Tema 2

Comunicación y sincronización entre procesos en Python

Sistemas Distribuidos
Grado en Ingeniería Informática
Universidad Carlos III de Madrid

Contenido

- Repaso de los conceptos de proceso y *threads*
- Concurrencia
- Mecanismos de comunicación
- Mecanismos de sincronización
- Llamadas al sistema POSIX
  - Para comunicación y sincronización de procesos
  - Gestión de threads
- Threads en C
- Threads en Python
- Threads en Java

Threads en Python

- En Python se puede crear un thread usando el módulo threading

```
import threading

def worker(a, b):
    for i in range(10):
       print('worker ' , a, b, i)

threads = []
for i in range(5):
    t = threading.Thread(target=worker, args=(i,i+1,))
    threads.append(t)
    t.start()

for i in range(5):
        print('Esperando por ', i)
        t = threads[i]
        t.join()
```

Algunas funciones del módulo threading

- `threading.active_counts`
  - Devuelve el número de threads actualmente ejecutando
- `threading.current_thread()`
  - Devuelve el objeto thread actual que se corresponde con el que realiza la llamada
- `threading.get_ident()`
  - Devuelve el identificador del thread que realiza la llamada
- `threading.enumerate()`
  - Devuelve la lista de todos los threads actualmente ejecutando

Threads de tipo Daemon

- En Python los threads independiente se denominan Daemon, no es necesario esperar la terminación de estos threads.

```
import threading
Import time

def worker(a, b):
    for i in range(10):
       print('worker ' , a, b, i)

for i in range(5):
    t = threading.Thread(target=worker, name='Daemon,' args=(i,i+1,))
    t.start()

time.sleep(3)
```

Objetos de tipo Lock

- Similares a los mutex

```
import threading

lock = threading.Lock()

def worker(a, b):
    lock.acquire()
    for i in range(10):
       print('worker ' , a, b, i)
    lock.release()

threads = []
for i in range(5):
    t = threading.Thread(target=worker, args=(i,i+1,))
    threads.append(t)
    t.start()

for i in range(5):
        print('Esperando por ', i)
        t = threads[i]
        t.join()
```

Ejemplo con una clase contador

```
from threading import Thread, Lock
from time import sleep

class Counter:
    def __init__(self):
        self.value = 0
        self.lock = Lock()

    def increase(self, by):
        self.lock.acquire()
        current_value = self.value
        current_value += by
        sleep(0.1)
        self.value = current_value
        print(f'counter={self.value}')
        self.lock.release()
```

Ejemplo con una clase contador

```
counter = Counter()

# create threads
t1 = Thread(target=counter.increase, args=(10, ))
t2 = Thread(target=counter.increase, args=(20, ))

# start the threads
t1.start()
t2.start()

# wait for the threads to complete
t1.join()
t2.join()

print(f'The final counter is {counter.value}')
```

Objetos de tipo condition

- El objeto lleva asociado un lock
- *class* threading.Condition(`lock=None`)
  - Crea una condition y le asocia un lock. Si no se proporciona lock se crea uno
- Métodos:
  - acquire(): método acquire sobre el lock asociado
  - release(): método reléase sobre el lock asociado
  - wait(): libera el lock y bloquea al thread
  - notify(): despierta a un thread bloqueado en la condition
  - notify_all(): despierta a todos threadas bloqueados en la codition

Productor-consumidor en Python

```
from threading import Thread, Condition

class Productor_Consumidor:
    def __init__(self):
        self.cond = Condition()
        self.items = []


    def get(self):
        while True:
                self.cond.acquire()
                while (len(self.items) == 0):
                        self.cond.wait()
                        print('Get esperando')

                x = self.items.pop(0)

                self.cond.notify()
                self.cond.release()

                print("get " + str(x))
```

Productor-consumidor en Python

```
def put(self):
        dato = 0
        while True:
                self.cond.acquire()
                while (len(self.items) == 10):
                        self.cond.wait()
                        print('Put esperando')

                self.items.append(dato)

                self.cond.notify()
                self.cond.release()
                dato = dato + 1
```

Productor-consumidor en Python

```
prod_cond = Productor_Consumidor()

# create threads
t1 = Thread(target=prod_cond.put)
t2 = Thread(target=prod_cond.get)

# start the threads
t1.start()
t2.start()

# wait for the threads to complete
t1.join()
t2.join()
```

Semáforos en Python

- **`semaforo = threading.Semaphore(N)`**
  - Crea un semáforo con valor inicial N
- **`semaforo.acquire()`**
  - operación wait sobre un semáforo
- **`semaforo.release()`**
  - operación signal sobre un semáforo