Universidad Nacional Autónoma de México

Instituto de Investigaciones en Matemáticas Aplicadas y Sistemas

Tarea 1

Computación Concurrente

Autor:
Vázquez Rojas José David

Octubre 2020

Índice

1. Funciones en C para procesos e hilos

1.1. Funciones para procesos

- fork()
  Se utiliza esta función principalmente para crear un proceso padre y proceso hijo de forma simultánea o concurrente. La función regresá un -1 si la creación del proceso es fallido o un valor entero 0 para indicar que se creó el proceso hijo con otro valor positivo que corresponderá al ID del proceso hijo. padre

- pipe()
  La función pipe, se utiliza como un recurso o canal de comunicación entre procesos de forma unidireccional o direccional. Tradicionalmente usa un apuntador para indicar el espacio de memoria que será el medio de comunicación entre los procesos. Al igual que el fork() regresa un -1 si el proceso fue fallido o un 0 en caso de éxito.

- wait()
  Su principal utilidad para los procesos una vez declarado es esperar a que los procesos hijos terminen su ejecución para que el proceso padré prosiga una vez que los procesos hijos hayan finalizado. Así mismo, regresa el ID del proceso hijo finalizado.

- exit()
  Su utilidad es cerrar o terminar el proceso actual que lo declara. Regresa 0 en caso finalizado con éxito.

1.2. Funciones para hilos

- pthread create()
  Crea un nuevo hilo en el proceso actual. Una vez hecho lo anterior, el hilo inicia su ejecución. La función recibe como parámetro un puntero para el espacio de memoria donde vivirá durante la ejecución.

- pthread join()
  Se utiliza para indicar a un hilo el cual se desea esperar a que termine su ejecución. Regresa un -1 en caso fallido o un 0 en caso de éxito.

- pthread exit()
  La función pthreadexit() termina el hilo de llamada y devuelve un valor a través de retval que (si el hilo se puede unir) está disponible para otro hilo en el mismo proceso que llama pthreadjoin .

2. Investigación de conceptos

2.1. Global Interpreter Lock (GIL)

Es un recurso que protege el acceso a los objetos de Python, evitando que varios subprocesos ejecuten códigos de bytes de Python a la vez. No podemos lograr subprocesos múltiples en Python porque tenemos un bloqueo de interprete global que restringe los subprocesos y funciona como un solo subproceso.Este bloqueo es necesario principalmente porque la gestión de memoria de Python que no es segura para subprocesos.

2.2. Ley de Amdahal

La definición de esta ley establece que: «La mejora obtenida en el rendimiento de un sistema debido a la alternación de uno de sus componentes está limitada por la fracción de tiempo que se utiliza dicho componente». La formula está dad por:

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

donde:

F_m= fraccion de tiempo que el sistema utiliza el subsistema mejorado
A_m = factor de mejora que se ha introducido en el subsistema mejorado.
T_a = tiempo de ejecucion antiguo.
T_m= tiempo de ejecucion mejorado.

Esto con el fin de definir el speedup (aceleración) que se puede alcanzar al usar cierta mejora.

$$speedup = \frac{\text{Rendimiento al usar la mejora}}{\text{Rendimiento sin usar la mejora}}$$

2.3. Multiprocessing

El módulo multiprocessing incluye una interfaz de programación para dividir el trabajo entre múltiples procesos basados en la interfaz de programación para threading. En algunos casos multiprocessing es un reemplazo directo, y puede ser usado en lugar de threading para aprovechar los múltiples núcleos de CPU para evitar cuellos de botella computacionales asociados con Python bloqueo global de intérprete.

- 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.

- start()
  Inicie 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.

- isalive()
  Devuelve si el proceso está vivo.Aproximadamente, un objeto de proceso está vivo desde el momento en que el método start () regresa hasta que termina el proceso hijo.

- join()
  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.

- close()
  Cierre 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.

3. Clase multiprocessing.process y herencia

Un objeto de la clase multiprocessing.process puede heredar los métodos y atributos de una subclase. Unam manera de hacer esto es instanciando una nueva clase cuyo parametro será la clase process. Posteriormente tendremos que hacer un override de los métodos init y run. De manera flexible, podemos añadir otros métodos o atributos a esta clase creada.Un ejemplo es:

```
import multiprocessing
import random
import time
def worker(name: str) -> None:
    print(f'Started worker {name}')
    worker_time = random.choice(range(1, 5))
    time.sleep(worker_time)
    print(f'{name} worker finished in {worker_time} seconds')
if __name__ == '__main__':
    processes = []
    for i in range(5):
        process = multiprocessing.Process(target=worker, args=
        (f'computer_{i}',))
        processes.append(process)
        process.start()
    for proc in processes:
        proc.join()
```

4. Referencias

- https://wiki.python.org/moin/GlobalInterpreterLock
- https://www.programacion.com.py/escritorio/c/creacion-y-duplicacion-de-procesos-en-c-linux
- https://man7.org/linux/man-pages/man3/
- https://dzone.com/articles/python-101-creating-multiple-processes
- https://python-docs-es.readthedocs.io/es/3.8/library/multiprocessing.html

5. Link del archivo Latex

https://www.overleaf.com/3181797393pqnfhttgrdsq