Programación Científica y HPCI

Máster Universitario en Ingeniería Matemática y Computación

Tema 8 / Programación Paralela - HPC

unir LA UNIVERSIDAD EN INTERNET

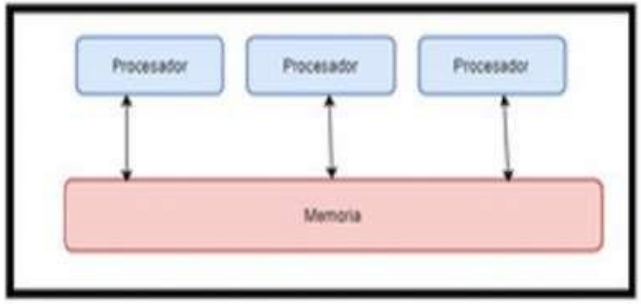
Universidad Internacional de La Rioja

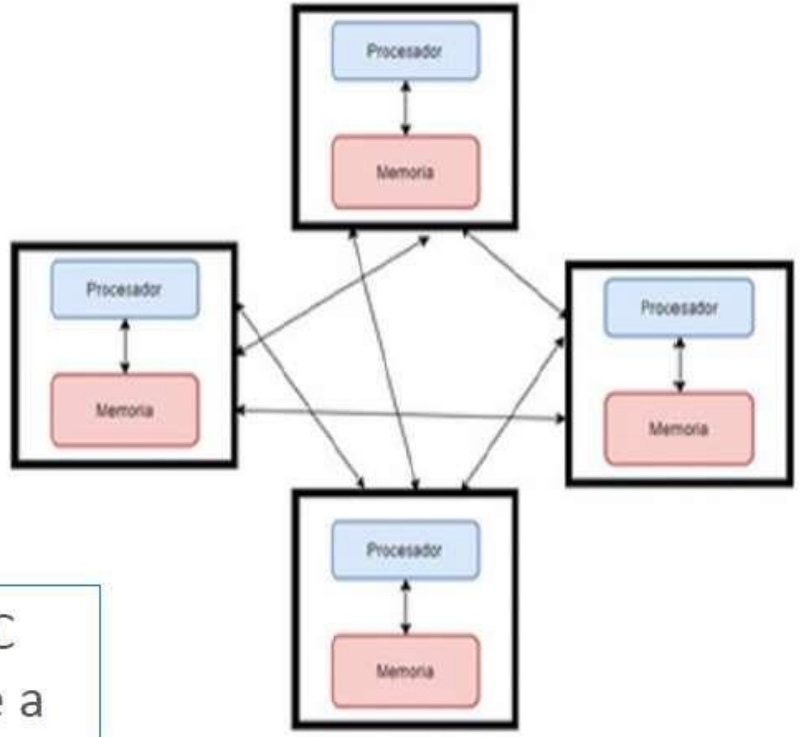
Conceptos básicos

Multiprogramación →gestión de procesos en un sistema monoprocesador.
Multiprocesamiento→ gestión de procesos en un sistema multiprocesador (puede existir memoria común).
Procesamiento distribuido → gestión de procesos en un procesadores separados (memoria no compartida).

Programación concurrente →acciones que pueden ser ejecutadas de forma simultanea.
Programación paralela→ programación concurrente en un sistema multiprocesador.
Programación distribuida→ programación paralela en un sistema distribuido.

Sistema de multiprocesamiento y sistema distribuido





computación de altas prestaciones o HPC (*high performance computing*) equivalente a computación paralela

Fuente: https://es.wikipedia.org/wiki/Computación_distribuida

Enfoques y aspectos importantes

- Paralelismo a nivel de tareas→*fork-join*
- Paralelismo a nivel de datos→SMPD(*Single Program Multiple Data*)

- La sincronización de las distintas tareas que conforman un algoritmo que se ejecuta en paralelo
- El seguimiento de los pasos de la computación
- El registro o la transmisión de datos desde varios dispositivos de computación.

Módulo Processing

Elementos	Descripción
Clase Process	definición de procesos
Clase Pool	creación de conjuntos de procesos → invocación paralela a una función
Métodos	Inicialización, activación y consulta de estados
Objetos para la sincronización	Cerrojos y condiciones de espera
Objetos para la comunicación	Colas, pipes, variables y arrays compartidos.
Excepciones	Problemas durante la ejecución

Definición de procesos

Objetos de Process	Objetos clases derivadas de Process
Definición de objetos	Definición de la clase y de objetos
Inicialización con constructor de Process	Inicialización con constructor de clases derivadas
Asociar una función para su ejecución	Sobrecarga del método run()
Activar hilo start()	
Invocar join() para espera a su finalización	

```
import os
import time
import threading
import multiprocessing

NUMERO_ACTIVIDADES = 6

def dormir():
    """ solo espera a que pase el tiempo"""
    print("IDProceso: %s, Nombre  Proceso: %s, Nombre Hilo: %s \n" % (
        os.getpid(),
        multiprocessing.current_process().name,
        threading.current_thread().name)
    )
    "se obtiene el identificador del proceso actual, el nombre del proceso"
    "que se ejecuta y el nombre del hilo"
    time.sleep(1)


if __name__=='__main__':

    processes = [multiprocessing.Process(target=dormir) for _ in range(NUMERO_ACTIVIDADES)]
    [process.start() for process in processes]
    [process.join() for process in processes]
```

```
import time
from multiprocessing import Process


class ProcesoDurmiente(Process):

    def run(self):

        print("El proceso que va a dormir es "+ self.name)
        time.sleep(3)



if __name__ == '__main__':
    proceso1 = ProcesoDurmiente()
    proceso2 = ProcesoDurmiente(name="Proceso 2")
    proceso1.start()
    proceso2.start()

    proceso1.join()
    proceso2.join()
```

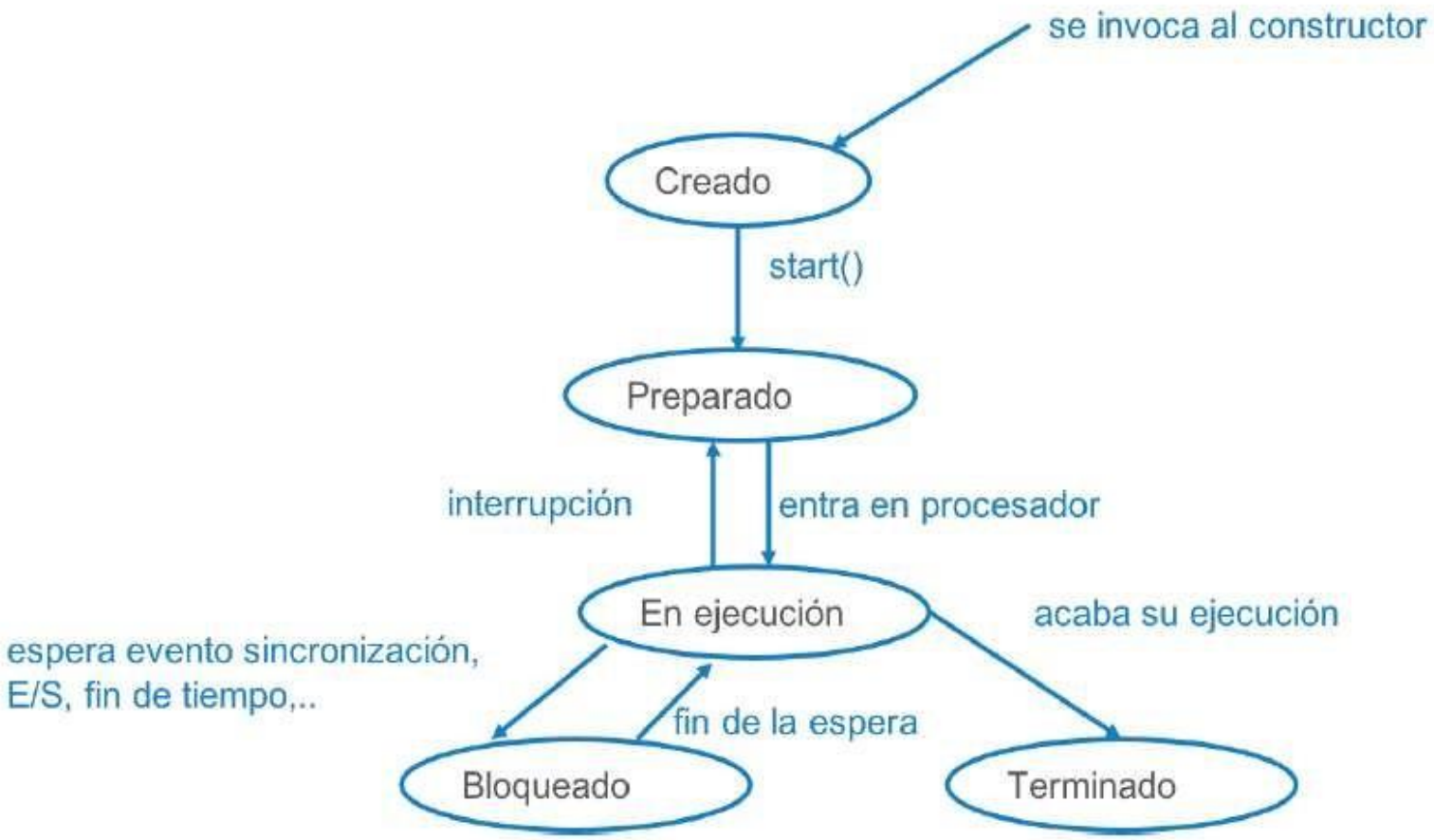
```python
import os
import time
import threading
import multiprocessing

NUMERO_ACTIVIDADES = 6

def dormir():
    """ solo espera a que pase el tiempo"""
    print("IDProceso: %s, Nombre  Proceso: %s, Nombre Hilo: %s \n" % (
        os.getpid(),
        multiprocessing.current_process().name,
        threading.current_thread().name)
    )
    "se obtiene el identificador del proceso actual, el nombre del proceso"
    "que se ejecuta y el nombre del hilo"
    time.sleep(1)



if __name__=='__main__':
    multiprocessing.freeze_support()
    tiempo_inicial = time.time()
    for _ in range(NUMERO_ACTIVIDADES):
        dormir()
    tiempo_final= time.time()

    print("Tiempo ejecución secuencial=", tiempo_final - tiempo_inicial)

    tiempo_inicial = time.time()
    threads = [threading.Thread(target=dormir) for _ in range(NUMERO_ACTIVIDADES)]
    [thread.start() for thread in threads]
    [thread.join() for thread in threads]
    tiempo_final = time.time()

    print("Tiempo ejecución concurrente=", tiempo_final - tiempo_inicial)


    tiempo_inicial = time.time()
    processes = [multiprocessing.Process(target=dormir) for _ in range(NUMERO_ACTIVIDADES)]
    [process.start() for process in processes]
    [process.join() for process in processes]

    tiempo_final = time.time()
```

```
IDProceso: 45572, Nombre  Proceso: MainProcess, Nombre Hilo: MainThread

IDProceso: 45572, Nombre  Proceso: MainProcess, Nombre Hilo: MainThread

Tiempo ejecución secuencial= 6.019713878631592
IDProceso: 45572, Nombre  Proceso: MainProcess, Nombre Hilo: Thread-23

IDProceso: 45572, Nombre  Proceso: MainProcess, Nombre Hilo: Thread-24

IDProceso: 45572, Nombre  Proceso: MainProcess, Nombre Hilo: Thread-25

IDProceso: 45572, Nombre  Proceso: MainProcess, Nombre Hilo: Thread-26

IDProceso: 45572, Nombre  Proceso: MainProcess, Nombre Hilo: Thread-27

IDProceso: 45572, Nombre  Proceso: MainProcess, Nombre Hilo: Thread-28

Tiempo ejecución concurrente= 1.0076842308044434
IDProceso: 45908, Nombre  Proceso: Process-7, Nombre Hilo: MainThread

IDProceso: 45909, Nombre  Proceso: Process-8, Nombre Hilo: MainThread

IDProceso: 45910, Nombre  Proceso: Process-9, Nombre Hilo: MainThread

IDProceso: 45912, Nombre  Proceso: Process-11, Nombre Hilo: MainThread

IDProceso: 45911, Nombre  Proceso: Process-10, Nombre Hilo: MainThread

IDProceso: 45913, Nombre  Proceso: Process-12, Nombre Hilo: MainThread

Tiempo  ejecución paralela= 1.1363909244537354
```

```python
import os
import time
import threading
import multiprocessing

NUMERO_ACTIVIDADES = 6

def dormir():
    """ olo es pera a que pase el tiempo"""
    print("IDProceso: %s, Nombre  Proceso: %s, Nombre Hilo: %s" % (
        os.getpid(),
        multiprocessing.current_process().name,
        threading.current_thread().name)
    )
    "se obtiene el identificador del proceso actual, el nombre del proceso"
    "que se ejecuta y el nombre del hilo"
    time.sleep(1)


def  incremento_grande():
    """ Incrementa un número grande de veces  la variable contador """
    print("IDProceso: %s, Nombre  Proceso: %s, Nombre Hilo: %s" %  (
        os.getpid(),
        multiprocessing.current_process().name,
        threading.current_thread().name)
    )
    "se obtiene el identificador del proceso actual, el nombre del proceso"
    "que se ejecuta y el nombre del hilo"
    x = 0
    while x < 10000000:
        x += 1
```

```
Library/Mobile Documents/com~apple~CloudDocs/MAP/EJERCICIOS')
IDProceso: 45572, Nombre  Proceso: MainProcess, Nombre Hilo: MainThread
IDProceso: 45572, Nombre  Proceso: MainProcess, Nombre Hilo: MainThread
IDProceso: 45572, Nombre  Proceso: MainProcess, Nombre Hilo: MainThread
IDProceso: 45572, Nombre  Proceso: MainProcess, Nombre Hilo: MainThread
IDProceso: 45572, Nombre  Proceso: MainProcess, Nombre Hilo: MainThread
IDProceso: 45572, Nombre  Proceso: MainProcess, Nombre Hilo: MainThread
Tiempo ejecución secuencial= 2.8434810638427734
IDProceso: 45572, Nombre  Proceso: MainProcess, Nombre Hilo: Thread-31
IDProceso: 45572, Nombre  Proceso: MainProcess, Nombre Hilo: Thread-32
IDProceso: 45572, Nombre  Proceso: MainProcess, Nombre Hilo: Thread-33
IDProceso: 45572, Nombre  Proceso: MainProcess, Nombre Hilo: Thread-34IDProceso: 45572, Nombre  Proceso: MainProcess, Nombre Hilo:
Thread-35IDProceso: 45572, Nombre  Proceso: MainProcess, Nombre Hilo: Thread-36


Tiempo ejecución concurrente= 2.7601449489593506
IDProceso: 46000, Nombre  Proceso: Process-13, Nombre Hilo: MainThread
IDProceso: 46002, Nombre  Proceso: Process-15, Nombre Hilo: MainThread
IDProceso: 46004, Nombre  Proceso: Process-17, Nombre Hilo: MainThread
IDProceso: 46003, Nombre  Proceso: Process-16, Nombre Hilo: MainThread
IDProceso: 46001, Nombre  Proceso: Process-14, Nombre Hilo: MainThread
IDProceso: 46005, Nombre  Proceso: Process-18, Nombre Hilo: MainThread
Tiempo  ejecución paralela= 0.8633029460906982

In [4]:
```

Estados de un proceso en Python



Métodos de Comunicación. Cola

- Permite almacenamiento
- Operación de inserción atómica → put
- Operación de extracción atómica → get
- Excepciones de cola llena y vacía

```
Proceso Process-43 inserta valor 0.07389663850044603
Proceso Process-44 inserta valor 0.4213060979867054
Proceso Process-45 inserta valor 0.38680167393746046
Proceso Process-46 inserta valor 0.0849998292697206
Proceso Process-47 inserta valor 0.060567422072080324
[0.07389663850044603, 0.4213060979867054, 0.38680167393746046, 0.0849998292697206, 0.060567422072080324]
```

```python
import multiprocessing
from multiprocessing import Process, Queue
import random


def insertar_valor(cola):
    valor = random.random()
    cola.put(valor)
    print ("Proceso "+multiprocessing.current_process().name+ " inserta valor "+str(valor))


if __name__ == "__main__":
    cola = Queue()

    procesos = [Process(target=insertar_valor, args=(cola,)) for _ in range(5)]

    for p in procesos:
        p.start()

    for p in procesos:
        p.join()

    resultado = [cola.get() for _ in procesos]
    print(resultado)
```

Métodos de Comunicación. Pipe

- Comunicación bidireccional entre procesos conectados por la tubería
- Enviar datos→ send()
- Recibir datos→ recv()

ejecución simultánea en extremos distintos

```
from multiprocessing import Process, Pipe
def enviar(conn):
    conn.send(["num_contadores",1000, 2000])
    conn.close()
def recibir(conn):
    print(conn.recv())
    conn.close()
if __name__ == '__main__':
    conexion_receptor, conexion_emisor = Pipe()
    emisor = Process(target=enviar, args=(conexion_emisor,))
    receptor=Process(target=recibir, args=(conexion_receptor,))
    receptor.start()
    emisor.start()


    emisor.join()
    receptor.join()
```

La salida que se muestra es:

```
['num_contadores', 1000, 2000]
```

Métodos de Comunicación. Memoria compartida

- Value→ de tipo básico de CPython
- Array→ sus valores de tipo básico de Cpython

operaciones atómicas

```
from multiprocessing import Process, Value, Array
def calcular_longitudes(pi, valores):
    pi.value = 3.1415927
    for i in range(len(valores)):
        valores[i] = 2*pi.value*valores[i]

def calcular_areas(pi, valores):
    pi.value = 3.1415927
    for i in range(len(valores)):
        valores[i] = pi.value*valores[i]**2

if __name__ == '__main__':
    valor_cte = Value('d', 0.0)
    array_valores = Array('d', range(1,10))
    print(array_valores[:])
    p1 = Process(target=calcular_longitudes, args=(valor_cte, array_valores))
    p2 = Process(target=calcular_areas, args=(valor_cte, array_valores))
    p1.start()
    p2.start()
    p1.join()
    p2.join()
    print (valor_cte.value)
    print (array_valores[:])
```

$$PRIMER\ PROCESO(area)$$
$$3.1415\times1^2 \rightarrow valores[0]$$
$$3.1415\times2^2 = 12.5663 \rightarrow valores[1]$$

$$SEGUNDO\ PROCESO(longitud)$$
$$2\times3.1415\times3.1415 = 19.7392 \rightarrow valores[0]$$
$$2\times3.14\times12.56 = 78.9568 \rightarrow valores[1]$$

```
[1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
3.1415927
[19.73920938538658, 78.95683754154632, 177.65288446847921, 315.8273501661853, 493.4802346346645, 710.6115378739169, 967.2212598839425,
1263.3094006647411, 1598.875960216313]
```

www.unir.net