente.

```cpp
#include <iostream>
#include <thread>
#include <chrono>
#include <mutex>
#include <stdlib.h>

using namespace std;

// Procesos a ejecutar
void processes(int id) {
    for (int i = 0; i < 10; i++) {
        cout << "Hilo " << id << ": proceso " << i << endl;
        this_thread::sleep_for(chrono::milliseconds((int) (1000*(((double) rand() / (RAND_MAX))))));
    }
}

int main() {

    // Crear e inciar hilos
    thread treads[10];
    for (int i = 0; i < 10; i++) {
        treads[i] = thread(processes, i);
    }

    // Sincronizar hilos
    for (int i = 0; i < 10; i++) {
        treads[i].join();
    }

    return 0;
}
```

Paralelismo

Hilo 3: proceso 0
Hilo 4: proceso 0
Hilo 5: proceso 0
...
Hilo 4: proceso 2
Hilo 2: proceso 3
Hilo 6: proceso 2
Hilo 1: proceso 3
....
Hilo 1: proceso 9
Hilo 6: proceso 9
Hilo 9: proceso 8
Hilo 9: proceso 9

```cpp
#include <iostream>
#include <thread>
#include <chrono>
#include <mutex>
#include <condition_variable>
#include <stdlib.h>

using namespace std;

mutex mtx;
condition_variable cv;

int num = 10;
thread treads[10];
bool waiting[10];
bool finished[10];

// Procesos a ejecutar
void processes(int id) {
    for (int i = 0; i < num; i++) {
        cout << "Hilo " << id << ": proceso " << i << endl;
        this_thread::sleep_for(chrono::milliseconds((int) (1000*(((double) rand() / (RAND_MAX))))));
        waiting[id] = true;
        unique_lock<mutex> lck(mtx);
        while (waiting[id]) cv.wait(lck);
    }
    finished[id] = true;
}

void sinc(){

    // Sincronizar hilos
    while (true) {

      // Verificar si todos los hilos han terminado un proceso
      bool processedAll = true;
      for (int i = 0; i < num; i++) {
          if (!waiting[i]) {
              processedAll = false;
              break;
          }
      }
```

Concurrencia

Hilo 0: proceso 0
Hilo 2: proceso 0
Hilo 3: proceso 0
...
Hilo 7: proceso 3
Hilo 1: proceso 3
Hilo 6: proceso 4
Hilo 5: proceso 4
....
Hilo 3: proceso 9
Hilo 4: proceso 9
Hilo 9: proceso 9
Hilo 0: proceso 9

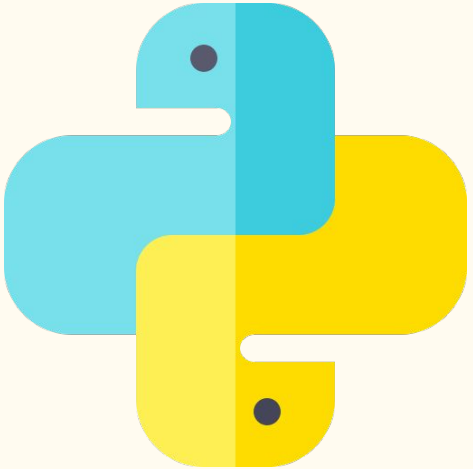
```
        // Si todos han terminado un proceso continuar con el siguiente
        if (processedAll) {
            unique_lock<mutex> lck(mtx);
            for (int i = 0; i < num; i++) {
                waiting[i] = false;
            }
            cv.notify_all();
        }

        // Si ya se han acabado todos los procesos
        if (finished[0]) {
            break;
        }

    }
}

int main() {

    // Crear e inciar hilos
    thread treads[num];
    thread treadSinc;
    for (int i = 0; i < num; i++) {
        waiting[i] = false;
        finished[i] = false;
        treads[i] = thread(processes, i);
    }
    treadSinc = thread(sinc);

    for (int i = 0; i < num; i++) {
        treads[i].join();
    }

    // Sincronizar hilos
    treadSinc.join();

    return 0;
}
```

Concurrencia

Hilo 0: proceso 0
Hilo 2: proceso 0
Hilo 3: proceso 0
...
Hilo 7: proceso 3
Hilo 1: proceso 3
Hilo 6: proceso 4
Hilo 5: proceso 4
....
Hilo 3: proceso 9
Hilo 4: proceso 9
Hilo 9: proceso 9
Hilo 0: proceso 9

Python

- Python no es realmente multitarea.
- Python tiene librerías que le permiten realizar programación paralela y concurrente por medio de hilos y procesos.
- Python tiene una librería “threading” que permite programar concurrentemente.



Paralelismo

```python
from time import sleep
import random
import threading

# Procesos a ejecutar
def processes(id):
    for i in range(10):
        print("Hilo ", id, ": proceso ", i)
        sleep(random.random())

# Crear hilos
treads = []
for i in range(10):
    treads.append(threading.Thread(target=processes, args=(i,)))

# Iniciar hilos
for i in range(10):
    treads[i].start()
```

Hilo 0 : proceso 0
Hilo 1 : proceso 0
Hilo 2 : proceso 0
...
Hilo 5 : proceso 5
Hilo 2 : proceso 3
Hilo 4 : proceso 4
Hilo 1 : proceso 2
....
Hilo 6 : proceso 8
Hilo 1 : proceso 8
Hilo 6 : proceso 9
Hilo 1 : proceso 9

```python
from time import sleep
import random
import threading

treads = []
waiting = []

# Procesos a ejecutar
def processes(id):
    for i in range(10):
        print("Hilo " + str(id) + ": proceso " + str(i) + "\n", end='')
        sleep(random.random())
        waiting[id] = True
        while waiting[id]: pass

# Crear hilos
for i in range(10):
    waiting.append(False)
    treads.append(threading.Thread(target=processes, args=(i,)))

# Iniciar hilos
for i in range(10):
    treads[i].start()

# Sincronizar hilos
while True:

    # Verificar si todos los hilos han terminado un proceso
    processedAll = True
    for i in range(10):
        if not waiting[i]:
            processedAll = False
            break

    # Si todos han terminado un proceso continuar con el siguiente
    if processedAll:
        for i in range(10):
            waiting[i] = False

    # Si ya se han acabado todos los procesos
    if not treads[0].isAlive():
        break
```

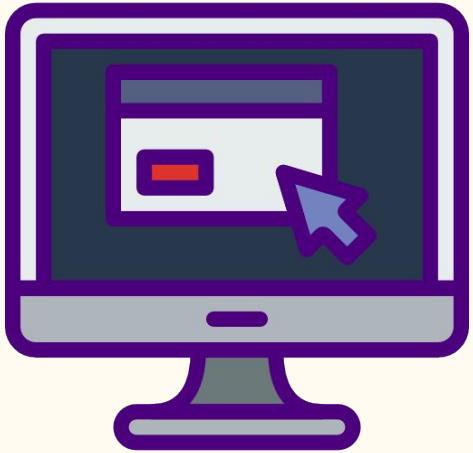
Concurrencia

Hilo 0: proceso 0
Hilo 1: proceso 0
Hilo 2: proceso 0
...
Hilo 3: proceso 4
Hilo 2: proceso 4
Hilo 2: proceso 5
Hilo 4: proceso 5
....
Hilo 3: proceso 9
Hilo 4: proceso 9
Hilo 6: proceso 9
Hilo 0: proceso 9

Aplicaciones

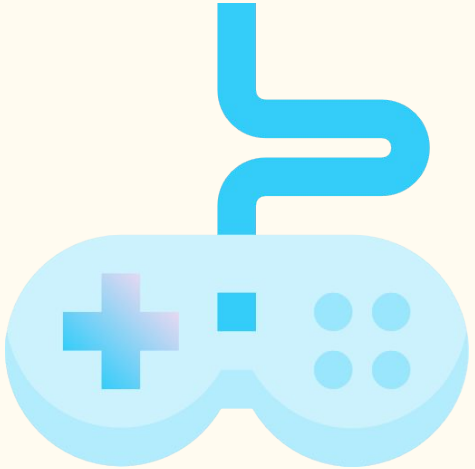
Aplicaciones

- Servidores Web
- Sistema multimedia
- Cálculo numérico
- Interacción por GUI
- Sistemas gestores de bases de datos



Aplicaciones

- Sistemas Operativos
- Sistemas de control
- Simulación
- Videojuegos



Referencias

https://webprogramacion.com/44/sistemas-operativos/exclusion-mutua.asp
https://es.wikipedia.org/wiki/Computaci%C3%B3n_concurrente
https://www2.ulpgc.es/hege/almacen/download/20/20233/tema1.pdf
https://www.monografias.com/trabajos106/programacion-concurrente/programacion-concurrente.shtml
https://www.fing.edu.uy/tecnoinf/mvd/cursos/so/material/teo/so07-concurrencia.pdf
Operating System Concepts, 10th Edition. Abraham Silberschatz, Greg Gagne, Peter B
https://www.ctr.unican.es/asignaturas/procodis_3_ii/doc/procodis_3_01.pdf
http://arco.inf-cr.uclm.es/~david.villa/pensar_en_C++/vol2/ch10s02.html
https://devnewbies.wordpress.com/2016/05/23/concurrencia-en-c/
https://code.tutsplus.com/es/articles/introduction-to-parallel-and-concurrent-programming-in-python--cms-28612
http://ferestrepoca.github.io/paradigmas-de-programacion/progconcurrente/concurrente_teoria/index.html
http://lsi.vc.ehu.es/pablogn/docencia/manuales/SO/TemasSOuJaen/CONCURRENCIA/3SinUtilizarMemoriaCompartida.htm