Universidad
Rey Juan Carlos

Practica 1:

Memoria Suma Secuencial

PROGRAMACIÓN CONCURRENTE

Sergio Reinoso Barrios | GIC

Índice

DESCRIPCIÓN GENERAL

El programa que he implementado funciona primero realizando una suma secuencial y tras dicha suma, mediante el uso de 2 semáforos y 5 procesos trabajando de forma concurrente, se puede observar como es el procedimiento de la suma concurrente con 5 procesos y muestra el mismo resultado que con la suma secuencial pero en el caso de la suma concurrente el proceso es más rápido y tarda menos tiempo que la suma secuencial ya que la primera suma es con 1 proceso y la segunda suma es con 5 al mismo tiempo.

Los semáforos los he utilizado, uno para controlar el cambio de valores del puntero que me daba el valor de una posición determinada en el array por dicho puntero y el otro semáforo para controlar cuando un proceso no tenía más valores que sumar se bloquee mientras espera a que el resto de procesos termines sus cometidos.

TIPOS DE SINCRONIZACIÓN

He utilizado dos tipos de sincronización en mi código:

- **Exclusión Mutua con Semáforos**

    - Sincronización usada para el cambio del valor del puntero que utilizo para obtener la posición de los números del array. Esta sincronización es necesaria para evitar las condiciones de carrera y así pueda cambiar el valor del puntero sin que se produzca ningún error.

```
// exclusion mutua
CogerDatos.acquire();
puntero_actual = posSumar; // puntero array
posSumar += 2;// apunta al siguiente valor para sumar
// System.out.println(posSumar);
CogerDatos.release();
```

    - También la uso para que cuando vayan a finaliza los procesos por falta de datos que sumar, vayan finalizando 1 a 1 para que la variable de terminados vaya aumentando ordenadamente.

```
// Exclusion mutua para que vayan finalizando los proceso de 1 en 1
CogerDatos.acquire();
println("Esperando a los demás procesos. Han terminado " + terminados + " procesos");

terminados++;

if (terminados < N_PROCESOS) {// si no es el ultimo, desbloqueamos el mutex y bloqueamos el proceso
                              // esperando al ultimo proceso
    CogerDatos.release();
    Barrera.acquire();
} else {
    // si hay 4 procesos ya terminados, el ultimo proceso imprime el nivel
    // y reestablece los contadores y los arrays, por ultimo desbloquea el resto de
    // procesos
}

nivel();
System.out.println("--------------------------------------------------------------------------");
datos = resultados;
if (datos.length > 1) {

    resultados = new int[datos.length / 2];
}

posSumar = 0;

terminados = 0;
nivel++;

for (int i = 0; i < N_PROCESOS - 1; i++) {
    Barrera.release();

}
CogerDatos.release();
```

- **Sincronización en barrera**

    - Esta sincronización es necesaria para poder controlar el bloqueo de los procesos cuando no tienen más valores que sumar y hasta que no acabe el último proceso en realizar su función no desbloquearlos para volver a realizar la misma función en el siguiente nivel con el nuevo array de datos y de resultados.

```
if (terminados < N_PROCESOS) {// si no es el ultimo, desbloqueamos el mutex y bloqueamos el proceso
                              // esperando al ultimo proceso
    CogerDatos.release();
    Barrera.acquire();
} else {
    // si hay 4 procesos ya terminados, el ultimo proceso imprime el nivel
    // y reestablece los contadores y los arrays, por ultimo desbloquea el resto de
    // procesos
}

nivel();
System.out.println("------------------------------------------------------------------------");
datos = resultados;
if (datos.length > 1) {

    resultados = new int[datos.length / 2];
}

posSumar = 0;

terminados = 0;
nivel++;

for (int i = 0; i < N_PROCESOS - 1; i++) {
    Barrera.release();

}
```

SEMÁFOROS

Para la sincronización de los distintos procesos que uso en el programa he utilizado 2 semáforos, CogerDatos y Barrera.

- **CogerDatos** es un semáforo con 1 solo permiso. Este semáforo controla que solo entre 1 proceso a la vez para realizar el cambio de valor del puntero que utilizo para obtener los valores que deseo sumar del array de datos y así que no se produzcan condiciones de carrera en mi programa.

  La razón por la que inicializo el semáforo con 1 permiso es para limitar la entrada de solo 1 proceso a la vez a la sección critica.

- **Barrera** es un semáforo que inicializo sin ningún permiso. Este semáforo lo he utilizado como el método de sincronización en barrera, según van acabando los procesos y no tienen nada más que hacer, se van bloqueando uno a uno y cuando ya el ultimo acaba de hacer su función, se realiza un desbloqueo de todos los procesos que estaban bloqueados para volver a calcular el siguiente nivel.

  La razón de inicializar este semáforo sin ningún permiso es para que se vayan bloqueando los semáforos hasta llegar al último que será el que los desbloquee a ellos.

VARIABLES DE CONTROL

Tras las variables facilitadas en el esqueleto del enunciado con el número de datos, numero de procesos y el array de datos. Decidí añadir el array de resultados con la mitad de la longitud que el array de datos, para que pudiera ir almacenando los resultados de las sumas en él.

Añado 2 semáforos estáticos con la clase SimpleSemaphore que nos ofrece la librería de SimpleConcurrent y en el main los inicializo con los permisos citados anteriormente.

En mi caso, no utilizo ninguna variable booleana, pero si utilizo 3 variables enteras.

- **posSumar** es el puntero de la posición del array que quiero sumar y este si le sumas 1 será el siguiente valor que deseo sumar. Inicializado a cero para controlar la primera posición del array
- **Terminados** es un contador de procesos que utilizo para contabilizar cuando un proceso termina y controlar que no se bloquen todos, para que el último proceso pueda desbloquear al resto. Inicializado a cero para controlar el contador desde cero.
- **Nivel** esta variable controlará el nivel que acabo de terminar. Lo inicializo a 1 para controlar el primer nivel y los respectivos.