CCTarea1

Pablo David Castillo del Valle

September 2021

1 Funciones usadas en C

A lo largo de las clases hemos usado distintas funciones de C para controlar los precesos.

1.1 Crear un proceso

`fork()`: la función fork nos crea un nuevo proceso a partir de uno dado duplicando la llamada del proceso, dicho proceso nuevo se conoce como proceso hijo, y el proceso previamente dado se denomina proceso padre.

1.2 ID

`getpid()`: nos regresa el ID de un proceso dado.

`getppid()`: nos regresa el ID del proceso padre de u proceso dado, dicho ID será el ID del proceso padre o en el caso de haber terminado el ID del nuevo proceso padre.

1.3 Manipulacion de procesos

`wait()`: la función wait suspende la ejecución del hilo que está sieno llamdo hasta que uno de los hijos haya terminado

`waitpid()`: la funcion waitpid suspende la ejecución del hilo que esté siendo llamado hasta que el hijo con un ID específico haya terminado, el valor del argumenteo *pid* puede ser < -1 lo cuál significa que esperar a cualquier hijo cuyo ID grupal sea igual al valor absoluto del argumento dado, -1 para esperar a un hijo con cualquier ID, 0 para esperar a cualquier hijo cuyo ID sea igual al del proceso que se esté llamando en el momento de la llmada de la función `waitpid()` o > 0 para esperar al proceso hijo con el ID de del argumento dado.

Notemos que entonces `wait()` es equivalente a `waitpid(-1, &wstatus, 0)`

`sleep()`: hace que el hilo que está siendo llamado se *duerma* es decir que espere durante una cantidad específica de tiempo dada en segundos o hasta que una señal que no sea ignorada llegue

`unsleep()`: Función que suspende la ejecución del hilo que se está llamando hasta que haya transcurrido un número específico de microsegundos, notemos que es distinto a `sleep()` ya que mientras uno espera al hilo actual, el otro lo suspende.

1.4 Comunicación entre procesos

`pipe()`: crea una *tuberia*, es decir un canal unidireccional de datos que puede ser usado para comunicacion entre procesos, dichas tuberias tienen una estructura de cola, de ahí que sea un canal unidireccional.

1.5 Creacion de Hilos

`pthread_create()`: función que comienza un nuevo hilo en el proceso de llamada, el nuevo hilo comienza su ejecución invocando a la función `start_routine()`

Hay demasiada información acerca de la programación multihilo en POSIX y por lo tanto no incluiremos toda, sin emabrgo para más información acerca de hilos en POSIX, referirse a `https://man7.org/linux/man-pages/man7/pthreads.7.html`

2 Investigación de conceptos

2.1 GIL

Debido a la manera en la que Python trabaja con la memoria al crear objtos o estructuras de datos, fue necesaria la implementación dle *GIL* el cuál previene que el sistema ejecute varios hilos a la vez para evitar problemas en como la memoria está siendo manipulada por las llamadas al sistema. Por un lado esto hace que los programas en Python sean naturalmente seguros al multiproceso, pero por otro lado no es posible acceder a la programación multiproceso de manera directa en python, sin embargo es posible en algunos casos específicos el deshacerse del GIL o usar bibliotecas multithread la desventaja radica en que muchas otras bibliotecas desarrolladas en python han sido construidas teniendo el GIL en cuenta por lo que no es posible deshacere de el en su totalidad ya que se perdería mucha funcionalidad.

2.2 Ley de Amdahl

Formulada por el científico de origen noruego Gene Amdahl la fórmula habla acerca de las limitaciones en la mejora del rendimiento de un programa al alteerarse los componentes, dicha mejora está limitada por la fracción de tiempo que utiliza cada componente. La fórmula esta dada por:

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

Donde:

- F_m es la fracción de tiempo que el sistema utiliza el subsistema mejorado
- Am es el actor de mejora que se ha introducido en el subsistema mejorado
- T_a es el tiempo de ejecución antes de la mejora
- T_m es el tiempo de ejecución mejorado

Dicha ley es importante porque nos habla acerca de los límites en que se puede mejorar un proceso a través de técnicas como programación concurrente o paraleismo.

2.3 Multipreocessing

Como mencionamos en la primera sección, al Python estar limitado por el GLP, existen bibliotecas que nos permiten tener acceso al multithreading, en este caso `multiprocessing()` nos ayuda a dicha tarea. Dentro de dicha biblioteca, cada proceso es generado mediante la clase `Processy` luego iniciandolos a través de su método `start()` dicha case tiene varios métodos que funcionand e manera similar a los vistos en C, por ejemplo contamos con `pid`, `pipe`, `sleep()` y más.
Hablaremos brevemente de algunos, para más información consultar el API `https://docs.python.org/3/library/multiprocessing.html`

- `run()`: nos permite la llamadda y ejecución de un objeto de la clase process o que haya heredado dicha clase no recibe argumentos.
- `pid`: Retorna el ID del proceso, similar a como sucedía en C
- `sleep()`: Permite la pausa de un proceso, recibe como argumento un número el cuál es el tiempo que se duerme el proceso.
- `start()`: Inicia el proceso, no recibe argumentos.
- `is_alive`: no recibe argumentos, pero nos regresa un booleano indicando si el proceso se encuentra activo.

3 Programa

Un ejemplo de un código utilizando dicha biblioteca de multiprocesos sería el siguiente código:

```
from multiprocessing import Process

def f(name):
    print('Hola', name)

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

Figure 1: Ejemplo de un programa multiproceso usando el módulo processing

Dicho programa nos imprime `Hola Pablo`

Bibliografía

- https://docs.python.org/3/library/multiprocessing.html
- https://man7.org/linux/man-pages/man7/pthreads.7.html
- https://es.wikipedia.org/wiki/Ley_de_Amdahl
- https://docs.python.org/3/library/multiprocessing.html
- https://wiki.python.org/moin/GlobalInterpreterLock
- https://python.land/python-concurrency/the-python-gil