Ejercicios Tema 2. Parte 1

Concurrencia con Memoria Compartida

Soluciones

Los ejercicios de esta hoja sirven para que los alumnos puedan ejercitar sus conocimientos en el tema de la sincronización con memoria compartida.

Ejercicio 1. Productor Consumidor

Se desea implementar un programa concurrente con un proceso que produce información (**productor**) y otro proceso que hace uso de esa información (**consumidor**). El proceso **productor** genera un número aleatorio y termina. El proceso **consumidor** muestra por pantalla el número generado y termina.

Ejercicio 2. Productor Consumidor Infinito

Se desea ampliar el programa del Ejercicio 1 de forma que el proceso **productor** esté constantemente produciendo números consecutivos. El proceso **consumidor** estará constantemente consumiendo los productos. No se puede quedar ningún producto sin consumir. No se puede consumir dos veces el mismo producto.

Ejercicio 3. Cliente Servidor 1 Petición

Programa formado por un **proceso servidor** y otro **proceso cliente.** El **proceso cliente** hace una petición al **proceso servidor** (en forma de número aleatorio) y espera su respuesta, cuando la recibe, la procesa. El **proceso servidor** no hace nada hasta que recibe una petición, momento en el que suma 1 al número enviado en la petición y contesta con ese valor. El **proceso cliente** procesa la respuesta mostrándola por pantalla.

Ejercicio 4. Cliente Servidor N Peticiones

Se desea ampliar el programa anterior de forma que el **proceso cliente** esté constantemente haciendo peticiones y el **proceso servidor** atendiendo a las mismas.

Ejercicio 5. Museo

Existen 3 personas en el mundo, 1 museo, y sólo cabe una persona dentro del museo. Las personas realizan cuatro acciones dentro del museo:

- Cuando entran al museo saludan: "hola!"
- Cuando ven el museo se sorprenden: "qué bonito!" y "alucinante!"
- Cuando se van del museo se despiden: "adios"
- Cuando salen del museo se van a dar un "paseo"

Después del paseo, les ha gustado tanto que vuelven a entrar. Se pide:

- Número de métodos diferentes que ejecutarán los procesos
- Número de procesos del programa concurrente y de qué tipo son
- Escribir lo que hace cada proceso
- Identificar los recursos compartidos
- Identificar la sección o secciones bajo exclusión mutua
- Escribir el programa completo

Ejercicio 6. Museo con infinitas personas

Considerar que caben infinitas personas dentro del museo. Cada persona al entrar tiene que saludar diciendo cuántas personas hay en el museo: "hola, somos 3". Al despedirse tiene que decir el número de personas que quedan tras irse: "adiós a los 2".

Ejercicio 7. Museo con regalo

Para incentivar las visitas, cuando una persona entre en el museo estando vacío, será obsequiado con un regalo. Las personas, después de saludar, dicen si les han dado un regalo ("Tengo regalo") o si no ("No tengo regalo"). Las personas deben permitir que otras personas saluden entre su saludo y el comentario sobre el regalo.

Ejercicio 8. Preguntas Cortas

8.1) Dado el programa siguiente ¿podría darse el caso de que uno o ninguno de los dos procesos finalizase? Razona la respuesta.

```
package ejercicio;

import static es.sidelab.sc.SimpleConcurrent.*;

public class Ejer8_1 {

    static volatile boolean fin = false;

    public static void proceso() {
        int i = 0;            // I1
        while (!fin) {        // I2
            i++;              // I3
            if (i == 3) {     // I4
                fin = true;   // I5
            } else {
                fin = false; // I6
            }
        }
    }

    public static void main(String[] args) {
        createThreads(2, "proceso");
        startThreadsAndWait();
    }
}
```

Concurrencia con Memoria Compartida

8.2) Utilizando la espera activa se pretende resolver el problema de sincronización condicional de dos procesos concurrentes cuyo ciclo de vida es el siguiente:

```
package ejercicio;

import static es.sidelab.sc.SimpleConcurrent.*;

public class Ejer8_2 {

    static volatile boolean flag = false;

    public static void p1(){

        while(true){
            print("A");
            flag = true;
            print("B");
        }
    }

    public static void p2(){
        while(true){
            print("C");
            while(!flag);
            print("D");
        }
    }

    public static void main(String[] args) {
        createThread("p1");
        createThread("p2");
        startThreadsAndWait();
    }
}
```

a) Después de la primera sincronización ¿se sincronizarán correctamente ambos procesos? Razone la respuesta.

b) Qué tipo de interacción se produce entre los procesos P1 y P2? ¿En qué estados pueden estar los procesos P1 y P2?

8.3) ¿Por qué razón el acceso (lecturas y/o escrituras) a variables globales compartidas por varios procesos debe realizarse bajo exclusión mutua en el modelo de variables compartidas?

8.4) ¿En qué consiste el problema de la exclusión mutua? ¿Cuáles son los requisitos de su solución?

8.5) Dibuje el diagrama de estados de un proceso

8.6) ¿Qué diferencia hay entre los estados de preparado y de ejecución de un proceso?

8.7) Defina en qué consisten las propiedades de seguridad de un programa concurrente. Cite dos propiedades de este tipo.

8.8) ¿Qué es un interbloqueo de procesos?

8.9) ¿Cuáles son las dos actividades principales necesarias para gestionar la ejecución de procesos concurrentes en multiprogramación?

8.10) En la gestión de procesos concurrentes, explique la diferencia entre planificación y despacho.

Ejercicio 9. Downloader

a) Se quiere implementar una aplicación para descargar ficheros. La aplicación debe tener la capacidad de descargar un único fichero, pero debe ser capaz de descargar varios fragmentos del fichero de manera concurrente para aprovechar de forma más eficiente la red.

Para simplificar la aplicación, consideramos que un fichero se representa en memoria como un array de enteros. Internamente, la aplicación dispone de una serie de procesos que van descargando los diferentes fragmentos del fichero (posiciones del array). Los procesos están ejecutando tres acciones: primero se determina el siguiente fragmento a descargar, a continuación se descarga ese fragmento, y por último se guarda el fragmento descargado en el array que representa el fichero.

La solución se puede implementar con espera activa o con semáforos.

La descarga de los distintos fragmentos se simulará con el método:

```
private static int descargaDatos(int numFragmento) {
    sleepRandom(1000);
    return numFragmento * 2;
}
```

Cuando se ha terminado de descargar completamente el fichero, se muestra por pantalla y el programa termina. Para mostrar el fichero por pantalla se utilizará el siguiente método:

```
private static void mostrarFichero() {
    println("------------------------------------------------");
    print("File = [");
    for (int i = 0; i < N_FRAGMENTOS; i++) {
        print(fichero[i] + ",");
    }
    println("]");
}
```

A continuación se presenta un esqueleto de la aplicación de descargas:

```
package ejercicio;

import static es.sidelab.sc.SimpleConcurrent.*;

public class Ejer_9A_Downloader_Plantilla {

    private static final int N_FRAGMENTOS = 10;
    private static final int N_HILOS = 3;

    private static volatile int[] fichero = new int[N_FRAGMENTOS];

    //Add the attributes you need

    private static int descargaDatos(int numFragmento) {
        sleepRandom(1000);
        return numFragmento * 2;
    }
```

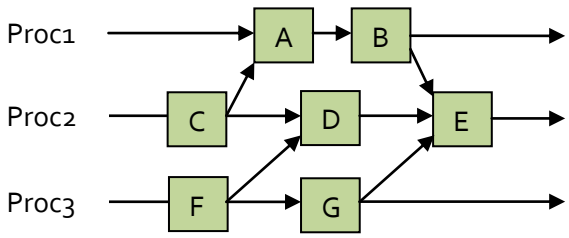
```
    private static void mostrarFichero() {
        println("-------------------------------------------------");
        print("File = [");
        for (int i = 0; i < N_FRAGMENTOS; i++) {
            print(fichero[i] + ",");
        }
        println("]");
    }

    public static void downloader() {

        // Mientras hay fragmentos que descargar...
            //Descargar los datos del siguiente fragmento
            //Almacenar los datos en el array
    }

    public static void main(String[] args) {

        createThreads(N_HILOS, "downloader");

        startThreadsAndWait();

        mostrarFichero();
    }
}
```

b) En la plantilla anterior, la impresión del fichero se realiza en el hilo principal. Se pide implementar el mismo programa de descarga de ficheros pero esta vez la impresión será realizada por el último proceso que guarde un fragmento descargado en el fichero.

Ejercicio 10. Sincronización Condicional

Implementar un programa concurrente en Java con SimpleConcurrent que tenga los siguientes requisitos:

- El programa consta de 3 procesos.
- Cada proceso escribe por pantalla varias letras y termina. El proceso 1 debe escribir la letra 'A' y la letra 'B'. El proceso 2 debe escribir la letra 'C', la letra 'D' y la letra 'E'. El proceso 3 debe escribir la letra 'F' y la letra 'G'.
- Los procesos deben sincronizarse para que se cumplan las siguientes relaciones de precedencia:



- La sincronización condicional se deberá implementar con espera activa.

- Algunas de las posibles salidas de este programa son:
  - **CFADBGE**
  - **CFDABGE**
  - **CFDAGBE**
  - **FCDAGBE**
- Algunas de las posibles salidas inválidas de este programa son:
  - **ACFDBGE**
  - **CDFABGE**

Ejercicio 11. Cliente-Servidor con Proxy (1 petición)

Implementar un programa concurrente formado por un proceso Servidor, un proceso Proxy y un proceso Cliente.

- El proceso Cliente genera un número aleatorio, se lo envía al Proxy y espera su respuesta. Cuando la recibe, la imprime por pantalla y termina.
- El proceso Proxy no hace nada hasta que recibe una petición. Cuando la recibe, el Proxy suma 1 al número enviado por el cliente y hace una petición al servidor con ese número, esperando su respuesta. Cuando el proceso Servidor reciba la petición, le suma 1 y se la envía de vuelta al proxy. Cuando el proxy recibe la petición del servidor, se la reenvía al cliente.
- La solución deberá implementarse con espera activa.

Ejercicio 12. Cliente-Servidor con Proxy (N peticiones)

Implementar un programa concurrente similar al anterior con la diferencia de que el proceso Cliente hace 10 peticiones, el proceso Proxy atendiendo esas 10 peticiones y se las redirige al proceso Servidor. El proceso servidor también responde a 10 peticiones. La solución deberá implementarse con espera activa.