Programación Concurrente y Distribuida

Problema de Sección Crítica

Ms.Carlos Jara García
pcsicjar@upc.edu.pe

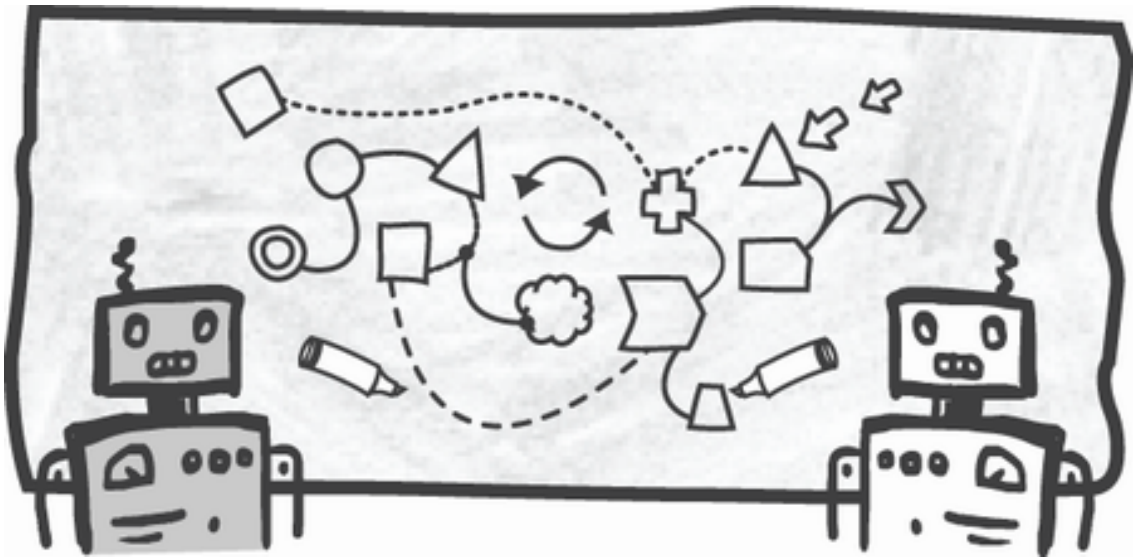
Ciencias de la Computación
Universidad Peruana de Ciencias Aplicadas

2024

Contenidos

Memoria Compartida

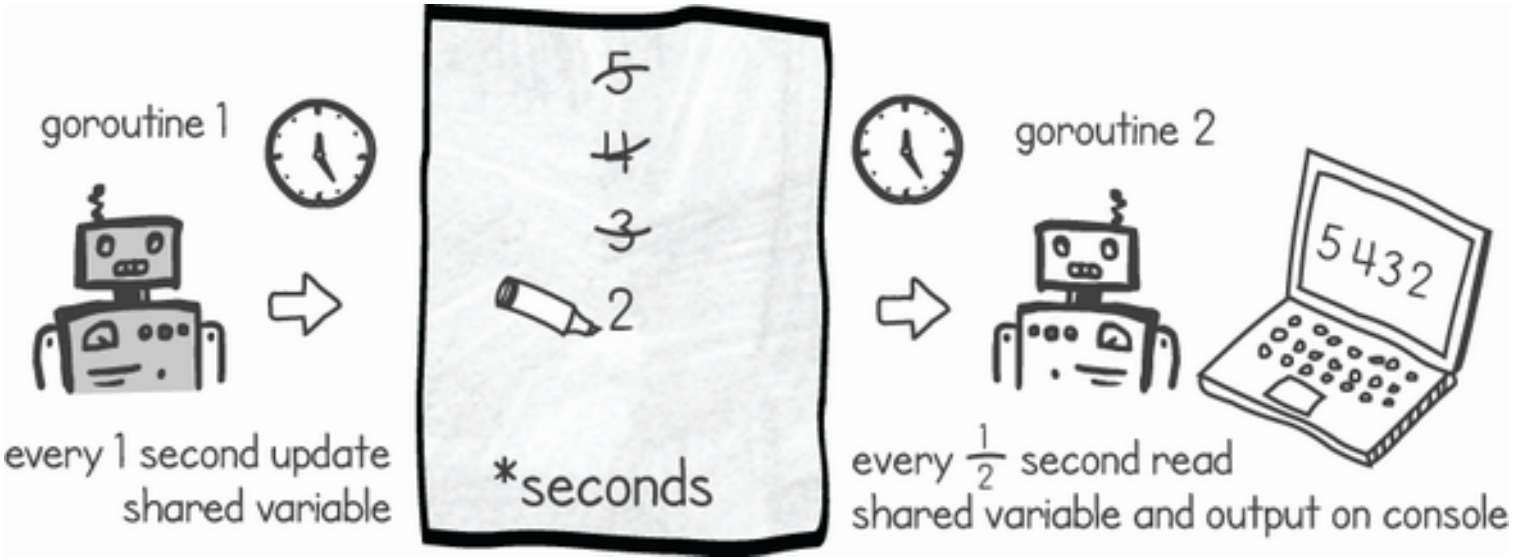
En la programación concurrente que utiliza memoria compartida, asignamos una parte de la memoria del proceso (por ejemplo, una estructura de datos compartida o una variable) y tenemos diferentes gorutinas que trabajan simultáneamente en esta memoria. En nuestra analogía, la pizarra es la memoria compartida utilizada por las distintas goroutines.



Memoria compartida es una manera en que multiples goroutines pueden comunicarse para realizar una tarea.

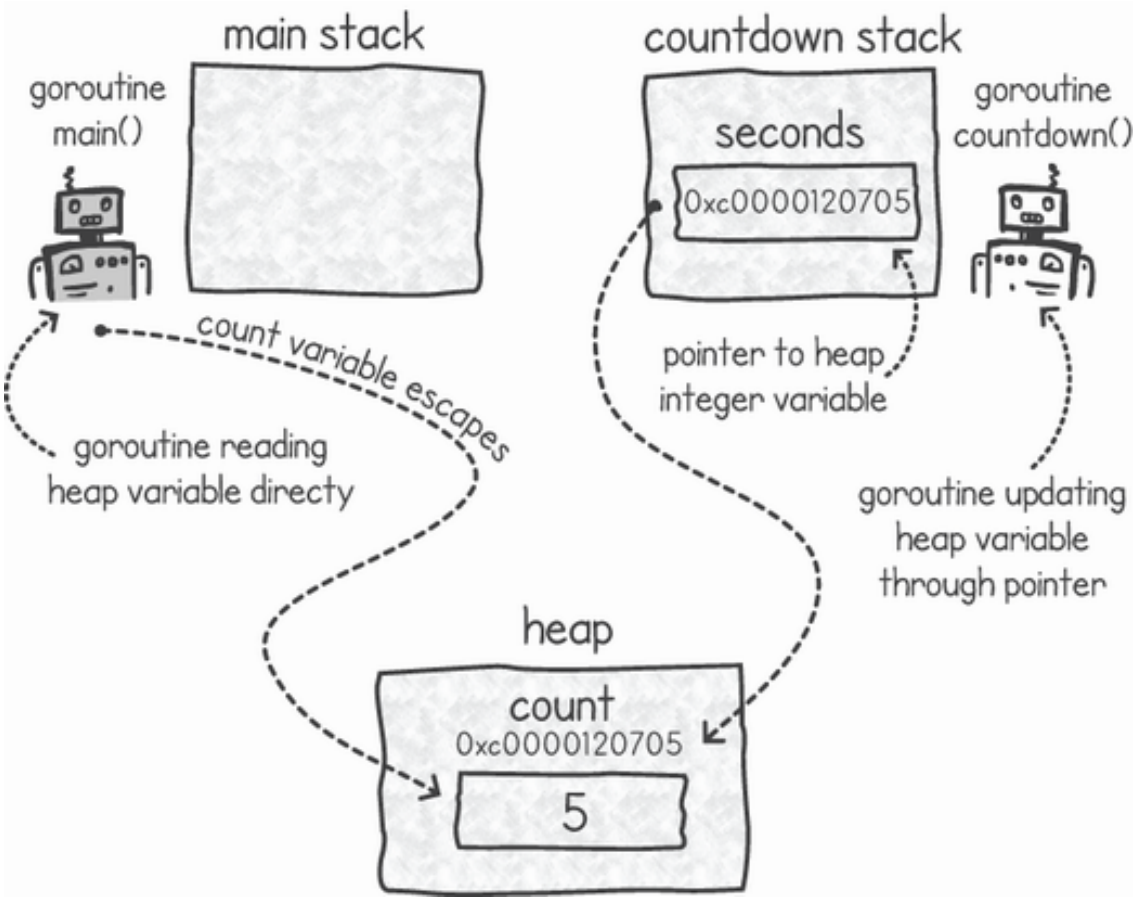
Compartiendo una variable entre goroutines

¿Cómo conseguimos que dos goroutines compartan memoria? En este primer ejemplo, crearemos un goroutine que compartirá una variable en la memoria con la goroutine main() (que ejecuta la función main()). La variable actuará como un temporizador de cuenta regresiva. Un goroutine disminuirá el valor de esta variable cada segundo y otro goroutine leerá la variable con más frecuencia y mostrará el valor en la consola.



Análisis de escape

Cada vez que se comparte una variable fuera del alcance del marco de stack de una función, la variable se asigna al heap. Compartir la referencia de una variable entre goroutines es un ejemplo de ello.



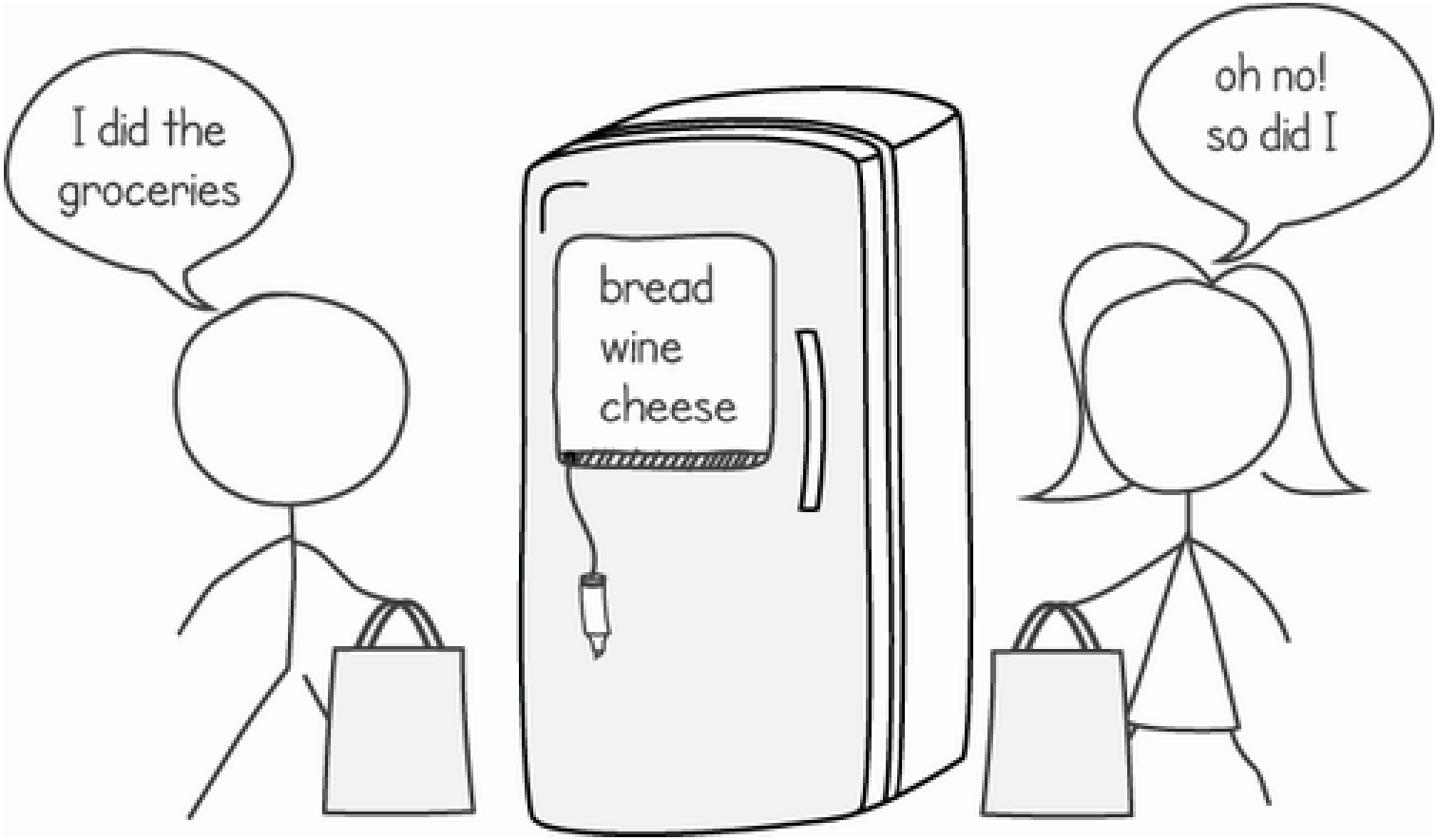
Condición de carrera

Caso: Interrupción en todo el sistema

- El ambiente en la reunión en el piso 24 de Turner Belfort, un enorme banco de inversión internacional, era de lo más sombrío posible. Los desarrolladores de software de la empresa se reunieron para discutir el mejor camino a seguir después de que una aplicación central crítica fallara y provocara una interrupción en todo el sistema. La falla del sistema provocó que las cuentas de los clientes informaran montos erróneos en sus tenencias.
- Finalmente, David Holmes, gerente de entrega, rompió el silencio con esta pregunta: "La aplicación ha estado funcionando durante meses sin ningún problema y no hemos publicado ningún código recientemente, entonces, ¿cómo es posible que el software simplemente se haya estropeado?".

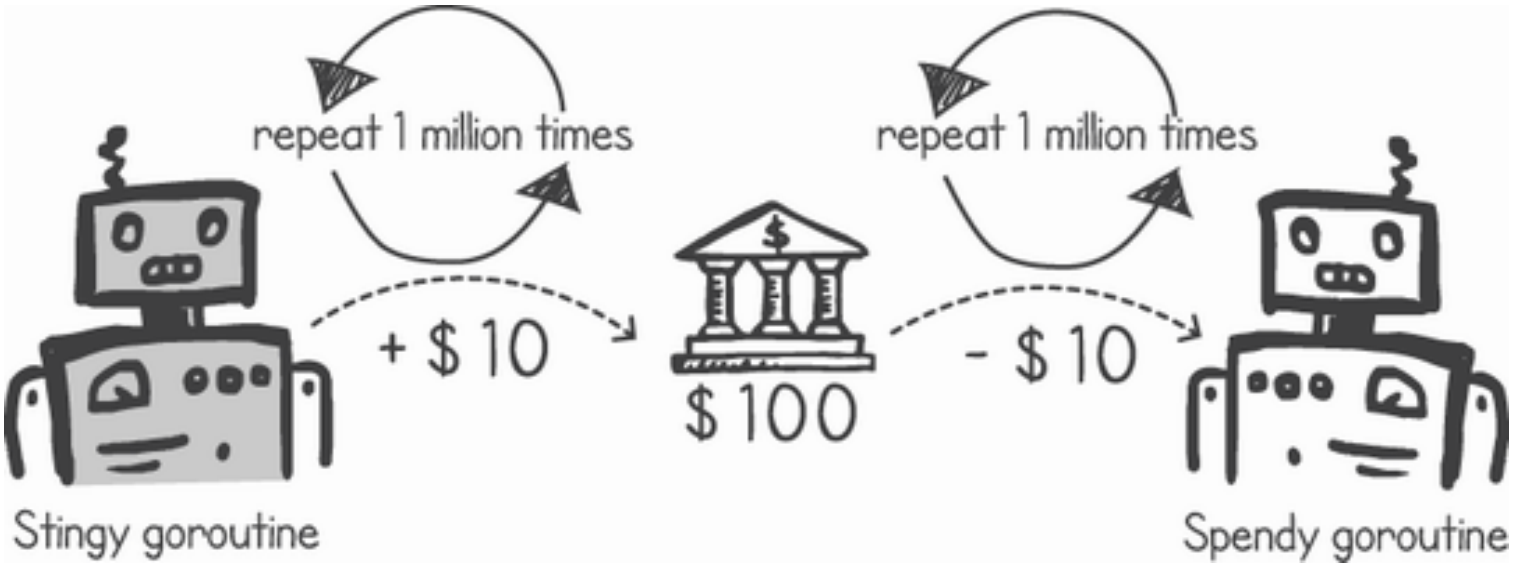
Condición de carrera

Caso: Compra de comestibles



Condición de carrera

Caso: Stingy y Spendy



Condición de carrera

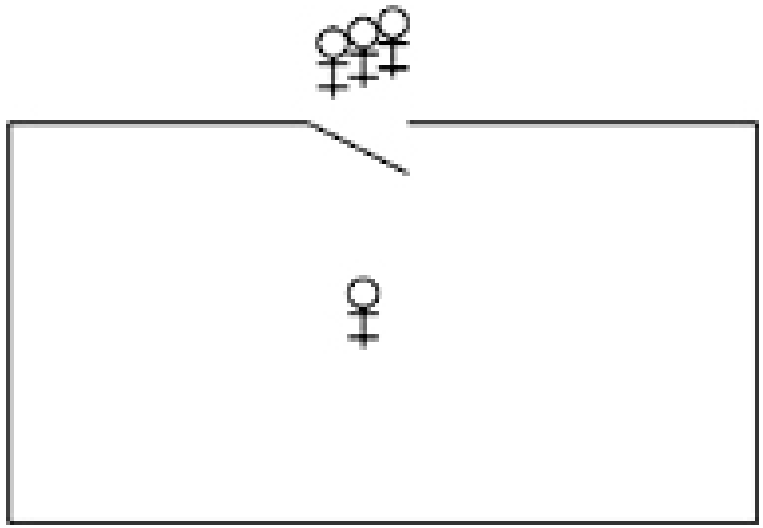
- Las condiciones de carrera son lo que sucede cuando un programa intenta hacer muchas cosas al mismo tiempo y su comportamiento depende del momento exacto de eventos independientes e impredecibles.
- Se dan cuando surgen resultados inesperados debido al intercambio de recursos, como la memoria, entre goroutines.

Contenidos

Sección Crítica

Una sección crítica es un conjunto de instrucciones que deben ejecutarse sin interferencia de otras ejecuciones simultáneas. Cuando se permite que ocurra interferencia, pueden ocurrir condiciones de carrera.

James Cutajar



Definición del problema

- Cada uno de N procesos está ejecutando en un bucle infinito una secuencia de declaraciones que se pueden dividir en dos subsecuencias: la sección crítica (CS) y la sección no crítica (NCS).
- Las especificaciones de corrección requeridas de cualquier solución son:
  - **Mutual exclusion.** No se deben intercalar declaraciones de las secciones críticas de dos o más procesos.
  - **Freedom from deadlock.** Si algunos procesos intentan ingresar a sus secciones críticas, entonces uno de ellos eventualmente tendrá éxito.
  - **Freedom from starvation.** Si un proceso desea ingresar a su CS (en espera de su turno) y otro proceso se niega a establecer el turno, se dice que el primer proceso está starved (muerto de hambre), entonces eventualmente ese proceso debe tener éxito.
  - **Freedom from livelock.** A veces puede existir un ciclo de transiciones desde un estado para cada proceso, desde el cual no se puede hacer ningún progreso útil en el programa paralelo, entonces no debe haber tiempos muertos.

Definición del problema

- Se debe proporcionar un mecanismo de sincronización para garantizar que se cumplan los requisitos de corrección. El mecanismo de sincronización consta de declaraciones adicionales que se colocan antes y después de la sección crítica.

```
global variables
```

```
p                                   q
```

```
local variables
loop forever
   non-critical section
   preprotocol
   critical section
   postprotocol
```

```
local variables
loop forever
   non-critical section
   preprotocol
   critical section
   postprotocol
```