

# Parcial 2 S.O Juan Pablo Bedoya Sánchez

## → Hilos / Concurrencia

Concurrencia = Ver que se ejecutan procesos de manera simultánea

Hilos: Varios puntos de ejecución dentro de un proceso

Parallelismo: Varias funciones ejecutadas al mismo tiempo / simultáneamente

\* Hilos no comparten espacio stack

Para que sirven los hilos = Parallelismo / concurrencia en el mismo trabajo / aprovechar paralelismo

Concurrencia no quiere decir paralelismo.

parallelismo si requiere concurrencia

parallelismo aumenta desempeño de las aplicaciones

→ Cambio Contexto entre hilos = hilos comparten espacio de direccionamiento, pero no el stack → pila independiente

→ Race Condition: hilos acceden de manera concurrente

a la misma posición de memoria.

→ Sección Crítica: Región Crítica: Variables compartidas

sección de código donde se presenta race condition.

Usar exclusión mutua → atomicidad.

→ LOCKS = acceso exclusivo en una sección crítica

asegura que una sección crítica se execute como

instrucción atómica. → dos estados: Disponible / Adquirido

exclusión mutua: Solo permite que un hilo esté en

la región crítica. NO proporciona acceso equitativo

a todos los hilos. Desempeño: - mono procesador:

alto costo → # procesadores es comparable con # de hilos, buen desempeño.

Librería mutex

## Implementación de un lock:

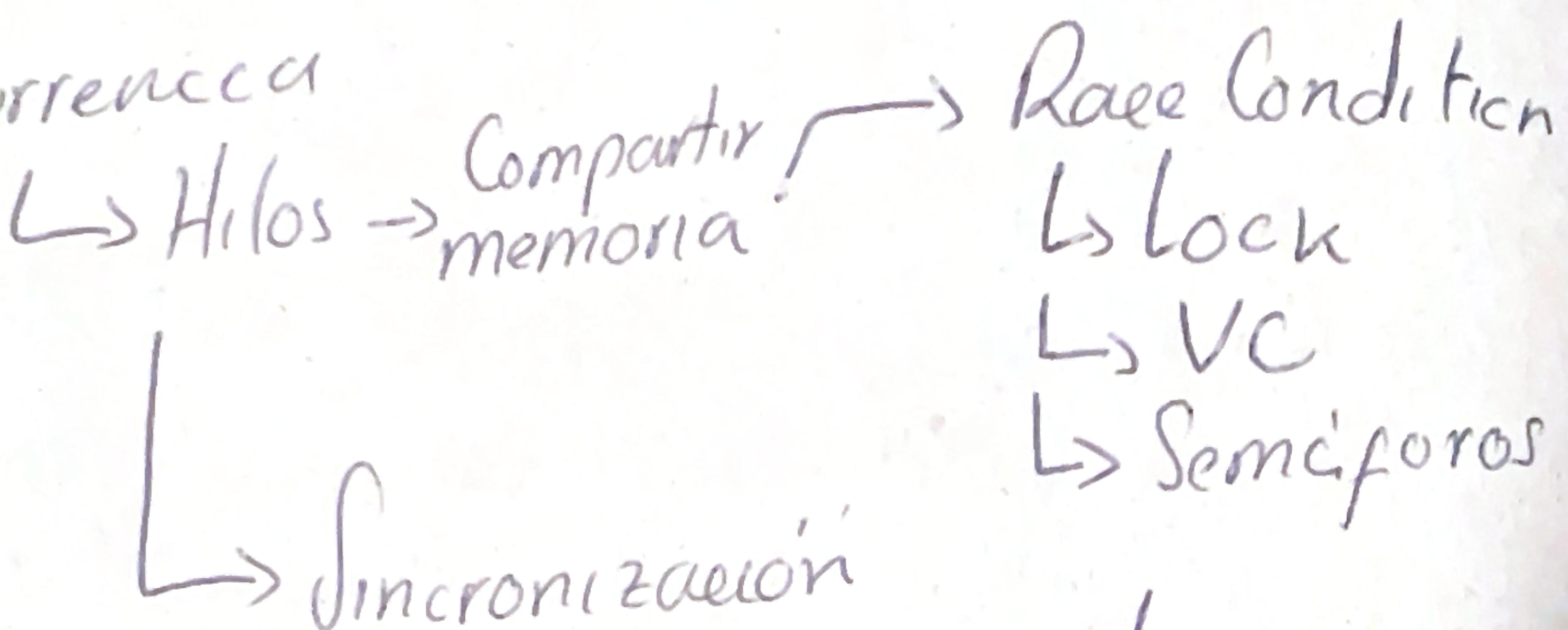
- req ayuda del HW y SW → Criterios de evaluación: a) exclusión mutua b) equidad
- c) Desempeño.
  - Deshabilitar interrupciones al entrar en las secciones críticas. → utilizar un flag para determinar el estado del lock (iniciación ciclos infinitos)
  - test and set = HW para soportar la creación de lock
  - Compare and swap = chequera si el valor \*ptr es igual al esperado.
  - Se hace de manera atómica = spin-lock
    - Se hace de manera atómica = spin-lock
    - un bloqueo de una espera activa. No garantiza equidad
  - ~~load-linked~~ load-linked sotore-conditional
  - Fetch and add = incrementa de manera atómica una posición de memoria retornando valor antiguo
  - Ticket lock = asegura equidad → Dos variables ticket y turno → turno indica turno cada hilo fetch and add
  - Solución al gasto de CPU por el While:
    - \* Yield. → llamar el yield del S.O.
    - \* Estado sleep.
    - \* Uso de colas = cuales hilos están esperando por el lock
  - # Variables de condición: variables se necesitan para un hilo poder avanzar.

- spin-waiting: padre \*espera por hijo
- funcion wait: hilo  $\rightarrow$  cola
- funcion signal: otro hilo  $\rightarrow$  cambio de estado
- Padre: Crea hijo y continua ejecución (sleep)
- Hijo: mensaje child  $\rightarrow$  llama función thr\_exit para despertar padre
- Importancia mutex =
- problema Productor - Consumidor = hilo productor accede a un buffer  $\rightarrow$  produce ítems de datos.
  - hilo consumidor  $\rightarrow$  consume ítems
- Versión 1:
  - solo ingrese al buffer cuando esté vacío
  - solo retire datos del buffer cuando esté lleno
  - Siempre variable de condición debe ir en un while.
- \* SEMAFOROS = un elemento de sincronización que usa un contador, mezclar entre lock y variable de condición
- Semáforos binarios = locks.  $S=1$ , inicializar.
  - Semáforos como variables de condición.
- \* Problemas de sincronización
- lectores-escritores/cena filósofos) Barbero-Dormilon → Interbloqueo (No deseado). Dos hilos se bloquean
  - Deadlock = ejecutar un programa (no avanza)
  - Reader-Writer-Lock = Solo un escritor puede adquirir el lock / multiples lectores

oxapost  
→ Reina de filósofos.  $\Rightarrow$  Dead lock = es el bloqueo de todos los hilos a la espera de un evento que solo puede pasar en ese conjunto.  
solución: alterar el orden de uno de ellos

→ Problemas Conurrencia:

Conurrencia



Problemas Clásicos  $\rightarrow$  Productor - Consumidor  
 $\hookrightarrow$  lectores - escritores

Problemas  $\rightarrow$  Interbloqueos.

$\hookrightarrow$  No derivados de interbloqueos = \* Violación de atomicidad  
 $\Rightarrow$  Se intrrumpe por el scheduler - exclusión mutua.  
\* Violación de orden: el orden de acceso a memoria se invierte  $\rightarrow$  Se asegura orden de ejecución= variable de condición - Semáforos.

→ Interbloqueos:  
Dependencias complejas

1- Prevenir

2- Evitar

3- Detectar y Solucion

4- Nada.

