CC-Tarea1

Joel Sebastian Avalos Gonzalez

29 de octubre de 2020

0.1. Funciones que fueron utilizadas en C para crear, comunicar y manipular procesos e hilos en C

1. **fork()** crea un nuevo proceso
2. **wait()** deja dormido al proceso que la llama hasta que alguno de sus procesos hijo termina.
3. **pthread_create()** nos permite crear un un nuevo hilo de ejcución
4. **sleep()** llamadas que dejen dormido al thread en espera de algo.
5. **yield()** Los demás threads quedan parados hasta que el primero çeda el control".
6. **getppid()** devuelve el identificador de proceso del proceso actual.
7. **getuid()** devuelve el identificador de usuario real del proceso actual.
8. **pthread_t()** . Compara dos identificados de hilos tid1 y tid2.
9. **pthread_join()** la función suspende la ejecución del hilo de llamada hasta que el hilo de destino finaliza, a menos que el hilo de destino ya haya terminado
10. **pthread_exit()** finaliza el hilo de llamada y devuelve un valor a través de retval que (si el hilo es unible)
11. **pipe()** Se utiliza para pasar información de un proceso a otro.

0.2. Investiga de forma indiviudallos siguientes conceptos:

0.2.1. Global Interpreter Lock (GIL)

Python Global Interpreter Lock o GIL, en palabras simples, es un mutex (o 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 vinculado a la CPU y de varios subprocesos.

Dado que GIL permite que solo se ejecute un subproceso a la vez, incluso en una arquitectura de subprocesos múltiples con más de un núcleo de CPU, GIL se ha ganado la reputación de ser una característica ïnfame"de Python.

Cuando observa un programa típico de Python, o cualquier programa de computadora para el caso, hay una diferencia entre los que están vinculados a la CPU en su rendimiento y los que están vinculados a la E/S.

Los programas vinculados a la CPU son aquellos que están llevando la CPU al límite. Esto incluye programas que realizan cálculos matemáticos como multiplicaciones de matrices, búsqueda, procesamiento de imágenes, etc.

0.2.2. Ley de Amdahal

La mejora en el rendimiento de un sistema debido a la alteración de uno de sus componentes está limitada por la cantidad de tiempo que se utiliza dicho componente.

Dicho de otro modo: si ampliamos la memoria RAM de nuestro equipo, pero nuestro programa está leyendo mucho de disco duro, la mejora que repercute sobre el programa será mínima.

La ley de Amdahl no solo es un enunciado teórico. La mejora implicada de un componente se puede cuantificar mediante la siguiente ecuación:

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

En la ecuación anterior, A es la aceleración o ganancia de un sistema con la mejora, también llamado speed-up; F_m es la fracción de tiempo que se emplea la mejora introducida; y A_m es el factor de mejora.

Esto solo se tiene en cuenta cuando se trata de mejorar sistemas avanzados que se dedican a cálculos científicos, grandes centros de computación, servidores, y éste tipo de sistemas informáticos industriales o profesionales. ¿Cómo podemos utilizar la ley de Amdahl a nuestro favor? Pues muy simple. Cuando vayamos a comprar un nuevo equipo, por ejemplo, podemos tener en cuenta si nos va a salir rentable gastarnos más dinero en una memoria de 1866MHz respecto a una de 1333Mhz, que ya os aventuro que no. Así podremos ahorrar de unos componentes que no vamos a sacar provecho y mejorar otros que sí.

0.2.3. multiprocessing

multiprocesamiento es un paquete que admite procesos de generación utilizando una API similar al módulo de subprocesamiento. El paquete de multiprocesamiento 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 de multiprocesamiento permite al programador

aprovechar al máximo múltiples procesadores en una máquina determinada. Funciona tanto en Unix como en Windows.

El módulo de multiprocesamiento también presenta API que no tienen análogos en el módulo de subprocesamiento. Un buen ejemplo de esto es el objeto Pool, que ofrece un medio conveniente para 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. Este ejemplo básico de paralelismo de datos usando Pool,

1. **run()** Método que representa la actividad del proceso. Puede anular este método en una subclase. El método estándar run () invoca el objeto invocable pasado al constructor del objeto como el argumento de destino, si lo hay, con argumentos secuenciales y de palabras clave tomados de los argumentos args y kwargs, respectivamente.

2. **start()** Inicia la actividad del proceso.Esto se debe llamar como máximo una vez por objeto de proceso. Organiza que el método run () del objeto sea invocado en un proceso separado.

3. **join([timeout])** Si el tiempo de espera del argumento opcional es Ninguno (el predeterminado), el método se bloquea hasta que finaliza el proceso cuyo método join () se llama. Si el tiempo de espera es un número positivo, se bloquea como máximo en segundos. Tenga en cuenta que el método devuelve None si su proceso termina o si el método se agota.

4. **terminate()** En Unix esto se hace usando la señal SIGTERM; en Windows Se utiliza TerminateProcess (). Tenga en cuenta que los manejadores de salida y las cláusulas finalmente, etc., no se ejecutarán. Tenga en cuenta que los procesos descendientes del proceso no se terminarán, simplemente quedarán huérfanos

5. **close()** El objeto Proceso, liberando todos los recursos asociados con él. ValueError se genera si el proceso subyacente aún se está ejecutando. Una vez que close () regresa con éxito, la mayoría de los otros métodos y atributos del objeto Process generarán ValueError.

0.3. Manera de crear procesos como un objeto que hereda de la clase multiprocessing.processy muestra un ejemplo

El objeto Proceso representa una actividad que se ejecuta en un proceso independiente. La clase multiprocessing.Process tiene equivalentes de todos los métodos de threading.Thread. El constructor del proceso siempre debe llamarse con argumentos de palabras clave.

El argumento de destino del constructor es el objeto invocable que será invocado por el método de ejecución. El nombre es el nombre del proceso. El método de inicio inicia la actividad del proceso. El método de unión se bloquea hasta que finaliza el proceso cuyo método de unión se llama. Si se proporciona la opción de tiempo de espera, bloquea como máximo los segundos de tiempo de espera. El método is_alive devuelve un valor booleano que indica si el proceso está activo. El método terminate termina el proceso.

El método de unión bloquea la ejecución del proceso principal hasta que finaliza el proceso cuyo método de unión se llama. Sin el método de unión, el proceso principal no esperará hasta que finalice el proceso

```
!/usr/bin/python
from multiprocessing import Process import time
def fun():
print('starting fun')
time.sleep(2)
print('finishing fun')
def main():
p = Process(target=fun)
p.start()
p.join()
if _name_ ==' _main_' :
print('starting main')
main()
print('finishing main')
```

Referencias

[1] *What Is the Python Global Interpreter Lock (GIL)?.*

[2] *La ley de Amdahl.*

[3] *multiprocessing.*