



ALGORITMOS DE EXCLUSIÓN MUTUA

SEMINARIO DE PROGRAMACIÓN CONCURRENTE, PARALELA Y DISTRIBUIDA
ALAN EXARCHOS, ERNESTO DAVOGUSTTO, FLORENCIA ROSSI

CONTENIDO

01

PROGRAMACIÓN CONCURRENTE

02

EXCLUSIÓN MUTUA

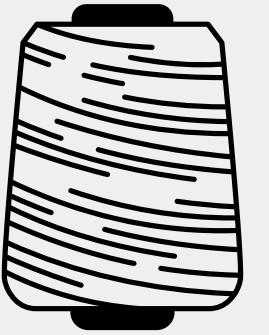
03

ALGORITMOS DE EXCLUSIÓN MUTUA

04

CONCLUSIONES

PROGRAMACIÓN CONCURRENTE

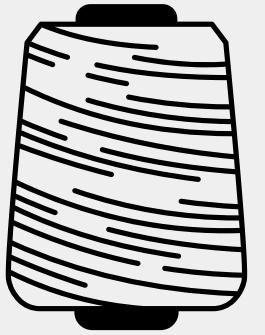


Los procesos están relativamente superpuestos en el tiempo y pueden operar sobre el mismo espacio en memoria y utilizar los mismos recursos.



Problema a evitar -> DEADLOCK

EXCLUSIÓN MUTUA



Asegurar las SECCIONES CRITICAS
-> Deben ser ejecutadas de manera indivisible



Premisas de Dijkstra

DEKKER



Algoritmo 5

- Permite a dos procesos o hilos de ejecución compartir un recurso sin conflictos. Fue uno de los primeros algoritmos de exclusión mutua inventados
- Si ambos procesos intentan acceder a la sección crítica simultáneamente, el algoritmo elige un proceso según una variable de turno. Si el otro proceso está ejecutando en su sección crítica, deberá esperar su finalización.

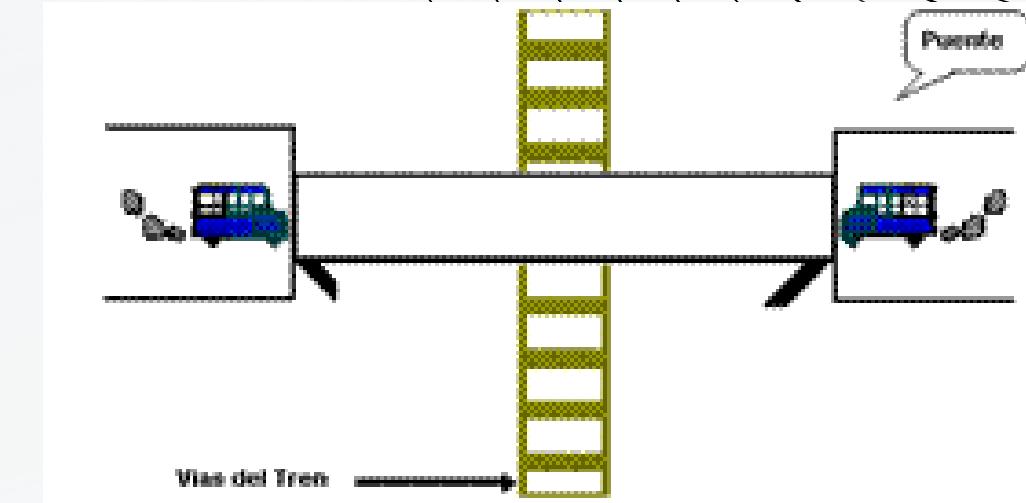
Combina ideas de versiones anteriores para resolver problemas de concurrencia:

- Usa **banderas de intención** (de las versiones 2 y 3) para indicar el deseo de entrar en la sección crítica.
- **Turno compartido** (de las versiones 1 y 4) para decidir qué proceso avanza en caso de conflicto.

DEKKER - VENTAJAS Y DESVENTAJAS

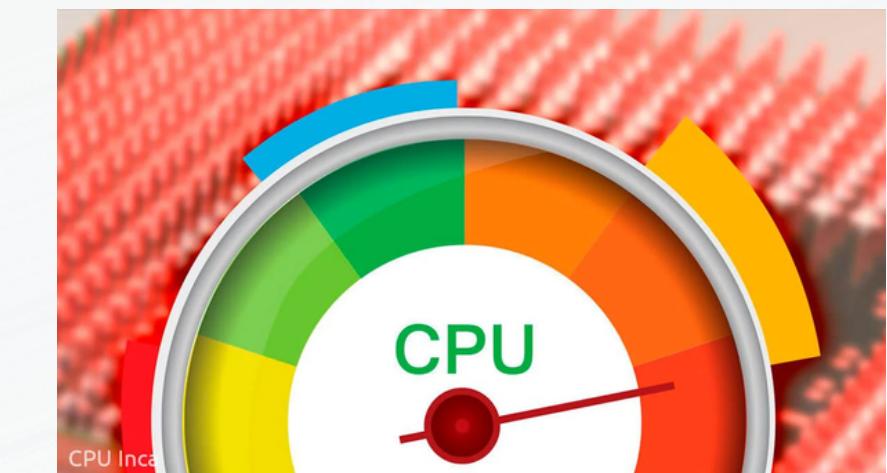
VENTAJAS

- 1. Exclusión mutua garantizada
- 2. Evita interbloqueo
- 3. Evita inanición
- 4. No requiere hardware especial.
- 5. Equidad.



DESVENTAJAS

- 1. Espera activa
- 2. Escalabilidad limitada
- 3. Complejidad
- 4. Rendimiento bajo
- 5. Riesgo de bloqueo



PETERSON



- Permite compartir recursos utilizando únicamente memoria compartida.
- Cumple con los requisitos para resolver un problema de sección critica.

IMPLEMENTACIÓN

1. Se usan dos variables compartidas: flag[i] y turn.

FUNCIONAMIENTO

- Un proceso desea entrar a la sección crítica.
- Verificación de entrada.
- Si ambos procesos quieren entrar al mismo tiempo.
- Salida de la sección crítica.



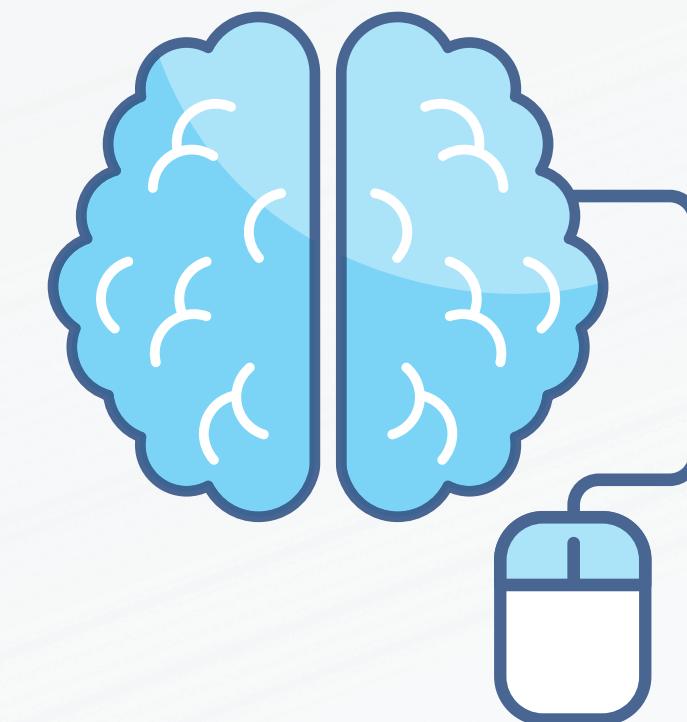
PETERSON- VENTAJAS Y DESVENTAJAS

VENTAJAS

- 1.Exclusión mutua garantizada
- 2.Evita inanición
- 3.Tiene una lógica fácil de implementar y llevar a práctica
- 4.Al ser una solución de software se puede usar en cualquier hardware

DESVENTAJAS

- 1.Utiliza espera activa, por lo que puede desperdiciar recursos.
- 2.No escala bien para más de dos procesos.



BAKERY



- Otra solución al problema de exclusión mutua es de Leslie Lamport , se la conoce como el algoritmo de la panadería (bakery algorithm) por su similitud a los clientes de una tienda que sacan un número para ser atendidos.

IMPLEMENTACIÓN

- 1.Cada proceso tiene un ticket.
- 2.Cada proceso espera hasta que su valor del ticket sea el mínimo entre todos los procesos esperando.
- 3.El proceso con el valor mínimo accede la sección crítica.

BAKERY- VENTAJAS Y DESVENTAJAS

VENTAJAS

- 1.Exclusión mutua garantizada
- 2.No requiere hardware especial.

DESVENTAJAS

- 1.Rendimiento bajo
- 2.Consumo de memoria

CONSIDERACIONES

INCÓVENIENTE

1. Puede suceder que varios hilos puedan obtener el mismo número de turno.
2. Mientras un hilo está seleccionando su número (entre leer los números de otros threads y asignarse el más alto), otros threads pueden intentar leer su número antes de que haya terminado de seleccionarlo, lo que podría causar una inconsistencia temporal.

SOLUCIÓN

1. Se aplica un algoritmo de desempate, que garantiza que sólo un hilo entra en sección crítica. Consumo de memoria.
2. Para evitarlo hay que impedir que un proceso avance si el proceso contra el que está por comparar su número todavía lo está seleccionando. Para ello se usa otro array, choosing, que indicará si el proceso está en medio de la selección.

IMPLEMENTACION

```
number = [0] * cant_hilos # Array que guarda los turnos de cada thread
choosing = [False] * cant_hilos # Flags para indicar que un thread esta eligiendo su numero

def obtener_max(): # Funcion para retornar el mayor número en el array number.
    return max(number)

def bakery_lock(thread_id):
    global number, choosing

    choosing[thread_id] = True # Indica que el thread esta eligiendo su numero
    number[thread_id] = 1 + obtener_max()
    choosing[thread_id] = False # Termina de elegir su numero

    for j in range(cant_hilos): # Esperar a que los demás threads obtengan su número y respetar el turno
        if j == thread_id:
            continue # Saltar la comparación consigo mismo
        while choosing[j]: # Esperar a que el otro thread termine de elegir su numero
            pass # Mantenerse en este bucle mientras el otro thread esta eligiendo
        while number[j] != 0 and (number[j], j) < (number[thread_id], thread_id): # Esperar a que los threads con numeros mas bajos (o igual número y menor ID) terminen
            pass

def bakery_unlock(thread_id):
    global number
    number[thread_id] = 0 # Libera el turno
```

GRACIAS