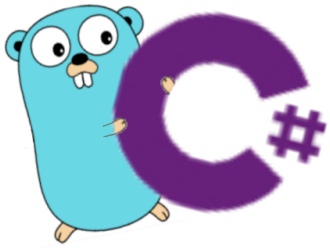
Lenguajes de Programación

Concurrencia



Leandro Rodríquez Llosa
Laura V. Riera Pérez
Marcos M. Tirador del Riego

Tercer año. Ciencias de la Computación.
Facultad de Matemática y Computación, Universidad de La Habana, Cuba

Noviembre 2022

I. Concurrencia y paralelismo

La concurrencia es la ejecución simultánea de varias hebras (una hebra es una ejecución secuencial de instrucciones) pero esta simultaneidad puede ser solo en apariencia. Cada programa se ejecuta de forma secuencial, el sistema operativo es el encargado de administrar y distribuir el tiempo que el procesador le da a cada tarea. La vida de los procesos comparten el mismo tiempo, pero la ejecución de todos ellos no ocurre en el mismo instante.

Se comparte el tiempo de ejecución, dándole a cada proceso una suma de tiempo limitada para ejecutarse. Solo un proceso se ejecuta en un instante dado, y si no completa su operación dentro de su tiempo, el proceso se coloca en pausa, y se le da lugar a otro proceso para iniciar o resumirse, hasta que le vuelva a tocar su tiempo.

Sea T el tiempo total de ejecución de dos programas concurrentes P_1 y P_2:

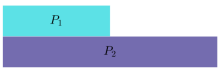
Concurrencia

$$T = T(P_1) + T(P_2)$$

En la computación paralela, la ejecución ocurre en el mismo instante físico, los cálculos se realizan de forma verdaderamente simulánea. Para maximizar el uso de múltiples procesadores o núcleos, presentes en las CPU modernas, el procesamiento en paralelo dividirá el trabajo entre varios subprocesos, cada uno de los cuales puede ejecutarse de forma independiente en un núcleo diferente.

Sea T el tiempo total de ejecución de dos programas paralelos P_1 y P_2:

Paralelismo



$$T = max(T(P_1), T(P_2))$$

Paralelismo implica concurrencia, pero no se cumple el recíproco.

I. Monitors en C#

Proporcionan un mecanismo que sincroniza el acceso a los objetos.

La clase Monitor permite sincronizar el acceso a una región de código tomando y liberando un bloqueo en un objeto determinado llamando a los métodos Monitor.Enter, Monitor.TryEnter y Monitor.Exit . Los bloqueos de objeto proporcionan la capacidad de restringir el acceso a un bloque de código, normalmente denominado sección crítica. Aunque un subproceso posee el bloqueo de un objeto, ningún otro subproceso puede adquirir ese bloqueo. También puede usar la Monitor clase para asegurarse de que ningún otro subproceso pueda acceder a una sección del código de aplicación que ejecuta el propietario del bloqueo, a menos que el otro subproceso ejecute el código mediante un objeto bloqueado diferente.

II. Semáforos en C#

III. Barriers en C#

Permite que múltiples tareas trabajen de manera cooperativa en un algoritmo en paralelo a través de múltiples fases.

En esta sincronización de barrera, tenemos varios subprocesos que trabajan en un solo algoritmo. El algoritmo funciona en fases. Todos los subprocesos deben completar la fase 1 y luego pueden continuar con la fase 2. Hasta que todos los subprocesos no completen la fase 1, todos los subprocesos deben esperar a que todos los subprocesos lleguen a la fase 1.

IV. Countdowns en C#

V. Propuesta en Go para la sincronización en la concurrencia

VI. Solución a los filósofos

vi.1. Go

vi.2. C#

vi.3. Comparación