Multiprocessing

patobarrero

October 2020

1 Primera Parte: Funciones que fueron utilizadas en C para crear, comunicar y manipular procesos e hilos en C

pid_t fork()
Crea un nuevo proceso mediante la duplicación del proceso que llamó a esta función. El nuevo proceso es llamado proceso hijo, mientras que el proceso que llamó a esta función es llamado proceso padre. En caso de éxito se regresa el PID del proceso hijo en el proceso padre, y 0 en el proceso hijo. En caso de alguna falla se regresa 0

void exit()
Termina el proceso actual (el proceso que llamó a esta función) inmediatamente.

void perror(char* s)
Imprime un mensaje en la salida de error estándar describiendo el último error encontrado durante una llamada a una función de sistema o biblioteca.

pid_t wait(int *wstatus)
Permite suspender la ejecución del proceso que llamó a esta función hasta que alguno de sus hijos termine de ejecutarse. Regresa el PID del hilo hijo que terminó.

pid_t getpid(void)
regresa el PID del proceso que llamó a esta función.

pid_t getppid(void)
regresa el PID del proceso padre que llamó a esta función.

uid_t getuid(void)
regresa el ID del usuario real del proceso que llamó a esta función.

int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg)

Empieza un nuevo hilo en el proceso que llamó a esta función.

int pthread_join(pthread_t thread, void **retval)
Función que espera hasta que thread termine. Sí thread ya terminó, la función regresa inmediatamente.

void pthread_exit(void *retval)
Termina el hilo que llamo a esta función.

2 Segunda Parte: Conceptos

2.1 Global Interpreter Lock (GIL)

El global interpreter lock es un mecanismo en python que es usado para lidiar con procesos. Usualmente python utiliza únicamente solo hilo para ejecutar cada una de las líneas de código, esto implica que únicamente un hilo se ejecuta en un determinado momento.

El tiempo que tarda en python ejecutarse un programa con un solo hilo y con varios hilos es el mismo (similar) debido a GIL. No podemos lograr subprocesos múltiples en Python porque tenemos un bloqueo de intérprete global que restringe los subprocesos y funciona como un solo subproceso.

2.2 Ley de Amdahl

La ley de Amdahl se utiliza para determinar la mejora de un sistema de cómputo cuando solo una parte de este es mejorado. Esta ley establece *La mejora obtenida en el rendimiento de un sistema debido a la alteración de uno de sus componentes está limitada por la fracción de tiempo que se utiliza dicho componente* lo hace mediante la siguiente fórmula:

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

donde:
Tm: Tiempo de ejecución mejorado
Ta: Tiempo de ejecución antiguo
Fm: Fracción de tiempo que el sistema utiliza el subsistema mejorado.
Am: Factor de mejora que se ha introducido al sistema

Esta ley implica que cuando se introduce una mejora a una computadora mejorada previamente, el incremento del rendimiento es menor que si se introduce la mejora sobre el sistema sin mejorar. También nos permite determinar si realizar una mejora en el sistema de cómputo vale o no.

2.3 Multiprocessing

Multiprocessing es un módulo de Python que permite la generación de procesos usando una API similar al módulo threading. Este módulo permite trabajar con con concurrencia tanto de forma local como de forma remota, evita usar el Global Interpreter Lock usando procesos en lugar de hilos, permitiendo al programador aprovechar el máximo de procesador de los cuales posee.

Posee una clase llamada Process que representan la actividad que se ejecuta en un proceso separado. Posee los siguientes métodos importantes:

void run()
Representa la actividad que el proceso está realizando

void start()
Permite empezar la actividad del proceso. Cuando es llamado el método start de un proceso se llama al método run del mismo.

bool is_alive()
Permite determinar si un proceso se encuentra vivo o no. Regresa true si se encuentra vivo, false en caso contrario

void join([timeout])
El proceso que llamo al join de otro proceso se queda esperando hasta que termine su actividad (si el timeout es None). Si el tiempo de espera es un número positivo, se bloquea como máximo timeout segundos.

void terminate()
Termina el proceso.

3 Tercera Parte: Heredar clase multiprocessing.process

Si definimos una clase es posible hacer que esta herede de Process del modulo Multiprocessing con la finalidad de que esta clase posea todos los métodos y atributos de Process de tal forma que la podamos utilizar como un proceso y como la abstracción de la realidad que esta clase representa.

Para realizar esto hacemos que la clase que creamos, en el ejemplo Alumno, que herede de Process. Tenemos que sobre-escribir el método run, que va a representar la actividad que realizara nuestro proceso. Si deseamos que nuestra clase tenga un constructor debemos de llamar al constructor padre, tal y como se muestra en el ejemplo

```
from multiprocessing import Process
from time import sleep
```

```
class Alumno(Process):
    def __init__(self, nombre):
        Process.__init__(self)
        self.nombre = nombre

    def estudia(self, tiempo = 2):
        print("soy el alumno", self.nombre, "y voy a estudiar", tiempo,
            "segundos")
        sleep(tiempo)
        print("yo",self.nombre, "termine de estudiar")

    def run(self):
        self.estudia()


if __name__ == "__main__":
    a1 = Alumno("Bob")
    a2 = Alumno("Alice")
    a1.start()
    a2.start()
    a1.join()
    a2.join()
```

References

https://www.ecured.cu/Ley_de_Amdahl
https://hardzone.es/reportajes/que-es/ley-de-amdahl/
https://docs.python.org/3/library/multiprocessing.html
https://man7.org/linux/man-pages/man2
https://recursospython.com/guias-y-manuales/multiprocessing-tareas-concurrentes-con-procesos/