Tecnológico Nacional de México

Instituto Tecnológico De Culiacán

Carrera:

Ingeniería en Sistemas Computacionales

Materia:

Tópicos Avanzados de Programación

Unidad 3. Programación Concurrente

Síntesis. Programación Concurrente

Alumno:

Eliu Gastelum Gurrola

Numero de control:

19170587

Índice

Concepto de hilo

Según la investigación que llevé a cabo entendí que un **hilo** o **hebra** o tambien su nombre en inglés **thread** es una secuencia de instrucciones (subproceso o proceso ligero) que estan en estado de ejecución dentro las circunstancias o contexto del proceso, la razón que estos hilos no pueden ejecutarse por su cuenta es porque necesitan la supervisión del proceso el cual es un programa que se ejecuta en su espacio de direcciones.

En cuanto a la maquina virtual de java este proporciona o da soporte a los hilos a través de interfaz y conjuntos de clases las cuales proporcionan tambien una serie de funcionalidades las cuales son:

- *Thread*
- *Runnable*
- *ThreadDeath*
- *ThreadGroup*
- *Object*

Tambien la **máquina virtual** de java es capaz de manejar multihilos que en palabras mas simples seria tener la capacidad de crear muchos hilos los cuales son flujos de ejecución al mismo tiempo a los cuales esta máquina virtual se encarga de administrar detalles minuciosos como lo son el asignar los tiempos de ejecución de los hilos o tambien como sus prioridades las cuales van del 1 al 10, esto último que se dijo sobre los factores que administra se asimila mucho a como un sistema operativo administra múltiples procesos.

El **planificador** o **scheduler** es aquel que da soporte, es el que se encarga de controlar todos los hilos que se estan ejecutando en todos nuestros programas, tambien cuales deben de encontrarse y por último cuando deben de ejecutarse y esto ultimo esta relacionado con la prioridad que habia mencionado previamente. La prioridad se asigna en el rango de números del 1 al 10 de forma en que mientras mas grande sea el numero que se le asigna al hilo mayor prioridad tiene este. Otro dato importante acerca de la prioridad se trata de que el hilo puede heredar la prioridad que el hilo padre tiene. Mas adelante sabremos más sobre algunos métodos del planificador que ayudan a controlar los hilos.

Un **ejemplo** para que quede más claro el uso es de estos y luego dar paso para hablar más profundamente acerca de los hilos. Creo yo que un ejemplo sencillo de entender es el uso de hilos en aplicacion Word donde un hilo “background” está checando la gramática continuamente a la vez que uno está escribiendo, mientras tanto otro hilo está guardando periódicamente los cambios automáticamente en el que se está trabajando.

Saber con lo que cuenta un hilo tambien son un elemento muy importante que debe de conocer, estos cuentan con:

- Un estado
- Contexto del procesador. El cual es muy útil al momento en que se debe de reanudar un hilo en donde hubo una interrupción con anterioridad y asi será más fácil saber en donde se debe de continuar la ejecución del hilo.
- Pila de ejecución. Lugar donde se almacenarán todas las instrucciones que serán ejecutadas.
- Espacio de almacenamiento estático. Aquí es donde se guardarán todas las variables.
- Acceso a los recursos de la tarea. Los cuales los hilos se encargan de compartir usados en el proceso.

Algunas ventajas acerca de los hilos que podemos notar cuando hacemos uso de esto:

- Son mas eficientes ya que se tardan menos tiempo en crear un hilo en un programa existente que crear otro nuevo proceso.
- Relacionado a la primera ventaja tambien se tarda menos tiempo en terminar un hilo que un proceso
- Es más sencilla la comunicación entre los hilos que entre diferentes procesos.
- Otro punto a favor seria la capacidad de reutilizar los hilos para evitar la necesidad de crear mas hilos y asi seria mas barato para la memoria.
- Los ejecutores se encargan de la gestión del procesamiento de las tareas como la creación de hilos y el uso del start y join.
- Multihilos son importantes para las aplicaciones cliente – servidor.
- Agilizan los tiempos de retraso de la comunicación cliente – servidor.

Los estados o también conocido como ciclo de vida es el campo de acción de un hilo el cual lo compone la etapa runnable esto quiere decir cuando el subproceso se está ejecutando en donde después de esta etapa hay otras etapas a los que se pueden se asignados a los hilos según lo permitas estos son:

- Estado **new**:

  Este estado es el que tiene el hilo cuando recién fue creado o inicializado y está esperando la llamada de la siguiente etapa para empezar a trabajar la cual es el método start, mientras tanto estos no harán absolutamente nada hasta que se les de la señal.

- Estado **runnable**:

  Este estado llega o se le es asignado cuando al hilo se le da la señal que antes mencionamos la cual es start y este invoca al metodo run y es cuando el hilo se encuentra en ejecución y está trabajando.

- Estado **not runnable**:

  En este estado los hilos son detenidos por métodos que se verán mas adelante pero este estado quiere decir que el metodo no hará nada momentáneamente hasta que se vuelva a habilitar.

- Estado **dead**:

  Un hilo entra en este estado cuando ya no es necesitado, en este estado ya no se puede volver a iniciar y solo puede entrar a este estado por dos causas la primera es cuando el metodo run termina su ejecución y la segunda pasa cuando a este se le realiza una llama al metodo stop.

Comparación de un programa de flujo único contra uno de flujo múltiple.

Primero comenzaré explicando lo que son estos dos distintos tipos de programas lo cual para el primero será bastante sencilla ya que si no lo saben ya lo he y han utilizado antes.

Por lo que leí un **programa de flujo único** se trata de un programa de una tarea se utiliza un único flujo de ejecución es decir un solo hilo en donde este se encarga de hacer todo el trabajo secuencialmente ya que en varios programas de distintos contextos no veremos la necesidad de emplear múltiples tareas que trabajen por separado es decir no es necesario crear mas hilos mas que el que ya tenemos cuando iniciamos nuestro proyecto, como ejemplo esta el caso del main como se ve en el ejemplo.

Ejemplo:

```
public class Ejemplo{
        public static void main (string args[]){
                System.out.println( " Hola Mundo " );
        }
}
```

Hablando del ejemplo que se muestra en la parte superior podemos observar que cuando se llama al main el programa imprime un mensaje “Hola Mundo” en consola para luego terminar su ejecución. El hilo inicia y entra al estado dead cuando termina su ejecución.

En cuanto a programa de flujo múltiple esto seria todo lo contrario y ya con lo que se ha explicado con anterioridad creo yo que ya se tendría alguna idea llegados a este punto.

Un **programa de flujo múltiple** se trata de realizar distintas tareas al mismo tiempo, ya que un programa multihilos permite que los hilos comiencen y termine lo mas pronto que sea posible en donde cada hilo controla diferentes aspectos dentro del programa puede supervisar la entra de alguna información como la de un periférico o estar haciendo otros trabajos como se vio en el ejemplo del Word donde se estaba checando periódicamente si la gramática era correcta. Todos los hilos comparten recursos, cada uno tiene sus propios códigos y datos.

Aquí podemos ver un ejemplo hacer de como se compone un programa de este estilo:

```
class TestTh extends Thread {
        private String nombre;
        private int retardo;

           // Constructor para almacenar nuestro nombre
           // y el retardo
           public TestTh( String s,int d ) {
              nombre = s;
              retardo = d;
              }

           // El metodo run() es similar al main(), pero para
           // threads. Cuando run() termina el thread muere
           public void run() {
              // Retasamos la ejecución el tiempo especificado
              try {
                 sleep( retardo );
              } catch( InterruptedException e ) {
                 ;
              }

              // Ahora imprimimos el nombre
              System.out.println( "Hola Mundo! "+nombre+" "+retardo );
              }
           }
        public class MultiHola {


              public static void main( String args[] ) {
              TestTh t1,t2,t3;

              // Creamos los threads
              t1 = new TestTh( "Thread 1",(int)(Math.random()*2000) );
              t2 = new TestTh( "Thread 2",(int)(Math.random()*2000) );
              t3 = new TestTh( "Thread 3",(int)(Math.random()*2000) );

               // Arrancamos los threads
              t1.start();
              t2.start();
              t3.start();
              }

        }
}
```

Creación y control de hilos

A la hora de crear hilos existen dos maneras:

- Heredar de la clase **Thread**
- Utilizar la interfaz **Runnable**

En cualquiera de los dos que selecciones tienes que definir el metodo run() el cual será el que tenga escrito el codigo que quieres que trabaje cada hilo, en cuanto se entré a este metodo este entrara en ejecución y terminara su ejecución cuando se termine el codigo del metodo run().

Un ejemplo utilizando la clase Thread seria:

```
public class EjemploHilo extends Thread
{
   public void run()()
   {
      // Código del hilo
   }
}
```

Una vez se escriba el codigo que queremos que haga se debe de instanciar con el siguiente ejemplo:

```
Thread t = new Thread(new EjemploHilo());
t.start();
```

Entonces cuando se ejecute el programa se creara el hilo y se iniciara cuando llegue a “t.start()” en donde empezara a ejecutar el codigo que este en el metodo run(), en cuanto a un punto que se debe de tener en cuenta es una inconveniencia es que no se podrá heredar ninguna otra clase y por ende un hilo no podrá heredar de alguna otra clase.

Es por esta razón que es más beneficioso usar la interfaz Runnable a la hora de implementar hilos y aparte en este caso si se podrán implementar mas interfaces aparte de la interfaz para hilos, aquí tenemos un ejemplo de cómo se crean:

```
public class EjemploHilo implements Runnable
{
   public void run()
   {
      // Código del hilo
   }
}
```

Al momento de instanciar e iniciar el hilo se puede hacer casi lo mismo:

```
Thread t = new Thread(new EjemploHilo());
t.start();
```

Aquí lo que se hizo es lo mismo solo implementamos la interfaz a la clase, luego creamos el metodo run() para agregar el codigo que queremos que se ejecute cuando pongamos la instrucción start() al hilo, y en el ultimo instanciamos el objeto para después iniciarlo y haga su función.

Al momento de meternos con el control de los hilos debemos recordar antes cuando estábamos hablando del planificador habia dicho que hablaríamos un poco más sobre la prioridad de los hilos y es con lo que vamos a comenzar, hay un metodo que se utiliza para asignar la prioridad en el cual es el setPriority() al que se le debe de proporcionar un valor del uno al diez aunque tambien hay unas palabras reservadas para esto que tambien se le pueden asignar las que son MIN_PRIORITY que asigna el uno como prioridad el cual es el mas bajo entre todos y tambien esta la otra palabra reservada MAX_PRIORITY el que se encarga de asignar la mayor de todas las prioridades para que se ejecute primero, para terminar hay otro metodo el cual es yield() que se encarga de sacar del CPU cuando un hilo con alta prioridad se vuelva ejecutable.

Tambien comentamos que hablaríamos más acerca de cuándo un hilo está en estado not runnable este tambien se puede regresar al estado runnable de diferentes formas como:

- Si el hilo esta suspendido con suspend(), se puede utilizar el metodo resume().

- Si el hilo esta durmiendo con el metodo , solo se mantendrá unos milisegundos asi para luego regresar a su estado runnable.

- Si el hilo esta en espera este volverá a hacer su funcion cuando reciba una llamada con el metodo notify() o notifyAll().

- Si un hilo esta bloqueado por I/O este regresara a su estado cuando se complete la actividad que le dejaron haciendo.

Aparte de estos métodos de control tambien se encuentran otros que no se mencionaron uno de ellos es el stop() el cual permite que se detenga la ejecución del mismo, y hay otro que se llama isAlive() el cual te regresa un true si el hilo se esta ejecutando, tambien existen mas pero esos van en la parte de sincronización.

Este es un ejemplo de como de los métodos de control:

```
class java1003 {
   static public void main( String args[] ) {
      // Se instancian dos nuevos objetos Thread
      Thread hiloA = new MiHilo();
      Thread hiloB = new MiHilo();

      // Se arrancan los dos hilos, para que comiencen su ejecución
      hiloA.start();
      hiloB.start();

      // Aquí se retrasa la ejecución un segundo y se captura la
      // posible excepción que genera el método, aunque no se hace
      // nada en el caso de que se produzca
      try {
         Thread.currentThread().sleep( 1000 );
      }catch( InterruptedException e ){}

      // Presenta información acerca del Thread o hilo principal
      // del programa
      System.out.println( Thread.currentThread() );

      // Se detiene la ejecución de los dos hilos
      hiloA.stop();
      hiloB.stop();
      }
   }

class MiHilo extends Thread {
   public void run() {
      // Presenta en pantalla información sobre este hilo en particular
      System.out.println( Thread.currentThread() );
      }
   }
```

Aquí podemos ver que se inician cuando ya estan instanciados con el metodo start(), mas adelante los duerme con por 1000 milisegundos con el metodo Sleep() por ultimo podemos ver como los detiene con el metodo stop().

Sincronización de hilos

Este punto es una parte muy importante para los hilos ya que es muy utilizado para que los hilos que se han creado necesitan acceso a un recurso compartido y quieres que solamente pueda ser utilizado por un solo hilo a la vez, como por ejemplo cuando un hilo esta escribiendo un archivo, se debe evitar que el otro haga lo mismo hasta que lo desocupe como lo vimos en clase con lo del programa productor – consumidor.

Aquí se pueden ver dos que he me han enseñado en clase los cuales son por medio de un modificador en un metodo y otro que se llama por bloques vigilados.

El primer metodo de sincronización es el **synchronized** que permitirá evitar que mas de un hilo entre en el metodo para un objeto determinado, en pocas palabras evita que entren dos hilos al mismo tiempo, este metodo de sincronización se programa de la siguiente forma:

```
public synchronized void metodo_seccion_critica()
{
   // Código sección crítica
}
```

Como tambien se puede escribir de la siguiente manera:

```
synchronized(objeto_con_cerrojo)
{
  // Código sección crítica
}
```

Aquí tambien hay métodos de control que modifican los estados de los hilos como lo son el wait() que se usa para bloquear los hilos y para desbloquear estos utilizamos es que mencione antes llamado notify() que desbloqueara algun hilo bloqueado aleatoriamente y notifyAll() que desbloqueará todo los hilos que estén bloqueados. Hay casos en los que algun metodo se necesite bloquear hasta que haya terminado asi que para esto es este último metodo llamado join().

Hay otro metodo que se parece a este ya que una de igual manera el synchronized y es llamado **sincronización reentrante** que sirve para contener una seccion sincronizada dentro de una seccion sincronizada pero no causa ningún bloqueo, como lo podemos ver en el siguiente ejemplo.

Ejemplo:

```
class Reentrant {
   public synchronized void a() {
      b();
      System.out.println(" estoy en a() ");
   }
   public synchronized void b() {
      System.out.println(" estoy en b() ");
   }
}
```

Los **bloques vigilados** son el otro metodo de sincronización en donde estos tienen una condición que hasta que no se cumpla no se tiene que ejecutar el bloque, esto puede ser ineficiente porque se estará usando la CPU mientras el hilo se encuentra en espera, esto se codifica de la siguiente manera:

```
public synchronized bloqueVigilado() {
   while(!condicion) {
      try {
         wait(); // desocupa la CPU
      } catch (InterruptedException e) {}
   }
   System.out.println("La condición se ha cumplido");
}
```

En caso de que un hilo quiera entrar se bloqueará si se está ocupando el recurso compartido ya que estará el wait y se saldrá hasta que se le notifique que ya puede hacer su parte.