Unidad IV: Programación concurrente (MultiHilos)

4.1. Concepto de hilo

Hilo (theread) llamado también proceso ligero o subproceso, es la unidad de ejecución de un proceso y esta asociado con una secuencia de instrucciones un conjunto de registros y una pila. Cuando se crea un proceso el S.O crea su primer hilo (hilo primario) en la cual puede,a su vez,crear hilos adicionales. esto pone de manifiesto que un proceso no ejecuta,si no que es sólo el espacio de direcciones donde recide el código que es ejecutado mediante uno o más hilos.

En un S.O tradicional. Cada proceso tiene un espacio de direcciones y un único hilo de control por ejemplo, considere que un programa que incluya la siguiente secuencia de operaciones para actualizar el saldo de una cuenta bancaria cuando se efectúa un nuevo ingreso:

```
Saldo=Cuenta.ObtenerSaldo ();
Saldo+=ingreso;
Cuenta.EstablecerSaldo (saldo);
```

Este modelo de programación, en el que se ejecuta un solo hilo, es en el que estamos acostumbrados a trabajar habitualmente. Pero, siguiendo con el ejemplo anterior piensa en un banco real; en el varios cajeros pueden actuar simultáneamente. Ejecutar el mismo programa por cada uno de los cajeros tiene un costo elevado (recuerde los recursos que necesita). En cambio si el programa permitiera lanzar un hilo por cada petición de cada cajero para actualizar una cuenta, estaríamos en el caso múltiples hilos ejecutandose concurrente mente (multitriandi) . Esta característica ya es una realidad en los S.O modernos de hoy y como consecuencia contemplada en los lenguajes de programación actuales.

Cada hilo se ejecuta de forma estrictamente secuencial y tiene su propia pila en estados de los registros de UCP y su propio contador de un programa en cambio comparte el mismo espacio de direcciones, lo que significa compartir también las

mismas variables globales, el mismo conjunto de ficheros abiertos, procesos hijos (no hilos hijos), señales, semáforos, etc.
Los hilos comparten la UCP de la misma forma que lo hacen los procesos pueden crear hilos hijos y se pueden bloquear no obstante mientras un hilo del mismo proceso, en el caso de hilos soportados por el kernel (núcleo del S.O: programas en ejecución que hacen que el sistema funcione), no sucediendo lo mismo con los hilos soportados por una aplicación por ejemplo en Windows NT todos los hilos son soportados por el Kernel; imaginemos que alguien llaga a un cajero para depositar dinero en una cuenta y casi al mismo tiempo un segundo cliente realiza la misma operación sobre la misma cuanta en el segundo cajero quedara bloqueado asta que el registro que esta siendo actualizado por el primer cajero quede liberado.

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

4.3. Creación y control de hilos

Los hilos de java se pueden crear en dos formas: escribiendo una nueva clase derivada de Theread, o bien haciendo una clase existente implementa la interfaz Runnable .

La clase Thread,que implemente la interfaz Runnable,de forma resimida,está definida asi:

```
public class Thread extends Object implements Runnable

{

//Atributos
```

static int MAX_PRIORITY;

//Prioridad máxima que un hilo puede tener.

static int MIN_PRIORITY;

//Prioridad mínima que un hilo puede tener.

static int NORM_PRIORITY;

//Prioridad asignada por omisión a un hilo.

//Constructores

Thread ([argumentos])

//Métodos

static Theread currentThread()

//Devuelve una referencia al hilo que actualmente esta en ejecución.

long getID()

//Devuelve el identificador del hilo.

String getName()

//Devuelve el nombre del hilo.

int getPriority()

//Devuelve la prioridad del hilo.

void interrupt()

//Envía este hilo al estado de preparado.

boolean isAlive()

//Verifica sí este hilo está vivo (no ha terminado).

boolean isDaemon()

//Verifica sí este hilo es un demonio. Se da este nombre a

//un hilo se ejecuta en segundo plano, realizando una

//operación específica en tiempos predefinidos, o bien en

//respuestas a ciertos eventos.

boolean is Interrupted()

//Verifica si este hilo ha sido interrumpido.
void join([milisegundos[. nanosegundos]])
//Espera indefinidamente o el tiempo especificado,a que este
//hilo termine (a que muera).

void run()

4.4. Sincronización de hilos

Cuando se están utilizando hilos múltiples, algunas veces es necesario coordinar las actividades de dos o más. El proceso por el cual se logra esto se llama *sincronización.* La razón más común para la sincronización es cuando dos o más hilos necesitan acceso a un recurso compartido que sólo puede ser utilizado por un hilo a la vez. Otra razón para la sincronización es cuando un hilo está esperando un evento causado por otro hilo. En este caso, debe de haber algún medio por el cual el primer hilo se mantenga en estado suspendido hasta que el evento ocurra.

La sincronización esta soportada por la palabra clave ***synchronized*** y por unos cuantos métodos bien definidos que tienen todos los objetos.