Tarea 1
Computación Concurrente

Alfonso Barajas Cervantes

Ciencia de Datos, IIMAS

Keywords: Concurrencia, Procesos, Hilos, C, Python

Abstract

En este texto se revisarán temas de gran importancia para la computación concurrente. Tales temas serán la de la creación de procesos y de hilos. Asi como de distintas funciones que pueden tener y sus caraterísticas esenciales al ejecutarlos. Estos temas se abordarán desde los lenguajes de programación en C y Python 3.

1 Introducción

La simultaneidad o *concurrencia* significa que se realizan varios cálculos al mismo tiempo. La simultaneidad está en todas partes en la programación moderna, nos guste o no:

- Conjuntos de múltiples computadoras conectadas en la red.
- Conjuntos de múltiples aplicaciones ejecutándose al mismo tiempo en el ordenador.
- Múltiples procesadores en una computadora (hoy, a menudo, múltiples núcleos de procesador en un solo chip)

Ser capaz de programar con simultaneidad seguirá siendo importante en el futuro. Las velocidades de reloj del procesador ya no aumentan. En cambio, obtenemos más núcleos con cada nueva generación de chips. Entonces, en el futuro, para que un cálculo se ejecute más rápido, tendremos que dividir un cálculo en partes simultáneas.

2 Dos modelos para la Concurrencia

En la computación concurrente diferentes procesos o equipos se intercambian y acceden a un mismo recurso durante periodos de tiempo separados, es decir, todos acceden al mismo recurso pero de forma ordenada y nunca juntos. El acceso puede ser tan rápido, que desde el punto de vista del usuario pareciera que múltiples procesos acceden al recurso al mismo tiempo (o al menos esa es la finalidad de la concurrencia en equipos multitarea). Existen dos modelos bastante reconocidos que abordan los modelos que existen en la concurrencia, estos dos son la de la *memoria compartida* y la de la *comunicación* por medio del mensaje.

2.1 Modelo 1. Memoria compartida

En el modelo de concurrencia de memoria compartida, los módulos concurrentes interactúan leyendo y escribiendo objetos compartidos en la memoria. A continuación se mencionarán unos ejemplos sobre estos

- A y B pueden ser dos procesadores (o núcleos de procesador) en la misma computadora, compartiendo la misma memoria física.
- A y B pueden ser dos programas que se ejecutan en la misma computadora, compartiendo un sistema de archivos común con archivos que pueden leer y escribir.
- A y B pueden ser dos subprocesos en el mismo programa Java (explicaremos qué es un subproceso a continuación), compartiendo los mismos objetos Java

2.2 Modelo 2. Comunicación por mensaje

En el modelo deión de mensajes, los módulos simultáneos interactúan enviándose mensajes entre sí a través de un canal de comunicación. Los módulos envían mensajes y los mensajes entrantes a cada módulo se ponen en cola para su manejo. Ejemplos incluyen:

- A y B pueden ser dos computadoras en una red, que se comunican mediante conexiones de red.
- A y B pueden ser un navegador web y un servidor web: A abre una conexión a B, solicita una página web y B envía los datos de la página web a A.
- A y B pueden lanzar mensajes instantáneos al cliente-servidor.
- A y B pueden ser dos programad que se ejecutan en la misma computadora en la que el intput-output que han sido conectadas por medio de un pipe.

3 Concurrencia en C y comandos

3.1 Procesos en C

Lo primero que hay que saber es que todo programa en ejecución es un proceso. Entonces para la creación de los procesos se ocupa el comando *fork()*, su utilidad es la creación del

proceso hijo que es exactamente igual que el proceso original o el proceso padre.

El comando *wait()*, el cual hace como la función de un semáforo, el cual el sistema determina la cantidad de recursos disponibles o la cantidad de procesos que pueden acceder al recurso simultáneamente, si solo hablamos de un único recurso esta variable señala su disponibilidad. Cuando un proceso quiere utilizar el recurso, llamará a la función *wait()*, en este caso bloquea al proceso o lo pone en la cola de espera.

El comando *exit()* sirve cuando un proceso termina su ejecución de manera voluntaria para que el proceso hijo se ejecute.

Por último se vió el comando *pipe()* para la creación de canales de comunicación entre procesos, para que se pueda leer datos del canal que se utilizará y los datos necesarios para escribir.

3.2 Hilos en C

Un hilo es un lugar de control dentro de un programa en ejecución. Piense en ello como un lugar en el programa que se está ejecutando, más la pila de llamadas a métodos que llevaron a ese lugar al que será necesario regresar.

Para la creación de hilos se utiliza el comando *pthread create* gracias a que se colocó antes la libreria necesaria de POSIX Threads en C, el cual es *#include <pthread.h>*

Mientras que para al tener una cierta simultaneaidad de hilos y procesos, se usa el comando *pthread join* se bloquea la ejecuicńo de hilo actual hasta que el processo con id thread termine.

4 Conceptos importantes de Multiprocessing

4.1 Global Interpreter Lock (GIL)

Global Interpreter Lock o GIL, en palabras simples, es un mutex (o un bloqueo) que permite que solo un hilo mantenga el control del intérprete de algún programa como lo puede ser Python o Ruby.

Esto significa que solo un subproceso puede estar en estado de ejecución en cualquier momento. El impacto de GIL no es visible para los desarrolladores que ejecutan programas de un solo subproceso, pero puede ser un cuello de botella en el rendimiento en el código vinculado a la CPU y de varios subprocesos. Sin embargo en Python, dado que GIL permite que solo se ejecute un subproceso a la vez, incluso en una arquitectura de subprocesos múltiples con más de un núcleo de CPU, GIL se ha ganado la reputación de ser una característica ”infame” de Python.

4.2 Ley de Amdahal

La Ley de Amdahl se llama así por Eugene Amdahl, el arquitecto informático que formuló dicha ley. En la informática moderna y más concretamente para el desarrollo, se utiliza para averiguar la mejora de un sistema cuando solo una parte de éste es mejorado. 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

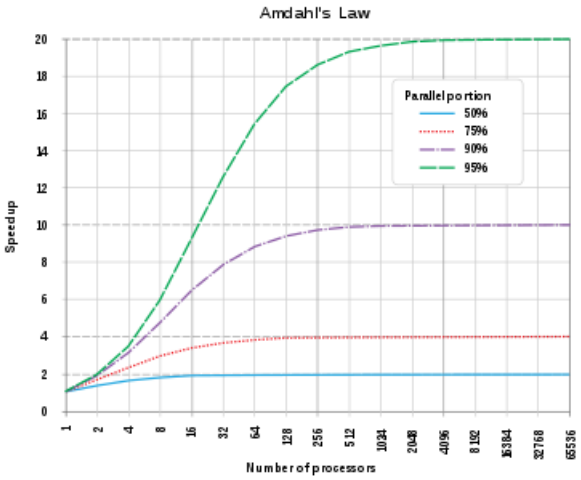

Figure 1. Ley de Amdahal aplicada a la paralelización de un programa

de tiempo que se utiliza dicho componente. La fórmula es la siguiente:

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

Donde tenemos que los términos relacionados son los siguientes:

- F_m es la francción de tiempo que el sistema utiliza el subsistema mejorado.
- A_m es el factor de mejora que se ha introducido en el sistema.
- T_a es el tiempo de ejecución antiguo.
- T_m es el tiempo de ejecución mejorado.

En propias palabras, lo qeu nos quiere decir está ley es que la mejora de rendimiento de un sistema (entendiéndose por sistema un conjunto de piezas como puede ser un PC) cuando cambias una única pieza, está limitada por el tiempo que se utilice dicho componente.

4.3 Librería Multiprocessing, Python

Multiprocessing es un paquete que soporta la creacion de procesos usando una API muy similar a la de creacion de hilos. Esta paqueteria ofrece tanto una concurrencia local como remoto, de manera efectiva y eficaz con el Lock de Global Interpreter que seria el GIL. A continuación mostraremos los meétodos más relevantes:

1. *start():* En multiprocesamiento, los procesos se generan creando un objeto Process y luego llamando a su método start (). El proceso sigue la API de subprocesamiento. El cual no recibe lo que sería atributos, debido a que lo deseamos inicializar.

2. *join():* Para esperar hasta que un proceso haya completado su trabajo y termine, usa el método join(). Por defecto, join() bloquea indefinidamente. También es posible pasar un argumento de tiempo de espera (un númedo de coma flotante que representa el número de segundos para esperar a que el proceso se vuelva inactivo).

3. *terminate():* Llamando a terminate() en un objeto de proceso mata el proceso hijo.

5 Creación de procesos en Python

En python la librería multiprocessing nos permite crear procesos de forma simple. La clase multiprocessing.Process(target, args) genera un nuevo proceso, el cual ejecutará la función target, usando los parámetros args.El método Process.start() inicializa el proceso. De manera similar a C usando el método Process.join() nuestro proceso padre esperará a que el proceso hijo termine su ejecución.

```
from multiprocessing import Process

def f(name):
    print('Hello',name, '. How are you doing today?')

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

```
Hello Alfonso . How are you doing today?
```

Figure 2. Ejemplo de creación de procesos

6 Referencias

1. Reading 17. Concurrency [[]http://web.mit.edu/6.005/www/fa14/classes/17-concurrency/]]
2.Comupaticion Concurrente [[]https://codingornot.com/que-es-la-computacion-concurrente]]
3.Ley de Amdahl. [[https://hardzone.es/reportajes/que-es/ley-de-amdahl/]]