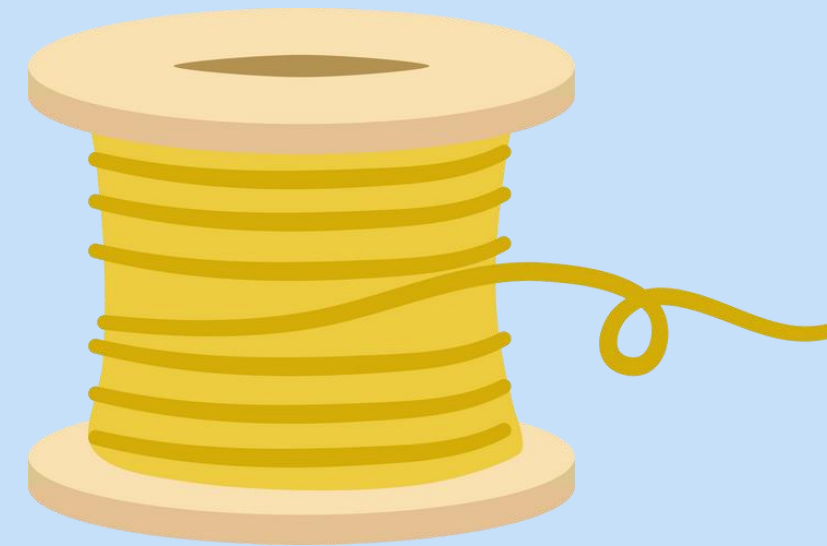
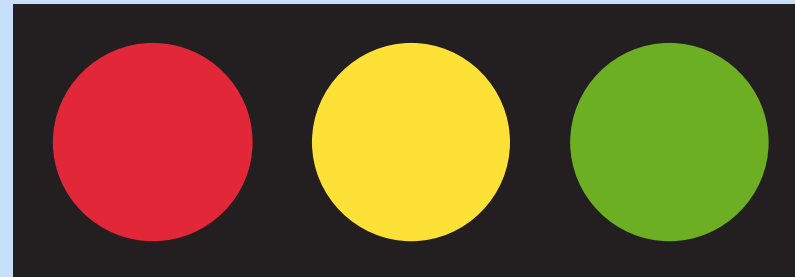


2.1.5 Semáforos

2.1.6 Pools de hilos



EQUIPO

- **Garcia Morales Rebecca**
- **Jimenez Anaya Erick**
- **Ramirez Vega Gerardo**

Agenda de hoy

○ ○ ○ ○

1

Introducción del tema

2

Desarrollo

3

Conclusión

4

Referencias

Introducción



En el mundo interconectado de hoy, la comunicación entre dispositivos a través de redes es esencial. Sin embargo, esta comunicación a menudo implica desafíos relacionados con la concurrencia y la coordinación de múltiples tareas. Es aquí donde entran en juego los semáforos e hilos.

Cuando hablamos de semáforos e hilos en el contexto de redes, generalmente nos referimos a cómo se utilizan en la programación de aplicaciones que involucran comunicación entre dispositivos a través de una red, como aplicaciones cliente-servidor o sistemas distribuidos.

Semáforos



¿Qué son?

Los semáforos son señales de control utilizadas para gestionar el acceso a recursos compartidos en un entorno de red.

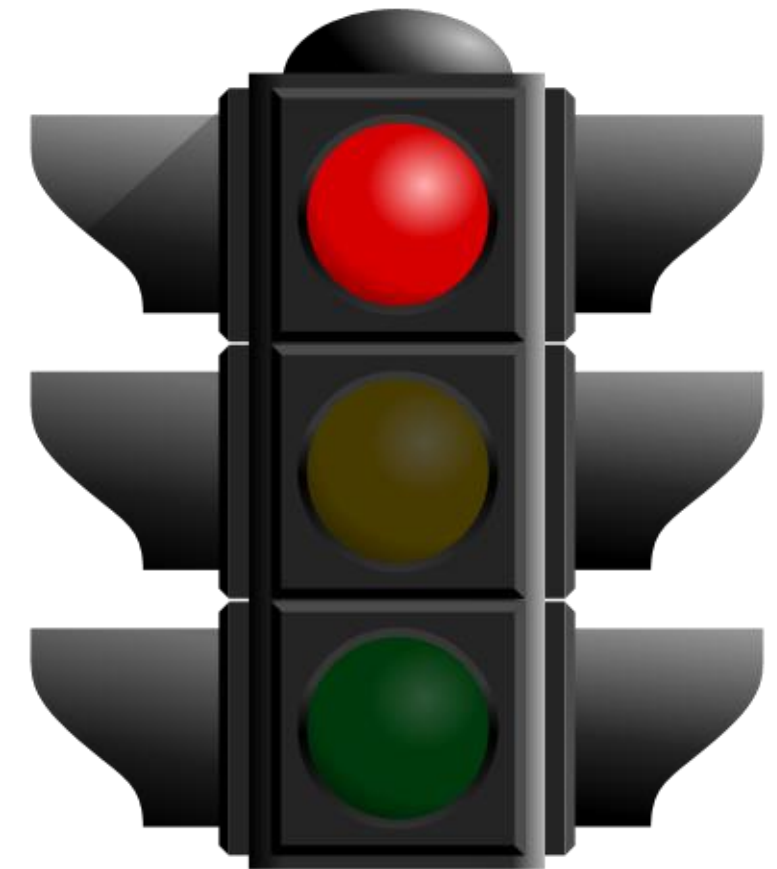
Para qué sirven



- Coordinar y controlar el acceso a recursos compartidos
- Ayudan a evitar colisiones
- Gestionar el flujo de datos
- Garantizar la integridad de la comunicación

Características

1. Coordinación de acceso
2. Evitar colisiones
3. Control de flujo de datos
4. Sincronización
5. Mejora del rendimiento



SEMÁFOROS

Los semáforos son un mecanismo de sincronización de procesos inventados por Edsger Dijkstra en 1965.

Los semáforos permiten al programador asistir al planificador del sistema operativo en su toma de decisiones de manera que permiten sincronizar la ejecución de dos o más procesos.

Semáforo:

Existen 3 estados de los procesos en el CPU



Bloqueado: Deben ocurrir ciertos eventos externos para el proceso pueda ejecutarse.

Listo: Está detenido temporalmente por la ejecución de otro proceso.

En ejecución: Este proceso se está ejecutado en este momento

Tipos de semáforos

Exclusión mutua

Dependiendo de la función que cumple el semáforo, vamos a diferenciar los siguientes tipos:

- Semáforo de exclusión mutua, inicialmente su contador vale 1 y permite que haya un único proceso simultáneamente dentro de la sección crítica.

Contador

Semáforo contador, permiten llevar la cuenta del número de unidades de recurso compartido disponible, que va desde 0 hasta N

Tipos de semáforos

Espera

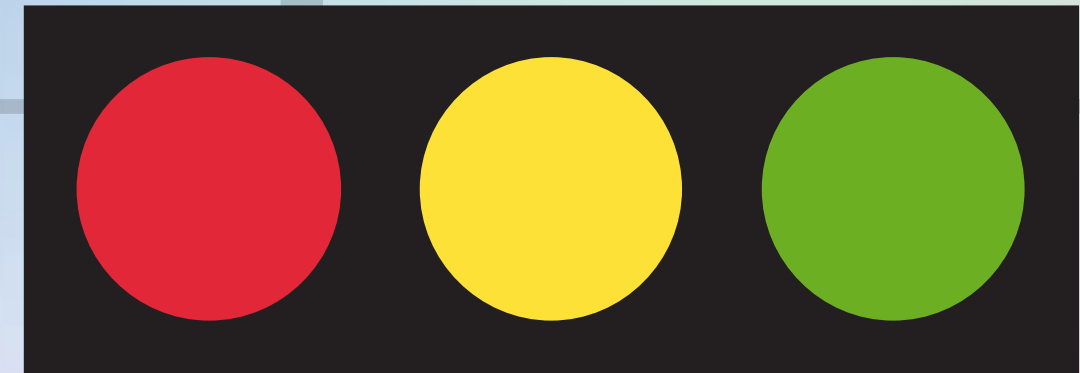
- Semáforo de espera, generalmente se emplea para forzar que un proceso pase a estado bloqueado hasta que se cumpla la condición que le permite ejecutarse. Por lo general, el contador vale 0 inicialmente, no obstante, podría tener un valor distinto de cero.

```
semaforo a = semaforo.create(1);  
semaforo b = semaforo.create(0);
```

```
/* código del hilo A */  
while (1) {  
    a.down();  
    printf("hilo A\n");  
    b.up();  
}
```

```
/* código del hilo B */  
while (1) {  
    b.down();  
    printf("hilo B\n");  
    a.up();  
}
```

Ventajas



La principal ventaja de los semáforos frente a los [cerrojos](#) es que permiten sincronizar dos o más procesos de manera que no se desperdician recursos de CPU realizando comprobaciones continuadas de la condición que permite progresar al proceso.

Desventajas



Los inconvenientes asociados al uso de semáforos son los siguientes:

- Los programadores tienden a usarlos incorrectamente, de manera que no resuelven de manera adecuada el problema de concurrencia o dan lugar a interbloqueos.
- No hay nada que obligue a los programadores a usarlos.
- Los compiladores no ofrecen ningún mecanismo de comprobación sobre el correcto uso de los semáforos.
- Son independientes del recurso compartido al que se asocian.

Debido a estos inconvenientes, se desarrollaron los [monitores](#).

Granularidad

Número de recursos controlados por cada semáforo.

Hay de dos tipos:

- Granularidad fina: Un recurso por semáforo. Hay un mayor grado de paralelismo, lo que puede mejorar la rapidez de ejecución de la protección. Aunque a mayor número de semáforos existe una mayor probabilidad de que se dé un interbloqueo entre semáforos, que no sería una mejora de lo que intentamos evitar.

Granularidad gruesa

- Granularidad gruesa: Un semáforo para todos los recursos. Caso totalmente inverso al anterior: Ahora al tener un semáforo no se produce interbloqueo entre ellos, aunque los tiempos de ejecución son excesivamente largos.

Erick

Pools de hilos

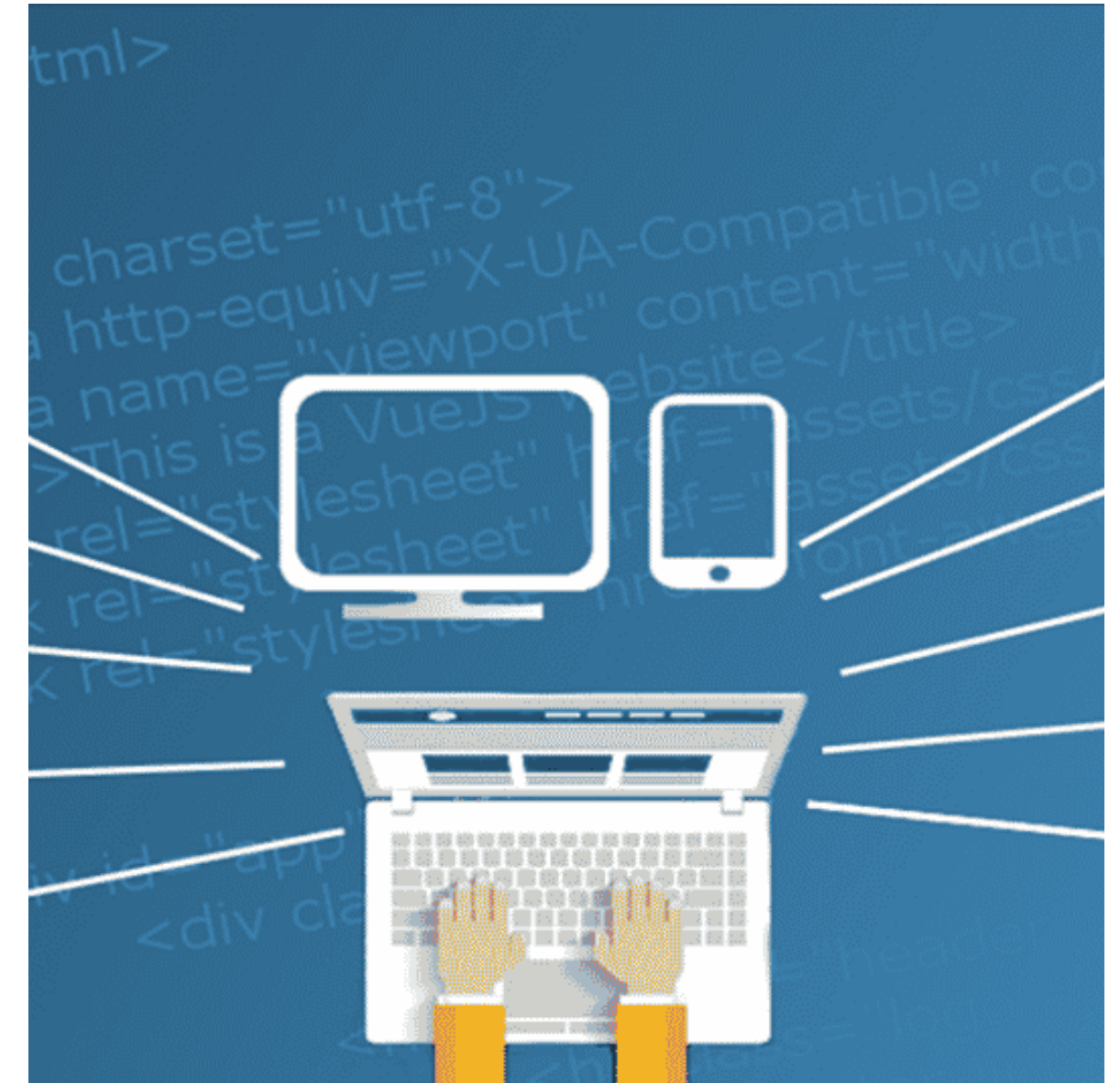
¿Estás listo?

Temas

- Técnica "Polling"
- Desventajas del "Polling"
- Eficiencia en el uso de hilos de ejecución
- Tamaño y gestión de grupos de hilos
- Consideraciones adicionales para determinar el tamaño del grupo de hilos
- Ejemplo

Técnica "Polling"

Una técnica en la que un programa verifica repetidamente el estado de un recurso para determinar si ha ocurrido algún evento o si el recurso está listo para ser utilizado.

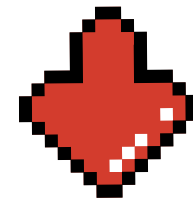


Aplicación en hilos de ejecución

Quando se utilizan hilos de ejecución, el "polling" se usa típicamente para que un hilo principal pueda verificar si otro hilo ha completado una tarea específica o ha producido algún resultado relevante.

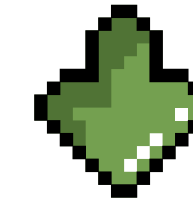
Desventajas del "polling" en hilos de ejecución

Ineficiencia y uso ineficiente de recursos



El "polling" es una técnica sencilla pero puede ser ineficiente debido a que el hilo principal verifica constantemente el estado del recurso, consumiendo recursos de CPU y aumentando la latencia.

Riesgos y limitaciones



El uso de "polling" no asegura resultados precisos y puede ocasionar problemas de sincronización. Si el hilo principal verifica el estado del recurso demasiado pronto o demasiado tarde, podría tomar decisiones basadas en información desactualizada o incorrecta.

Eficiencia en el uso de hilos de ejecución

Importancia en programas
ligados a I/O:



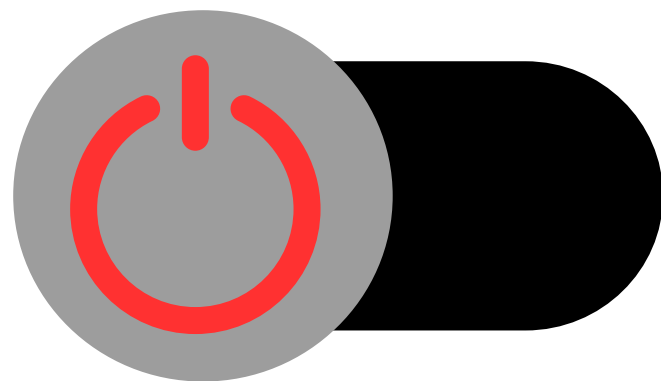
Gestión eficiente de grupos de hilos



Tamaño y gestión de grupos de hilos

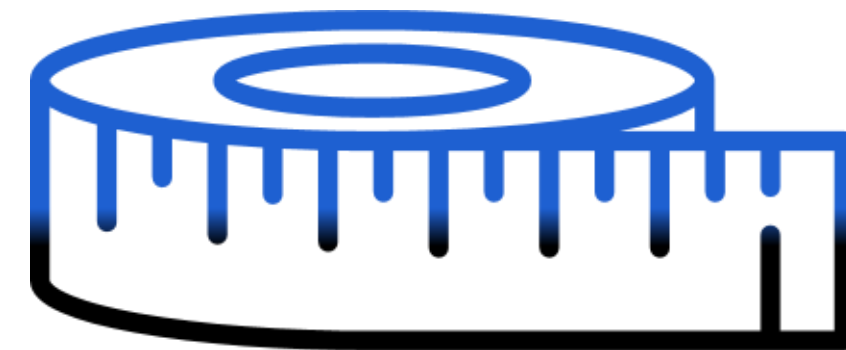
Shutdown y ShutdownNow

Al finalizar el uso de un pool de hilos, es crucial realizar una finalización adecuada para liberar los recursos. El método `shutdown()` notifica al pool que no se agregarán más tareas y permite que complete las pendientes. Mientras que `shutdownNow()` aborta las tareas en ejecución y no procesa las pendientes.




Determinación del tamaño ideal

El tamaño óptimo de un grupo de hilos depende de diversos factores, como las características del sistema y el tipo de tareas a ejecutar. Es crucial calcular dinámicamente o configurar el tamaño del grupo basado en estas consideraciones para maximizar la eficiencia y el rendimiento.



**Consideraciones
adicionales para
determinar el tamaño del
grupo de hilos**



Factores a tener en cuenta

El tamaño del grupo de hilos debe considerar el número de procesadores, la memoria disponible y la naturaleza de las tareas. Tareas intensivas en CPU pueden requerir un tamaño diferente al de las tareas de E/S. Ajustar el tamaño según estos factores mejora la eficiencia y el rendimiento.



Estrategias para determinar el tamaño del grupo

Se pueden emplear diferentes estrategias para determinar el tamaño óptimo del grupo de hilos. Esto incluye calcular dinámicamente el tamaño según el número de procesadores disponibles o estimar la relación entre el tiempo de espera y el tiempo de cálculo de las tareas.

Erick

Ejemplo



CONCLUSION

Comprender cómo trabajar con semáforos e hilos es fundamental para desarrollar software robusto y eficiente en entornos concurrentes. Sin embargo, como veremos, también presenta desafíos únicos que debemos abordar con cuidado y atención.

Referencias

[1] Harold, E. R. (2014). Java Network Programming (4th ed.). O'Reilly.

<https://lib.fbtuit.uz/assets/files/Java-NetworkProgrammin.pdf>

[2] Goetz, B. (2010). Java Concurrency in Practice. Addison-Wesley.

<https://github.com/AngelSanchezT/books1/blob/master/concurrency/Java%20Concurrency%20in%20Practice.pdf>