Maestría en Sistemas Computacionales

Tecnologías de Programación

Starvation

D. en C. PATRICIA ELIZABETH FIGUEROA MILLÁN

G2146012 Ricardo Gutiérrez Valenzuela
G2146001 Ángel Primitivo Vejar Cortés
G2146006 Brandon Jesús Santana Gudiño

¿Qué es?

En informática, inanición (**starvation** en inglés) es un problema relacionado con los sistemas multitarea (concurrency programming), donde a un proceso o un hilo de ejecución **se le deniega siempre el acceso a un recurso compartido**. Sin este recurso, la tarea a ejecutar no puede ser nunca finalizada. Anthony D. Joseph. (2004)

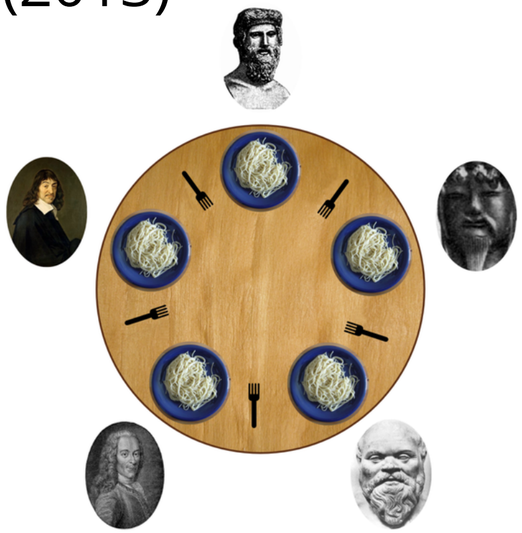
En la programación concurrente, **thread starvation** se refiere a la situación en la que un hilo **monopoliza un recurso** protegido por una **primitiva de concurrencia (lock, rlock, condition, etc.)** e impide que uno o más subprocesos accedan al recurso de manera justa.

¿Cuándo sucede?

Se dice que un programa sufre inanición cuando algún hilo es incapaz de utilizar los recursos que necesita porque otros hilosno los liberan. Aunque el programa sigue haciendo cosas, el hilo afectado no va a ninguna parte. dit(no date)

Caso de ejemplo

Un caso de inanición lo ilustra perfectamente la paradoja conocida como la cena de los filósofos de Edsger Dijkstra cuando se da el caso de que todos los filósofos cogen el tenedor a la vez. rafael (2013)



Deadlock

Un interbloqueo (Deadlock) es una situación en la que los hilos no pueden progresar y no se pueden ejecutar.

Livelock

Un livelock es una situación en la que los hilos no pueden progresar en la aplicación, pero continúan ejecutándose.

Starvation

Starvation sucede cuando **uno** o más hilos **no** pueden progresar, pero **otros** hilos del sistema **sí** pueden progresar y el subproceso hambriento puede o no estar **temporalmente** en starvation.

Brownlee, J. (2022)

Ejemplos de inanición de hilos

La inanición de hilos puede ocurrir por muchas razones, aunque es común cuando una primitiva de concurrencia se adquiere demasiado lejos de la sección crítica. Aquí, "demasiado lejos" se refiere al número de líneas de código o instrucciones ejecutadas antes del punto en el código donde el recurso o la sección crítica requiere protección contra la programación de concurrencia.

La falta de subprocesos puede ocurrir, de acuerdo con Brownlee, J. (2022), con la mayoría de las primitivas de simultaneidad que deben adquirirse, como: Bloqueos de exclusión mutua (mutex), , Semáforos, Condiciones.

Ejemplos de inanición de hilos (mutex)

Por ejemplo, puede haber una sección crítica de código protegida por un bloqueo de exclusión mutua (mutex). Es posible que esta sección crítica deba ejecutarse muchas veces por un subproceso determinado y por varios subprocesos. La falta de subprocesos puede ocurrir si un subproceso continúa adquiriendo el bloqueo mutex y ejecutando la sección crítica, privándose o impidiendo que todos los demás subprocesos adquieran el bloqueo y ejecuten la sección crítica. Brownlee, J. (2022)

Ejemplos de inanición de hilos (Semáforo)

Otro ejemplo podría ser un recurso con acceso limitado protegido por un semáforo. Múltiples hilos pueden necesitar adquirir una posición en el semáforo repetidamente como parte de su tarea. La falta de hilo puede ocurrir si el número de posiciones en el semáforo es limitado y el mismo hilo o hilos continúan adquiriendo esas posiciones, muriendo de hambre o impidiendo que otros hilos adquieran una posición en el semáforo. Brownlee, J. (2022)

Ejemplo de Starvation

```python
def tarea(lock):
    with lock:  # primitiva de concurrencia
        sleep(0.01) # tareas tardadas
        for i in range(ITERATIONS): # trabajo
            sleep(0.01) # más tareas tardadas
            logging.debug(f'Working {i}') # acción crítica
            sleep(0.01) # más tareas tardadas
```

```
2022-10-19 16:43:49,949 (MainThread) Ejemplo de Starvation
2022-10-19 16:43:49,953 (hilo-0    ) Working 0
2022-10-19 16:43:49,958 (hilo-0    ) Working 1
2022-10-19 16:43:49,960 (hilo-0    ) Working 2
2022-10-19 16:43:49,961 (hilo-0    ) Working 3
2022-10-19 16:43:49,964 (hilo-0    ) Working 4
2022-10-19 16:43:49,966 (hilo-0    ) Working 5
2022-10-19 16:43:49,967 (hilo-1    ) Working 0
2022-10-19 16:43:49,971 (hilo-1    ) Working 1
2022-10-19 16:43:49,972 (hilo-1    ) Working 2
2022-10-19 16:43:49,974 (hilo-1    ) Working 3
2022-10-19 16:43:49,975 (hilo-1    ) Working 4
2022-10-19 16:43:49,976 (hilo-1    ) Working 5
2022-10-19 16:43:49,977 (hilo-2    ) Working 0
2022-10-19 16:43:49,981 (hilo-2    ) Working 1
2022-10-19 16:43:49,982 (hilo-2    ) Working 2
2022-10-19 16:43:49,983 (hilo-2    ) Working 3
2022-10-19 16:43:49,984 (hilo-2    ) Working 4
2022-10-19 16:43:49,985 (hilo-2    ) Working 5
```

Solución #1

Mover la primitiva más cerca de la operación crítica

```python
def tarea(lock):
    sleep(0.01) # tareas tardadas
    for i in range(ITERATIONS): # trabajo
        sleep(0.01) # más tareas tardadas
        with lock:  # primitiva de concurrencia
            logging.debug(f'Working {i}') # acción crítica
        sleep(0.01) # más tareas tardadas
```

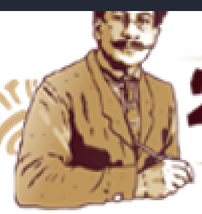
```
2022-10-19 16:58:21,430 (MainThread) Ejemplo de Starvation
2022-10-19 16:58:21,466 (hilo-0    ) Working 0
2022-10-19 16:58:21,469 (hilo-1    ) Working 0
2022-10-19 16:58:21,473 (hilo-2    ) Working 0
2022-10-19 16:58:21,498 (hilo-0    ) Working 1
2022-10-19 16:58:21,514 (hilo-1    ) Working 1
2022-10-19 16:58:21,515 (hilo-2    ) Working 1
2022-10-19 16:58:21,530 (hilo-0    ) Working 2
2022-10-19 16:58:21,546 (hilo-2    ) Working 2
2022-10-19 16:58:21,547 (hilo-1    ) Working 2
2022-10-19 16:58:21,561 (hilo-0    ) Working 3
2022-10-19 16:58:21,576 (hilo-2    ) Working 3
2022-10-19 16:58:21,577 (hilo-1    ) Working 3
2022-10-19 16:58:21,591 (hilo-0    ) Working 4
2022-10-19 16:58:21,606 (hilo-1    ) Working 4
2022-10-19 16:58:21,608 (hilo-2    ) Working 4
```

Solución #2

Utilizar manualmente acquire y release

```python
def tarea(lock):
    sleep(0.01) # tareas tardadas
    for i in range(ITERATIONS): # trabajo
        sleep(0.01) # más tareas tardadas
        lock.acquire()  # primitiva de concurrencia
        logging.debug(f'Working {i}') # acción crítica
        lock.release()  # primitiva de concurrencia
        sleep(0.01) # más tareas tardadas
```

```
2022-10-19 17:06:12,376 (MainThread) Ejemplo de Starvation
2022-10-19 17:06:12,405 (hilo-0    ) Working 0
2022-10-19 17:06:12,408 (hilo-1    ) Working 0
2022-10-19 17:06:12,421 (hilo-2    ) Working 0
2022-10-19 17:06:12,437 (hilo-0    ) Working 1
2022-10-19 17:06:12,453 (hilo-2    ) Working 1
2022-10-19 17:06:12,454 (hilo-1    ) Working 1
2022-10-19 17:06:12,469 (hilo-0    ) Working 2
2022-10-19 17:06:12,485 (hilo-1    ) Working 2
2022-10-19 17:06:12,486 (hilo-2    ) Working 2
2022-10-19 17:06:12,500 (hilo-0    ) Working 3
2022-10-19 17:06:12,515 (hilo-1    ) Working 3
2022-10-19 17:06:12,516 (hilo-2    ) Working 3
2022-10-19 17:06:12,531 (hilo-0    ) Working 4
2022-10-19 17:06:12,547 (hilo-1    ) Working 4
2022-10-19 17:06:12,548 (hilo-2    ) Working 4
```



Solución #3

Cambiar de primitiva

```python
# Crear un semáforo (primitiva de concurrencia)
sem = Semaphore(HILOS)
```

```python
def tarea(sem):
    sleep(0.01) # tareas tardadas
    for i in range(ITERATIONS): # trabajo
        sleep(0.01) # más tareas tardadas
        with sem:  # primitiva de concurrencia
            logging.debug(f'Working {i}') # acción crítica
        sleep(0.01) # más tareas tardadas
```

```
2022-10-19 17:18:00,058 (MainThread) Ejemplo de Starvation
2022-10-19 17:18:00,090 (hilo-0    ) Working 0
2022-10-19 17:18:00,092 (hilo-2    ) Working 0
2022-10-19 17:18:00,092 (hilo-1    ) Working 0
2022-10-19 17:18:00,121 (hilo-0    ) Working 1
2022-10-19 17:18:00,123 (hilo-2    ) Working 1
2022-10-19 17:18:00,137 (hilo-1    ) Working 1
2022-10-19 17:18:00,152 (hilo-2    ) Working 2
2022-10-19 17:18:00,152 (hilo-0    ) Working 2
2022-10-19 17:18:00,168 (hilo-1    ) Working 2
2022-10-19 17:18:00,184 (hilo-2    ) Working 3
2022-10-19 17:18:00,184 (hilo-0    ) Working 3
2022-10-19 17:18:00,200 (hilo-1    ) Working 3
2022-10-19 17:18:00,216 (hilo-2    ) Working 4
2022-10-19 17:18:00,216 (hilo-0    ) Working 4
2022-10-19 17:18:00,233 (hilo-1    ) Working 4
```

Acceso a programas de ejemplo

https://github.com/Primilink/thread-management-python

Conclusión

La programación de concurrencia permite realizar múltiples tareas de forma "simultánea", sin embargo, este puede traer problemas tales como "thread starvation" en donde el rendimiento se ve afectado por el progreso de los hilos debido al "monopolio" de los recursos en uno o pocos hilos, dejando al resto sin acceso. Para este tipo de problemas existen diferentes soluciones y es conveniente conocerlos para poder evitarlos o en su caso poder corregirlos de forma rápida y fácil.

Referencias

Anthony D. Joseph. (2004)CS 162 Operating Systems and Systems Programming

dit (no date) Ayuda. Available at: https://www.dit.upm.es/~pepe/libros/concurrency/index.html#!1024 (Accessed: October 19, 2022).

rafael (2013) Edsger Dijkstra, El Pensador informático, Rafael Rivas Estrada. Available at: http://webdelprofesor.ula.ve/ingenieria/rafael/edsger-dijkstra-el-pensador-informatico/#:~:text=%C2%B7%20Si%20todos%20los%20fil%C3%B3sofos%20cogen,que%20alguno%20deje%20sus%20tenedores). (Accessed: October 19, 2022).

Brownlee, J. (2022) Thread starvation in Python, Super Fast Python. Available at: https://superfastpython.com/thread-starvation-in-python/#What_is_Thread_Starvation (Accessed: October 19, 2022).

Maestría en Sistemas Computacionales

Tecnologías de Programación

Starvation

D. en C. PATRICIA ELIZABETH FIGUEROA MILLÁN

G2146012 Ricardo Gutiérrez Valenzuela
G2146001 Ángel Primitivo Vejar Cortés
G2146006 Brandon Jesús Santana Gudiño