Veleros Vega Luis Alfonso CC-Tarea1

luis veleros

29 de Octubre 2020

1 Funciones de C (hilos)

fork:es una función para crear un proceso , cuando llamamos a un **fork** se duplican los procesos y empiezan a correr por separado , cuando se retorna la función devuelve un identificador del proceso recién creado (hijo)

pthread_t: inicializa un hilo de forma que luego podramos referenciar al hilo para hacerle cosas

pthread_exit(): el hilo terminará

pthread_create(): creación de nuevo hilo con thread con los 'parámetros' posibles

pthread_join:Espera un hilo -¿ como primer parámetro espera al identificador del hilo que va a esperar , el segundo puede ser un variable donde queremos que se deje el hilo cuando termine

getpid(): regresa el id del hijo

getppid(): regresa el id del padre

2 Conceptos

Global Interpreter Lock (GIL):

El Python Global Interpreter Lock o GIL es 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.

Ley de Amdahal: La ley de Amdahal establece 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.esta ley lo que nos dice es que la mejora de rendimiento de un sistema (entendiéndose por sistema un conjunto de piezas como puede ser un PC) cuando cambias una única pieza, está limitada por el tiempo que se utilice dicho componente.

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

Donde

A es la aceleración o ganancia en velocidad conseguida en el sistema completo debido a la mejora de uno de sus subsistemas

A_m es el factor de mejora que se a introducido en el subsistema mejorado

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

Multiprocessing

multiprocessing es un paquete que admite procesos de generación mediante una API similar al módulo threading . El paquete multiprocessing ofrece simultaneidad local y remota, evitando efectivamente el bloqueo de intérprete global mediante el uso de subprocesos en lugar de subprocesos. Debido a esto, el módulo multiprocessing permite al programador aprovechar al máximo múltiples procesadores en una máquina determinada. Funciona tanto en Unix como en Windows.

El módulo multiprocessing también presenta API que no tienen análogos en el módulo threading . Un buen ejemplo de esto es el objeto Pool que ofrece un medio conveniente de paralelizar la ejecución de una función a través de múltiples valores de entrada, distribuyendo los datos de entrada entre los procesos (paralelismo de datos). El siguiente ejemplo demuestra la práctica común de definir tales funciones en un módulo para que los procesos secundarios puedan importar ese módulo con éxito

Spawn:el proceso paretal inicia un nuevo proceso de intérprete de python , el proceso hijo heredará los recursos necesarios para ejecutar los objetos e run()

pipe: retorna un par de objetos de conexión conectados por una tubería (pipe)

run: método que representa la actividad del proceso

join: el método se bloquea hasta que el proceso cuyo método join() se llama termina , si el argumento no es none se bloquea por el valor del argumento en seg

start: comienza la actividad del proceso

is_alive():retorna si el proceso está vivo

3 multiprocessing.process

```
from multiprocessing import Process

def f(name):
```

```
    print('hello', name)

if __name__ == '__main__':
    p = Process(target=f, args=('bob',))
    p.start()
    p.join()
```

Para mostrar las IDs individuales involucradas en el proceso

```
from multiprocessing import Process
import os

def info(title):
    print(title)
    print('module name:', __name__)
    print('parent process:', os.getppid())
    print('process id:', os.getpid())

def f(name):
    info('function f')
    print('hello', name)

if __name__ == '__main__':
    info('main line')
    p = Process(target=f, args=('bob',))
    p.start()
    p.join()
```