

Tarea 1

Ruiz Puga Ingrid Pamela

pamela_ruiz@ciencias.unam.mx

Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas,
Cómputo concurrente

Abstract.

Se realiza una investigación sobre las funciones y métodos utilizados en Python y C para la creación de procesos e hilos.

Key words: Procesos; hilos; Python.

1. Introducción

El cómputo concurrente es el programa en el cual múltiples tareas pueden estar en proceso en cualquier instante, ejecuta las tareas de manera espaciada.

Proceso: Un proceso está compuesto por las instrucciones del programa, el estado de ejecución, memoria del trabajo y descriptores de recurso.

Hilo: Son las ejecuciones de peso ligero que facilitan al sistema operativo el intercambio entre ellas. Un hilo a diferencia del proceso no le demanda tanta información y recursos al sistema operativo.

2. Funciones en C para crear procesos e hilos

- **fork():** Hace una llamada al sistema para crear un proceso hijo, idéntico al proceso padre con distinto PID. Haciendo a padre e hijo independientes, pues tienen distintas memorias.
- **exit():** Finaliza el proceso de manera controlada.
- **wait():** Sincroniza a los dos procesos, el padre y el hijo para su ejecución. Hace que algún proceso espere al otro.
- **sleep():** Espera al proceso el tiempo indicado en el sleep.
- **getpid():** Es una llamada para el control de procesos que retorna el pid del proceso hijo que la invoca.
- **getppid():** Es una llamada para el control de procesos que retorna el pid del proceso padre que la invoca.
- **getuid():** Regresa el identificador del usuario real del proceso actual.
- **pid_t:** Es un entero largo con el ID del proceso llamante (getpid)
- **pthread create:** Crea un hilo.
- **pthread join:** Función con la que espera la finalización de un hilo.
- **pthread exit:** Termina ejecución de hilo.

3. Conceptos

3.1 Global Interpreter Lock (GIL)

Python Global Interpreter Lock o GIL, en palabras simples un bloqueo que permite que solo un hilo mantenga el control del intérprete de Python.

Esto significa que solo un subproceso puede estar en estado de ejecución en cualquier momento. El impacto de GIL no es visible para los desarrolladores que ejecutan programas de un solo subproceso, pero puede ser un cuello de botella en el rendimiento en el código de CPU y multiproceso.

El GIL es un bloqueo a nivel de intérprete. Este bloqueo previene la ejecución de múltiples hilos a la vez en un mismo intérprete de Python. Cada hilo debe esperar a que el GIL sea liberado por otro hilo.

3.2 Ley de Amdahl

La Ley de Amdahl se llama así por Eugene Amdahl, el arquitecto informático que formuló dicha ley. En la informática moderna y más concretamente para el desarrollo, se utiliza para averiguar la mejora de un sistema cuando solo una parte de éste es mejorado.

La definición de esta ley establece que: La mejora obtenida en el rendimiento de un sistema debido a la alternación de uno de sus componentes está limitada por la fracción de tiempo que se utiliza dicho componente.

Básicamente la utilizan los ingenieros de sistemas, tanto de hardware como de software, simplemente para definir si introducir una mejora en su sistema merece o no la pena.

$$A = \frac{1}{(1 - F_m) + \frac{F_m}{A_m}}$$

donde

F_m es la fracción de tiempo que el sistema utiliza el subsistema mejorado.

A_m es el factor de mejora que se ha introducido en el sistema.

T_a es el tiempo de ejecución antiguo.

T_m es el tiempo de ejecución mejorado.

3.3 Multiprocessing

El módulo multiprocessing incluye una interfaz de programación para dividir el trabajo entre múltiples procesos basados en la interfaz de programación para threading. En algunos casos multiprocessing es un reemplazo directo, y puede ser usado en lugar de threading para aprovechar los múltiples núcleos de CPU para evitar cuellos de botella computacionales asociados con Python bloqueo global de intérprete.

3.3.1 Métodos de multiprocessing

- **run():** Invoca el objeto invocable pasado al constructor del objeto como argumento objetivo, si lo hay, con argumentos posicionales y de palabras clave tomados de los argumentos *args* y *kwargs*.

- **start():** Esto debe llamarse como máximo una vez por objeto de proceso. Organiza la invocación del método `run()` del objeto en un proceso separado.
- **join([timeout]):** Si el argumento opcional `timeout` es `None` (el valor predeterminado), el método se bloquea hasta que el proceso cuyo método `join()` se llama termina. Si `timeout` es un número positivo, bloquea como máximo `timeout` segundos.
- **terminate():** Terminar el proceso. En Unix, esto se hace usando la señal `SIGTERM`; en Windows se utiliza `TerminateProcess()`. Tenga en cuenta que los manejadores de salida y finalmente las cláusulas, etc., no se ejecutarán.
- **close():** El cierre del objeto `Process`, libera todos los recursos asociados a él. `ValueError` se lanza si el proceso subyacente aún se está ejecutando. Una vez `close()` se retorna con éxito, la mayoría de los otros métodos y atributos del objeto `Process` lanzará `ValueError`.

4. Procesos multiprocessing.process

4.1 Creación procesos como un objeto que hereda de la clase multiprocessing.process

Multiprocesamiento funciona creando un objeto del `process` y luego llamando a su método `start()`. Entonces:

1. Primero importamos la clase de proceso.
2. Instanciar el objeto del proceso con la función de saludo que queremos ejecutar.
3. Contamos el proceso para comenzar a usar el método `start()`.
4. Y finalmente terminamos el proceso con el método `join()`.

4.2 Ejemplo multiprocessing.process

Algunos ejemplos para `multiprocessing` son:

4.2.1 Ejemplo *Hola mundo*

```
1 from multiprocessing import Process
2 class ConcurrentProcess(Process):
3
4     def say_hello(self, name):
5         print("Hello, %s!" % name)
6
7     def run(self):
8         self.say_hello("world")
9
10 if __name__ == "__main__":
11     p = ConcurrentProcess()
12     p.start()
13     p.join()
```

Listing 1. Python example 1

4.2.2 Ejemplo Raiz cuadrada

```
1 from multiprocessing import Process
2
3 def square(x):
4
5     for x in numbers:
6         print('%s squared is %s' % (x, x**2))
7
8 if __name__ == '__main__':
9     numbers = [43, 50, 5, 98, 34, 35]
10
11     p = Process(target=square, args=('x',))
12     p.start()
13     p.join()
14     print "Done"
```

Listing 2. Python example 2