



Departamento de Programación
Facultad de Informática
Universidad Nacional del Comahue



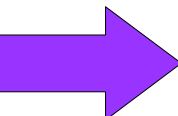
Programación Concurrente



Propiedades de concurrencia - Problemas

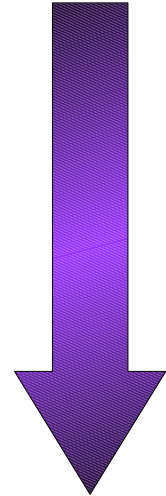


Continuamos con ... problemas de PC

- Trabajamos la concurrencia en un escenario de memoria compartida.
 - Los hilos tienen, en general, que consultar y actualizar variables compartidas.
 - Las acciones entre las tareas involucradas pueden entrelazarse en cualquier orden, y al trabajar sobre las mismas variables pueden producir inconsistencia ----> *condiciones de carrera* (es decir cual es el hilo que gana la carrera en el acceso al dato compartido)
 - Es necesario trabajar con “bloqueos” para proporcionar “exclusión mutua” en el acceso al área compartida, a la que llamamos “sección crítica”
 - El lenguaje de programación provee la forma de producir esos bloqueos, para permitir que un hilo tome el control del área compartida, y pueda tener la exclusividad de trabajo mientras dure el bloqueo.
- 
- Otros hilos no podrán leer ni escribir sobre ese área compartida hasta que se libere el bloqueo.

Propiedades que deben cumplir los PC

- Seguridad
 - recursos compartidos Thread Safe
 - no producir inconsistencias
- Viveza
 - todos los hilos deben poder progresar en sus acciones



Sincronización y comunicación entre los hilos

POO concurrente

correctitud

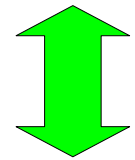
- Sistema orientado a objetos

- Centrado en objetos:

- colección de objetos interconectados

seguridad

reusabilidad



- Centrado en actividades

- Colección de actividades posiblemente concurrentes
 - Cada actividad puede envolver varios hilos

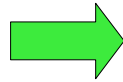
viveza

performance

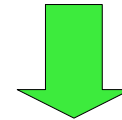
- Un objeto puede estar envuelto en múltiples actividades
 - Una actividad puede abarcar múltiples objetos

POO concurrente - Seguridad

Asegurar consistencia



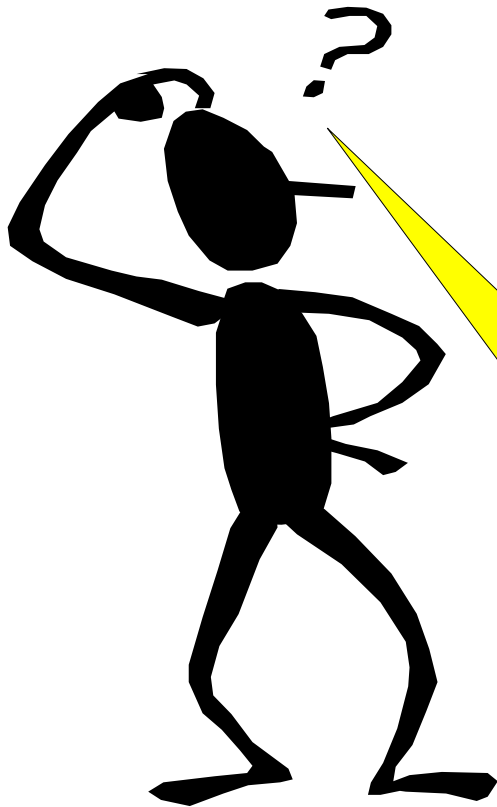
Emplear técnicas de exclusión




Garantizar la atomicidad de acciones públicas

Cada acción se ejecuta hasta que se completa sin interferencia de otras

Un balance de cuenta bancaria es incorrecto después de un intento de retirar dinero en medio de una transferencia automática



POO concurrente - Seguridad

- Inconsistencias  **condiciones de carrera**
- Conflictos de lectura/escritura: *un hilo lee un valor de una variable mientras otro hilo escribe en ella. El valor visto por el hilo que lee es difícil de predecir – depende de que hilo ganó la “carrera” para acceder a la variable primero*
- Conflictos de escritura/escritura: *dos hilos tratan de escribir la misma variable. El valor visto en la próxima lectura es difícil de predecir*

Ejemplos de condición de carrera

Cuando el resultado depende de la ejecución de los hilos.

Problema: varios hilos hacen **lectura + modificación + escritura** de una variable y el ciclo completo se interrumpe

Muy difícil de detectar, pues el programa puede funcionar bien muchas veces y fallar de repente.

	thread 1 { $x = x + 5;$ }	thread 2 { $x = x + 100;$ }
t0	x vale 100	
t1	lee 100	
t2	incrementa a 105	
t3		lee 100
t4	escribe 105	
t5		incrementa a 200
t6		escribe 200
t7	x vale 200	



Seguridad (safety)

- Un programa es seguro cuando no genera nunca resultados incorrectos.
- No tiene inconsistencias

Vivacidad (liveness)

- Cuando el programa responde ágilmente a las solicitudes del usuario.
 - cada actividad eventualmente progresa hacia su finalización.
 - cada método invocado eventualmente se ejecuta

Propiedades de la PC

Seguridad (safety)

Nada malo va a pasar

- Exclusión mutua

Algo bueno va a pasar

Viveza (liveness)

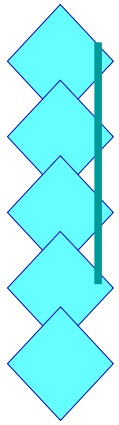
- Evitar inanición/starvation
- Livelock (interbloqueo activo)
- Evitar deadlock (interbloqueo pasivo)



Propiedades de la PC

Viveza (**liveness**)

- Se debe asegurar que todos los procesos *progresen* durante la ejecución del programa
 - Interbloqueo (activo) – **livelock**: una acción de reintentar continuamente, continuamente falla
 - **Inanición** – **starvation**: la JVM/SO falla en asignar tiempo de CPU a un hilo
 - Interbloqueo (pasivo) – **deadlock**: dependencias circulares entre locks

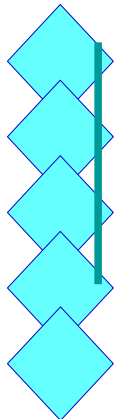


Estados consistentes - sistemas seguros

Exclusión mutua: mantener estados consistentes de los objetos evitando interferencias NO deseadas entre actividades concurrentes

Clases “Thread safe” , es decir **clases seguras para un ambiente concurrente**

¿La clase “pila” implementada en materias anteriores es Thread safe?



Prog concurrente – Seguridad y Viveza

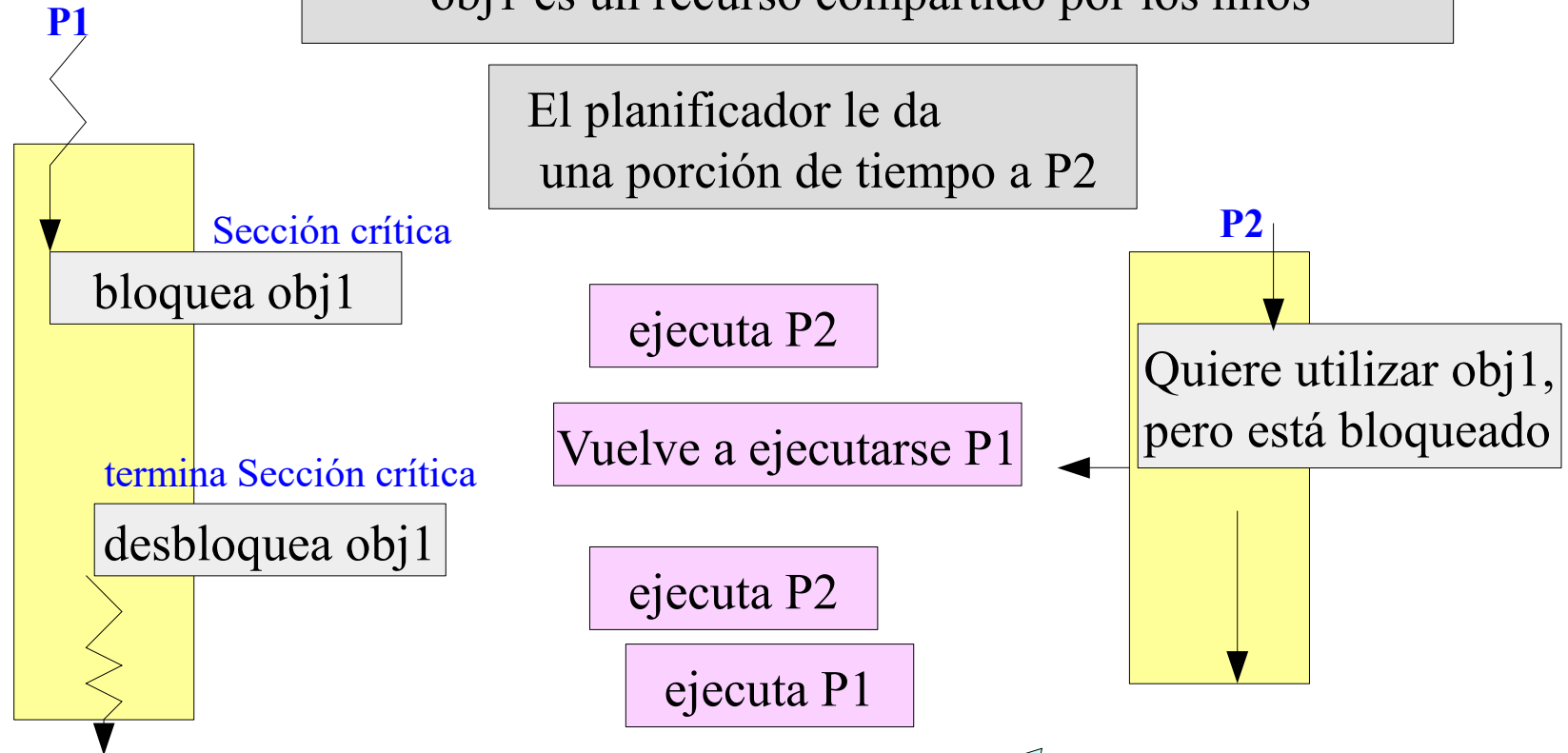
Problema serio: falta de progreso permanente

- **Deadlock:** dependencias circulares
 - “deadlock” (interbloqueo por inactividad)
- **Livelock:** una acción falla continuamente
 - “livelock” (interbloqueo por hiperactividad)
- **Starvation/inanición:** un hilo espera por siempre, la maquina virtual falla siempre en asignarle tiempo de CPU
- **Falta de recursos:** un grupo de hilos tienen todos los recursos, un hilo necesita recursos adicionales pero no puede obtenerlos
- Otros (investigar)

Deadlock: Situación 1

obj1 es un recurso compartido por los hilos

El planificador le da
una porción de tiempo a P2



Se ejecutan ambos procesos
NO HAY DEADLOCK

Deadlock: Situación 2

obj1 y obj2 son recursos compartidos por los hilos

P1

Sección crítica

bloquea obj1

Quiere usar obj2

~~desbloquea obj1~~

termina Sección crítica

El proceso P1 no logra llegar al desbloqueo

ejecuta P2

Vuelve a ejecutarse P1

ejecuta P2

ejecuta P1

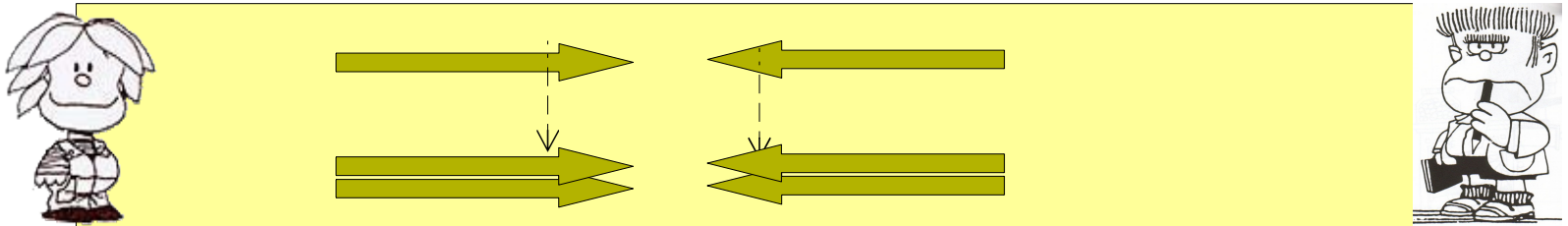
P2

Bloquea obj2

Quiere usar obj1

Ningun proceso desbloquea
DEADLOCK

Livelock



- Un livelock es similar a un deadlock,
- Los dos procesos envueltos con respecto al otro.
- Livelock es una forma de inanición
- Ejemplo: dos personas, al encontrarse en un pasillo angosto avanzando en sentidos opuestos, y cada una trata de ser amable moviéndose a un lado para dejar a la otra persona pasar
- Ninguna puede pasar pues ambos se mueven hacia el mismo lado, al mismo tiempo.
- Livelock es un riesgo que se puede detectar y recuperar, asegurando que sólo un proceso (escogido al azar o por prioridad) tome acción.



Resumiendo...

- La computación en paralelo utiliza múltiples elementos de procesamiento para resolver un problema
- Para ello se divide el problema en partes independientes para que cada elemento de procesamiento pueda ejecutar la parte que le corresponde del algoritmo en simultáneo con los demás
- Se considera el principio de que problemas grandes pueden dividirse (aveces) en subproblemas mas pequeños, que pueden ser resueltos en forma simultánea. Estrategia “divide y vencerás”
- Los problemas informáticos concurrentes/paralelos son mas difíciles de diseñar e implementar que los secuenciales.
- La concurrencia introduce nuevos tipos de errores de software.
- La comunicación y sincronización entre diferentes subtareas deben ser tenidos en cuenta y son fundamentales para obtener un buen resultado.



Continuamos con ... problemas de PC

- Cuando se necesita trabajar sobre varias variables compartidas, ¿cómo podemos producir el bloqueo? .
 - Puede hacerse un bloqueo de “un grupo de variables”,
 - *bloqueo atómico*
 - Puede trabajarse con bloqueos individuales
 - *bloqueo no atómico*
 - puede generarse un bloqueo mutuo entre los hilos
- Al existir varios hilos que comparten el recurso CPU, y comparten recursos, puede producirse:
 - Starvation (inanición)
 - Livelock (interbloqueo activo)
 - Deadlock (interbloqueo pasivo)
 - Condiciones de carrera