ALGORITMOS CONCURRENTES

De la Computación Concurrente

ALGUNOS DE LOS MÁS POPULARES

1

DEKKER

- Es un algoritmo de programación concurrente para exclusión mutua el cual consiste en que dos procesos quieren entrar a su región crítica por lo que cada proceso indica que quiere acceder su región crítica para después preguntar si el otro proceso también quiere acceder a su región crítica si si quiere entrar a su región crítica entonces preguntamos de cuál proceso es el turno para entrar a su región crítica si es el turno del otro proceso entonces cedemos el turno para después esperar a que sea nuestro turno (esto mediante una espera ocupada) y cuando sea nuestro turno indicamos que se quiere entrar a la región crítica en otro caso entra el proceso a su región crítica y después de acceder a la región crítica ahora secedemos el turno al otro proceso y decimos que no queremos entrar a la región crítica par después ejecutar la parte de la región no crítica y después volver repetir todo desde el principio
- Problemas: este algoritmo solo se pueden usar 2 hilos, para ir checando cual proceso será el turno de entrar a su región crítica se hace mediante una espera ocupada, no suspende a los procesos que están esperando acceso y puede llegar a entrar en ciclos infinitos.

PETERSON

- Es una simplificación del algoritmo de Dekker que consiste en que se tiene un bandera por proceso por lo que el proceso indica que quiere acceder a la sección crítica colocando la bandera en true para después ceder el turno luego preguntamos si el otro proceso quiere entrar a su sección crítica y si es su turno por lo que estaremos preguntando esto hasta que una de las condiciones anteriores sea falsa (esto mediante una espera ocupada) cuando salgamos de esto podremos entrar a la sección crítica y en cuanto salgamos de la sección crítica colocaremos la bandera en false indicando que no se quiere acceder a la sección crítica para después entrar a la sección no crítica y volver repetir todo desde el principio.
- Problemas: este algoritmo solo se pueden usar 2 hilos, para ir checando si puede entrar a su región crítica lo hace mediante una espera ocupada, no suspende a los procesos que están esperando acceso.

2

3

FILTRO

- Es una generalización del algoritmo de peterson pero ahora para más de 2 procesos en donde hay n-1 salas de espera donde n es el número de procesos y cada proceso tiene un identificador y debe de ir avanzando por cada una de las salas de espera para lo cual si 2 o más procesos llegan llegan a la misma sala de espera el ultimo que llego se quede en la sala de espera y los demás avanzan a la siguiente sala donde cuando un proceso sale de la última sala de espera ya pueden entrar a su región crítica.
- problemas:se necesita una espera ocupada para que cada proceso espere a que se vayan desocupando la siguiente sala de espera para ir avanzando de sala en sala hasta llegar a la última sala y así entrar a la región crítica .

PANADERO

- ¿Qué es?

Algoritmo que se encarga de implementar la exlusión mútua para n procesos.

- ¿Cómo funciona?

- Cada proceso nuevo se le da un ticket(en orden ascendente).
- Cada proceso espera hasta que el valor de su ticket sea el menor dentro de todos los procesos en espera.
- El proceso con el valor mínimo accede a la sección crítica.

4

- De esta forma se tiene una buena gestión de los procesos con respecto a los recursos.

5

ALGORITMO DE SZYMAŃSKI

Mejor conocido como el algoritmo de la sala de espera, este algoritmo consiste en una sala de espera con una puerta de entrada y salida. Inicialmente, la puerta de entrada está abierta y la puerta de salida está cerrada. Todos los procesos que solicitan el ingreso a la sección crítica aproximadamente al mismo tiempo ingresan a la sala de espera; el último de ellos cierra la puerta de entrada y abre la puerta de salida. Luego, los procesos ingresan a la sección crítica uno por uno (o en grupos más grandes si la sección crítica lo permite). El último proceso en salir de la sección crítica cierra la puerta de salida y vuelve a abrir la puerta de entrada, por lo que puede entrar el siguiente lote de procesos

COLAS

Para usar una cola de manera concurrente, podemos usar dos implementaciones:

- Cola parcial acotada: Las llamadas a los métodos pueden esperar a que se cumplan ciertas condiciones. Al tener un límite de elementos, es acotada.
- Cola total no acotada: Las llamadas a los métodos no esperan a que se cumplan ciertas condiciones. Al no tener un límite de elementos, es no acotada.
- Cola no acotada sin bloqueo: El campo next de cada nodo es una referencia atómica al siguiente nodo de la lista. Es libre de inanición.

6

7

LISTAS

Para usar una lista de manera concurrente tenemos múltiples opciones:

- Sincronización granularidad-gruesa.
- Sincronización granularidad-fina.
- Sincronización optimista.
- Sincronización perezosa.
- Sincronización sin bloqueo.

REFERENCIAS:

- colaboradores de Wikipedia. (2021, 11 diciembre). Algoritmo de Dekker. Wikipedia, la enciclopedia libre. Recuperado 23 de octubre de 2022, de https://es.wikipedia.org/wiki/Algoritmo_de_Dekker
- Rocha, A. (s. f.-b). Algoritmo de Peterson. prezi.com. Recuperado 23 de octubre de 2022, de https://prezi.com/8zl-ggevjkdg/algoritmo-de-peterson/

7.1

SINCRONIZACIÓN GRANULARIDAD-GRUESA

Se toma una implementación secuencial de la clase, se agrega una primitiva de sincronización escalable y se asegura de que cada llamada de método adquiera y libere esa primitiva.

SINCRONIZACIÓN GRANULARIDAD-FINA

En lugar de usar un solo bloqueo para sincronizar cada acceso a un objeto, dividimos el objeto en componentes sincronizados de forma independiente. Esto garantiza que las llamadas a métodos interfieran solo cuando se intenta acceder al mismo componente al mismo tiempo.

7.2

7.3

SINCRONIZACIÓN OPTIMISTA

Una forma de reducir el costo del bloqueo detallado es buscar sin adquirir ningún bloqueo. Si el método encuentra el nodo buscado, bloquea ese nodo y luego verifica que el nodo no haya cambiado en el intervalo entre el momento en que se inspeccionó y el momento en que se bloqueó.

SINCRONIZACIÓN PEREZOSA

A veces tiene sentido posponer el trabajo duro. Por ejemplo, la tarea de eliminar un nodo de una lista se puede dividir en dos fases: el nodo se elimina lógicamente simplemente configurando un bit de etiqueta y, más tarde, el nodo se puede eliminar físicamente desvinculándolo del resto de los datos.

7.4

7.5

SINCRONIZACIÓN SIN BLOQUEO

A veces podemos eliminar los bloqueos por completo, confiando en operaciones atómicas integradas como compareAndSet() para la sincronización.