Sincronización de hilos en Python

La sincronización de hilos es fundamental para evitar errores en programas multihilo, como condiciones de carrera o interbloqueos. A continuación, exploramos varios conceptos clave relacionados con la sincronización de hilos en Python, con ejemplos prácticos.

Exclusión mutua: condiciones de carrera y secciones críticas

Una **condición de carrera** ocurre cuando varios hilos acceden y modifican un recurso compartido sin coordinación adecuada, lo que genera resultados impredecibles. Para prevenir esto, se utilizan **secciones críticas**, que son fragmentos de código donde solo un hilo puede acceder a un recurso compartido a la vez.

Ejemplo:

[crayon-6a9d49da5773c198283718/]

Aquí, usamos un **lock** para proteger la sección crítica, asegurando que solo un hilo modifique el contador a la vez, evitando una condición de carrera.

Bloqueo intrínseco: bloques de código sincronizados

En Python, los **bloqueos intrínsecos** (también conocidos como **locks**) permiten a los hilos sincronizarse fácilmente. Cuando un hilo adquiere un lock, ningún otro hilo puede entrar en la sección bloqueada hasta que el primero libere el lock.

Ejemplo:

[crayon-6a9d49da57743955454990/]

En este ejemplo, el lock.acquire() y lock.release() aseguran que el recurso sea modificado por un solo hilo a la vez, previniendo interferencias entre hilos.

Compartición de recursos: interbloqueo

Un **interbloqueo** ocurre cuando dos o más hilos esperan indefinidamente que otro hilo libere un recurso, generando una situación donde ninguno puede continuar. Esto se puede evitar implementando medidas como la evitación de recursos circulares o utilizando **timeouts**.

Ejemplo de interbloqueo:

[crayon-6a9d49da57745734079472/]

En este código, ambos hilos intentan adquirir dos locks en diferente orden, lo que puede causar un interbloqueo si no se maneja adecuadamente.

Compartición de recursos con bloqueo dependiente de su estado

Para evitar interbloqueos, es importante diseñar los bloqueos de tal manera que los hilos no esperen indefinidamente si un recurso no está disponible. Esto se puede hacer utilizando bloqueos con **timeouts**.

Ejemplo con timeout:

[crayon-6a9d49da57747083624360/]

Aquí, el segundo hilo puede fallar en adquirir el lock si el primer hilo lo mantiene por más de 1 segundo, evitando un potencial interbloqueo.

Clases thread-safe

Las clases **thread-safe** son aquellas que permiten el uso seguro en entornos multihilo sin la necesidad de sincronización explícita por parte del usuario. Un ejemplo de esto en Python es la clase Queue, que permite a los hilos agregar y eliminar elementos de manera segura.

Ejemplo:

[crayon-6a9d49da57749791981166/]

Aquí, queue.Queue es una clase **thread-safe** que maneja automáticamente la sincronización de los hilos productores y consumidores.