Tarea 1 - Multiprocesamiento en Python

Padilla Robles Artemio Santiago
29 de octubre del 2020
Computación Concurrente - IIMAS
Universidad Nacional Autónoma de México

1. RESUME DE FORMA INDIVIDUAL LAS FUNCIONES QUE FUERON UTILIZADAS EN C PARA CREAR, COMUNICAR Y MANIPULAR PROCESOS E HILOS EN C.

Para crear procesos e hilos en C se utilizaron las funciones de la librería POSIX (Portable Operating System Interface)[1], en particular para hilos se utilizó `<pthread.h>` [2], para llamadas al sistema `<stdio.h>` [3]:

Estas librerías se implementaron mediante:

```
#include <stdio.h>
```
Para hacer llamadas al sistema. Librería estándar de entradas y salidas (I/O).

```
#include <pthread.h>
```
Para usar hilos.

```
#include <unistd.h>
```
Constantes y tipos simbólicos estándar.

```
#include <sys/types.h>
```
Da acceso a más tipos de datos.

```
#include <sys/wait.h>
```
Para usar `waits`

Las funciones y comandos para manejar procesos e hilos fueron [3]:

- `fork()`
  Crea un proceso hijo .
- `exit()`
  Termina un proceso.
- `perror()`
  Imprime un mensaje de error de sistema.
- `wait()`
  Espera a que un proceso cambie de estado.
- `getpid()`
  Obtén el identificador del proceso.
- `getppid()`
  Obtén el identificador del proceso padre del un proceso dado.
- `getuid()`
  Obtén el identificador de usuario que inició el proceso.
- `pthread_t NombreHilo;`
  Este comando sirve para identificar un hilo como NombreHilo;
- `pthread_create(&NombreHilo, NULL, codigo_del_hilo, &id);`
  Este comando sirve para crear un hilo con identificador NombreHilo, NULL lo creará con sus atributos por defecto, codigo_del_hilo le indica al hilo que se va a crear la función que este hilo va a ejecutar, &hilo (Su dirección de memoria), &id su id.
- `pthread_join()`
  Espera a que termine un hilo especificado.
- `pthread_exit()`
  Termina un hilo.
- `pipe()`
  Crea un *pipe*, que es un canal de comunicación entre procesos.
- `close()`
  Cierra un proceso dado.
- `wait()`
  Espera a que un proceso dado cambie de estado.

2. INVESTIGA DE FORMA INDIVIDUAL LOS SIGUIENTES CONCEPTOS:

1. Global Interpreter Lock (GIL) en el contexto del multiprocesamiento en python. Resume los conceptos más importantes.

La implementación estándar de Python tiene implementado el llamado Global Interpreted Lock o GIL que previene a dos hilos ejecutarse de manera simultanea, aun en sistemas con multiprocesadores.

El GIL previene el poder obtener por completo los beneficios del uso de hilos en Python pero mantiene la estabilidad de Python en todas las plataformas donde se ejecuta. [4]

2. Ley de Amdahal : Concepto, terminología relacionada, fórmula de la ley.

La ley de Amdahal dice que si se aplican P procesadores a una tarea de cómputo que tiene una *fracción serial* f, la mejora total en rendimiento es [5]:

$$\text{Mejora en rendimiento} = \frac{1}{f + \frac{1-f}{P}} \tag{1}$$

Un corolario de esta ley es que se aplica un numero muy grande de procesadores o algún otra mejora en rendimiento a un problema, la mejora neta no puede exceder $\frac{1}{f}$ [5].

Una fracción serial se refiere a una fracción de la carga de trabajo de computo que se realiza en serie y no puede en paralelo o secuencialmente.

3. Multiprocessing: Resume la descripción y uso del este módulo de python, enumera y describe los métodos o funciones más importantes del módulo, elige al menos 5 métodos y enlista los argumentos que reciben.

El modulo Multiprocessing es un paquete que permite la creación de procesos. Multiprocessing ofrece tanto concurrencia local y remota, por lo que puede circunvalar la limitación del GIL [6].

Las funciones y métodos más importantes son:

- `p = Process(target=f, args=(arg1,arg2,argn,))`
  Esta función inicializa un proceso hijo en p, que va a correr la función f, con los argumentos arg1,ar2,argn.

- `p.start()`
  El método start() inicializa el proceso hijo con los parámetros que se definieron con la función Process.

- `p.join()`
  El método join espera a que el proceso termine.

- `p.is_alive()`
  Informa sobre si el proceso esta vivo.

- `p.run()`
  Ejecuta la función del proceso p desde el proceso actual, sin crear un proceso nuevo.

3. INVESTIGA DE FORMA INDIVIDUAL LA MANERA DE CREAR PROCESOS COMO UN OBJETO QUE HEREDA DE LA CLASE MULTIPROCESSING.PROCESS Y MUESTRA UN EJEMPLO.

Con Multiprocessing los procesos se crean mediante la creación de objetos Process. Estos objetos Process luego se pueden manipular con métodos propios de esta clase, como `start()` o `join()`[6].

Un ejemplo donde se puede visualizar como se crean estos objetos y la implementación de algunos de sus métodos se muestra a continuación:

```
  from multiprocessing import Process
def f(name):
  print('Hola Oscar, soy ', name)
if __name__ == '__main__':
  p = Process(target=f, args=('Art',))
  p.start()
  p.join()
```


[1] B. Barney, *POSIX Threads Programming*. Revisado el 12 de octubre del 2020 en: https://computing.llnl.gov/tutorials/pthreads/.

[2] T. O. Group, *<pthread.h>*. Revisado el 12 de octubre del 2020 en: https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/pthread.h.html#tag_13_36.

[3] T. L. manual pages. Consultado a través de la terminal de Ubuntu 20.04 el 12 de octubre del 2020. También se puede consultar en: https://www.kernel.org/doc/man-pages/.

[4] R. Tayal, *Python's GIL — A Hurdle to Multithreaded Program*. Consultado el 12 de octubre del 2020 en: https://medium.com/python-features/pythons-gil-a-hurdle-to-multithreaded-program-d04ad9c1a63.

[5] J. L. Gustafson, *Amdahl's Law*, pp. 53–60. Boston, MA: Springer US, 2011.
[6] Python.org, *Multiprocessing — Process-based parallelism*.