Tarea 1

Hugo Rangel Ramírez

23 de Septiembre del 2021

1 Resumen funciones en C

- **fork()**. Crea un nuevo proceso que represente la duplicación del proceso de la persona que llama. Devuelve dos veces cuando tiene éxito; en el proceso padre y el hijo, devuelve el PID del proceso hijo en el padre y el valor 0 en el hijo.
- **getpid()**. Devuelve el ID del proceso hijo.
- **getppid()** Devuelve el ID del proceso padre.
- **getuid()** Devuelve el ID del usuario del proceso.
- **exit()** Provoca la terminación del proceso.
- **wait()** Suspende la ejecución del proceso actual hasta que sus hijos terminen.
- **pipe(pipefd)** Crea una tubería, esto es, un canal unidireccional de datos que puede ser usado para la comunicación interprocesos. El arreglo *pipefd* se usa para devolver dos descriptores de archivo de los extremos de la tubería: *pipefd[0]* que es para leer y *pipefd[1]* que es para escribir. Los datos escritos son almacenados en el buffer por el kernel hasta que son leídos.
- **read(fd, *buf, count)**. Intenta leer el descriptor del archivo *fd* hasta *count* bytes, en el buffer empezando en *buf*.
- **write(fd, *buf, count)**. Escribe desde *count* bytes del buffer empezando en *buf* en el archivo asociado con el descriptor del archivo *fd*.
- **close()**. Cierra un descriptor de archivo para que ya no se hagan referencias hacia éste y pueda ser reusado.
- **pthread_create(thread,attr, start_routine, arg)** Crea un nuevo hilo en el proceso actual. El nuevo hilo comienza la ejecución invocando a *start_routine* con argumentos *arg*.

2 Conceptos

- *Global Interpreter Lock (GIL).* El Global Interpreter Lock o GIL es un mecanismo que impide a la implementación de C de Python (CPython) la ejecución de bytecode de varios hilos a la vez. Esto significa que sólo un hilo puede ejecutarse en cualquier momento determinado, incluso en una arquitectura multihilo con más de un núcleo en el CPU. En consecuencia, con GIL sólo se obtiene concurrencia y no paralelismo.

  Python usa el conteo de referencias para el manejo de la memoria, esto es, los objetos creados en Python tienen una variable que cuenta el número de referencias que apuntan al objeto. Cuando el contador llega a cero se libera la memoria ocupada por el objeto. Un problema que se deriva de lo anterior es que probablemente dos hilos aumenten o disminuyen su valor de manera simultánea, provocando que nunca se libere la memoria de un objeto que ya no existe, o que se libere cuando el objeto sigue teniendo alguna referencia, causando así errores

en el programa. Dicha situación es la principal razón por la cual se requiere un sistema de protección, en Python se decidió confrontar el problema a través del GIL por su fácil implementación y porque se requería una manera segura de importar extensiones para las bibliotecas de C.

- *Ley de Amdahal* La ley de Amdahal formulada por Gene Amdahal es una fórmula que establece que: "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." Tiene dos formulaciones. La original:

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

En dónde:

  - F_m = Fracción del tiempo que el sistema utiliza el subsitema mejorado.
  - A_m = Factor de mejora que se ha introducido en el subsistema mejorado.
  - T_a = Tiempo de ejecución antiguo.
  - T_m = Tiempo de ejecución mejorado.

Y la modificada, usando la definición de incremento de la velocidad que está dado por $A = \frac{T_a}{T_m}$:

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

En la cual:

  - A = Aceleración o ganancia en velocidad conseguida en el sistema completo debido a la mejora de uno de sus subsistemas.
  - A_m = Factor de mejora que se ha introducido en el subsistema mejorado.
  - F_m = Fracción de tiempo que el sistema utiliza el subsistema mejorado.

Esta ley es usada en el cómputo en paralelo para predecir la aceleración cuando se usan múltiples procesadores.

- `multiprocessing`. Es un paquete que permite crear procesos. Ofrece concurrencia tanto local como remota, esquivando el Global Interpreter Lock mediante el uso de subprocesos en lugar de hilos. Debido a esto el módulo Pythonmultiprocessing permite aprovechar al máximo múltiples procesadores en una máquina determinada.

  La clase principal de este paquete es *Process* y sus métodos más importantes son:

  - **Process**(group=None, target=None, name=None, args=(), kwargs={}, *, daemon=None). Es el constructor de la clase con argumentos:
    - *group*. Siempre debe ser `None`; existe únicamente por compatibilidad con `threading.Thread`.
    - *target*. Es el objeto invocable a ser llamado por el método `run()`.
    - *name*. Es el nombre del subproceso.
    - *args*. Es la tupla de argumento para la invocación de destino.
    - *kwargs*. Es un diccionario de argumentos de palabras clave para la invocación de destino.
    - *daemon*. Establece el proceso `daemon` en True o False
  - **run()**. Método que representa la actividad del proceso. Invoca el objeto invocable pasado al constructor del objeto como argumento objetivo, si lo hay, con argumentos posicionares y de palabras clave tomados de los argumentos args y kwargs, respectivamente.
  - **start()**. Comienza la actividad del proceso. Debe llamarse como máximo una vez por objeto de proceso.

- **join**([timeout]). Si el argumento opcional timeout es None (el valor predeterminado), el método se bloquea hasta que el proceso cuyo método join() se llama termina. Si timeout es un número positivo, bloquea como máximo `timeout` segundos. Un proceso puede unirse muchas veces.
- **is_alive()**. Retorna si el proceso está vivo. Aproximadamente, un objeto de proceso está vivo desde el momento en que el método `start()` retorna hasta que finaliza el proceso hijo.
- **terminate()**. Termina el proceso. Si este método se usa cuando el proceso asociado está usando una tubería (pipe) o una cola (queue), entonces la tubería o la cola pueden corromperse y pueden quedar inutilizables por otro proceso.
- **close()**. El cierre del objeto *Process*, libera todos los recursos asociados a él. ValueError se lanza si el proceso subyacente aún se está ejecutando.

3 Creación de procesos

En el siguiente bloque de código se mostrará un ejemplo que indica de manera general cómo se crea un proceso como un objeto que hereda de la clase `multiprocessing.Process`.

```
1  from multiprocessing import Process
2
3  class Proceso(multiprocessing.Process):
4    '''
5    Clase que hereda de Process.
6    '''
7
8    def __init__(self, nombre):
9      '''
10     __init__ : Constructor de la clase.
11
12     Parámetros
13     ----------
14     nombre : Cadena
15       Nombre del Proceso.
16     '''
17     Process.__init__(self)
18     self.name = nombre
19
20   def tarea(self, n):
21     '''
22     tarea : Función que incluye una tarea o una parte de la tarea que
23     hará el proceso, en este caso imprime un mensaje y el cuadrado del
24     número que se le pasa por argumento.
25
26     Parámetros
27     ----------
28     n : Entero
29       Entero que se calculará su cuadrado.
30     '''
31     print("El proceso está realizando la tarea, cuadrado: ", n*n)
32
33   def run(self):
34     '''
35     run : Función que se mandará a llamar cuando se inicie el proceso.
36     Por fuerza debe tener este nombre.
```

```
37      '''
38      self.tarea(2)
```

La clase puede ser probada de la siguiente forma:

```
1  if __name__ == "__main__":
2    proceso1 = Proceso("Proceso1")
3    proceso1.start()
4    proceso1.join()
```

Generando el siguiente resultado:

```
El proceso está realizando la tarea, cuadrado:  4
```

Referencias

[1] Abhinav Ajitsaria. *What Is the Python Global Interpreter Lock (GIL)?* URL: https://realpython.com/python-gil/. (accessed: 23.09.2021).

[2] *Amdahl's law.* URL: https://en.wikipedia.org/wiki/Amdahl%5C%27s_law. (accessed: 23.09.2021).

[3] Lev E. Givon. *Demo of how to extend multiprocessing.* URL: https://gist.github.com/lebedov/2360424. (accessed: 23.09.2021).

[4] *Ley de Amdahl.* URL: https://es.wikipedia.org/wiki/Ley_de_Amdahl. (accessed: 23.09.2021).

[5] *Multiprocesamiento en Python: Global Interpreter Lock (GIL).* URL: https://www.genbeta.com/desarrollo/multiprocesamiento-en-python-global-interpreter-lock-gil. (accessed: 23.09.2021).

[6] *multiprocessing — Paralelismo basado en procesos.* URL: https://docs.python.org/es/3/library/multiprocessing.html. (accessed: 23.09.2021).