Debemos entender la ejecución de un proceso como **la ejecución simultánea** de los n hilos del proceso. **Un proceso es un árbol n_ario de hilos. El nodo raíz de ese árbol es el hilo primario**.

Por tanto, **un proceso es al menos un hilo de ejecución; el hilo primario.** En cada instante de la vida del proceso será un conjunto (un árbol n-ario) de n hilos que se ejecutarán simultáneamente obteniendo así **multitarea o paralelismo real: En un instante dado del proceso existen n instrucciones en ejecución, una instrucción de cada hilo que será atendida en una unidad de control.**

Para posibilitar la multitarea o paralelismo real necesitamos dos tipos de requerimientos:

a) De Hardware: Disponer de un procesador de varios núcleos.
b) De Software: .. Disponer de un sistema operativo multitarea

.. El proceso en estado de ejecución es una aplicación diseñada como un árbol de n hilos. Una aplicación multihilo**.**

Cada hilo es una secuencia de código que se ejecuta dentro del contexto de un proceso. Cada hilo tiene su propio estado y sus valores de registros de unidad de control. Los hilos de un proceso se ejecutan dentro del espacio de direcciones de memoria donde se ejecuta el proceso, por tanto, comparten la misma imagen de memoria, comparten todos los recursos del proceso

(objetos,… buffers , ficheros , etc..) .

Podrán acceder a los mismos objetos en memoria.

Cuando el proceso al que pertenecen se restaura y pasa a estado de ejecución entonces todos los hilos vivos del proceso se restauran, uno en cada unidad de control.

Cuando el proceso al que pertenecen deja de estar en ejecución y se salvaguarda su estado, se salvaguarda el estado de todos y cada uno de los hilos del proceso.

Ahora bien, en un instante dado el proceso en ejecución puede ser un conjunto de n hilos, donde n es mayor que la cantidad de unidades de control disponibles en el equipo, entonces aparece la ejecución simultánea de x hilos **y ejecución concurrente de los n hilos del proceso.**

Esta ejecución concurrente de los hilos del proceso es suministrada por el planificador de hilos del sistema operativo.

El conjunto de políticas de planificación nos proporciona la planificación de hilos y por tanto, su ejecución concurrente al igual que ocurría con la planificación de procesos.

PLANIFICACIÓN DE HILOS DENTRO DEL CONTEXTO DE UN PROCESO: Ejecución de múltiples hilos en cierto orden.

La ejecución de todas estas rutinas que constituyen el planificador de hilos nos proporcionan un **Algoritmo de planificación.**

El planificador de procesos del sistema operativo determina **qué proceso** es el que se ejecuta, está en estado de ejecución en un determinado momento en el procesador, (un proceso y sólo uno). Dentro de ese proceso, el hilo/ los hilos que se ejecutarán al mismo tiempo y concurrentemente estarán en función del número y tipo de núcleos disponibles y del algoritmo de planificación empleado.

PLANIFICACIÓN DETERMINISTA: Planificación por prioridad.

- **La máquina Java soporta un algoritmo de Planificación Determinista y Apropiativo**.

En caso de no disponibilidad de núcleos (UC) para la ejecución de todos los hilos del proceso en ejecución, el hilo **preparado** que se elige para su ejecución es el de **prioridad más alta**. Cada hilo tiene asignada una prioridad que está definida por un valor numérico comprendido entre MIN_PRIORITY y MAX_PRIORITY, habitualmente valores en rango [1, 10]. Cuando varios hilos están en estado preparado, será ejecutado el de mayor prioridad y se apropia de una CPU, sólo cuando la ejecución de este hilo se detenga podrá ser ejecutado otro de menor prioridad. Del mismo modo, cuando un hilo con prioridad más alta que el que está en ejecución pasa al estado de preparado, se ejecutará automáticamente. **Este algoritmo supone que la máquina java no reemplazará el hilo actual en ejecución por otro de igual prioridad.**

PLANIFICACIÓN NO DETERMINISTA: Tiempo Compartido: Asignación de un cuánto de ejecución.

Cuando los hilos del proceso en ejecución tienen la misma prioridad, entonces el planificador de hilos será el que asigne, dentro del **conjunto de hilos preparados de igual prioridad tratados como una cola FIFO,** una unidad de control para su ejecución utilizando **tiempo compartido**. Asigna un **cuánto de ejecución** a cada hilo. Esto siempre y cuando el sistema operativo subyacente a la máquina java lo permita. Las plataformas Windows y Macintosh admiten planificación por cuantos para hilos de igual prioridad. Solaris aplica una **planificación cooperativa**, el hilo debe ceder voluntariamente la ucp **(yield**).

HILOS DEMONIOS:

Los hilos Demonios son utilizados para prestar servicios en segundo plano a todos los procesos (hilos del proceso al que pertenecen) que lo necesiten.

Para crear un hilo demonio creamos un hilo normal pero ejecutamos para él el método setDaemon.

```
hilo.setDaemon(true);
```

Todos los hilos creados por un hilo demonio son hilos demonio.

Para saber si un hilo es un hilo demonio ejecutamos el método isDaemon. El método devolverá true si el hilo es un hilo demonio y false en caso contrario.

```
Boolean b=hilo.isdaemon();
```

El intérpetre Java finalizará su ejecución cuando todos los hilos del proceso que queden en ejecución (vivos) sean hilos demonios.

El método run() del hilo puede ser un ciclo sin fin.

PRIORIDADES DE LOS HILOS:

Cuando se crea un hilo éste nace con una prioridad por defecto que es la misma que tiene el hilo que lo creó, el hilo padre en el árbol de hilos.

Para modificar la prioridad de un hilo utilizaremos el método **setPriority**

Si no modificamos la prioridad de los hilos que creamos, el valor por defecto de cualquier hilo es NORM_PRIORITY de valor 5.

Cualquier hilo de un proceso tiene definido por defecto su prioridad: Es la prioridad heredada del su hilo padre. El hilo primario nace con prioridad normal, valor 5, por tanto, todos los hilos de un proceso tienen por defecto prioridad normal.

Las constantes relativas a las prioridades definidas en la clase Thread son:

Constante	*Valor*
MIN_PRIORITY	1
NORM_PRIORITY	5 (valor por omisión)
MAX_PRIORITY	10

```
hilo.setPriority(nuevaprioridad);
```

nuevaprioridad valor entero comprendido entre 1 y 10.

Para verificar la prioridad de un hilo, utilizaremos el método **getPriority.**

```
int n=hilo.getPriority();
```

HILOS INDEPENDIENTES:

Cada hilo contiene todo lo que necesita para su ejecución: datos y métodos. **Cada hilo se ejecuta concurrentemente con los demás hilos del proceso sin interferir en la ejecución de los otros hilos**. CONCEPTO DE EJECUCIÓN ASINCRÓNICA

Los hilos independientes de un proceso se ejecutan de forma simultánea/concurrente y asincrónica.

Las políticas de planificación de la ejecución concurrente de los distintos hilos lanzados en el contexto de un mismo proceso, que son llevadas a cabo por el planificador del sistema operativo subyacente y por la máquina virtual java son suficientes.

HILOS COOPERANTES:

Los hilos se ejecutan simultáneamente/concurrentemente pero además van a acceder a los mismos recursos y/o datos. **Los hilos cooperantes comparten uno o varios objetos .**Por ejemplo el mismo fichero, un objeto de datos, etc.**.**

En este caso, las políticas de planificación de la máquina Java y Sistema Operativo no son suficientes.

El programador debe tomar el control y asegurar que cada hilo accede a los recursos (compartidos) de una manera **sincronizad**a. → **Esto implica que se hacen necesarias operaciones de sincronización**. Aparece concepto de ejecución sincrónica y atómica.

Los hilos independientes se ejecutan simultáneamente/concurrentemente y asincrónicamente.

Los hilos cooperantes se ejecutan simultáneamente/concurrentemente y sincrónicamente

Elementos de sincronización o políticas de sincronización diseñadas por el programador para los hilos cooperantes:

A) **DEFINICIÓN DE SECCIONES CRÍTICAS PARA SINCRONIZAR HILOS COOPERANTES.**

B) **ADEMÁS USO DE LOS MÉTODOS WAIT() Y NOTIFY(), NOTIFY ALL() . Estos métodos son heredados de la clase object. PARA ADEMÁS DE SINCRONIZAR Y SECUENCIAR HILOS COOPERANTES.**

SECCIÓN CRÍTICA: Sección de código que en un instante determinado accede **exclusivamente a un objeto** de datos que es compartido por varios hilos cooperantes.

Cuando un hilo está ejecutando una sección crítica de un objeto ningún otro hilo puede ejecutar una sección crítica de ese objeto. **El acceso al objeto para ejecutar una sección crítica es exclusivo y atómico**

A) **PARA DEFINIR UNA SECCIÓN CRÍTICA, DISPONEMOS DE DOS FORMAS EN EL CÓDIGO FUENTE**:

A.1) Primera forma: Agrupar el código identificado (definido) como crítico en **un método** de la clase a la que pertenecerá el objeto, declarando en la cabecera del método **synchronized (sincronizado).**

Un hilo no podrá acceder al **método synchronized** cuando otro hilo lo está ejecutando. El mecanismo funciona así:

Cuando un hilo quiera ejecutar un método sincronizado de un objeto tendrá que adquirir previamente **el control del monitor** de ese objeto (también llamado **cerrojo del objeto**), si el monitor del objeto está disponible, es decir, no está controlado por otro hilo, lo adquirirá y podrá ejecutar el código sincronizado que se ha definido en la sección crítica del objeto, **al finalizar la ejecución, el hilo liberará el monitor del objeto** y con ello pasarán a estado listo (cola de hilos preparados) todos los hilos que **están bloqueados esperando** a adquirir el monitor del objeto para ejecutar una sección crítica de ese objeto. Se dicen que están bloqueados esperando por el monitor de ese objeto.

Si el hilo, al intentar ejecutar una sección crítica, no puede adquirir el control del monitor del objeto, por no estar libre, el hilo se bloqueará (pasará a **estado esperando** para adquirir monitor del objeto) y sólo retornará al estado listo en cola de hilos preparados cuando el monitor esté libre.

Definir una Sección de código sincronizada = definir una secciones críticas= es obtener **operaciones atómicas**

Operaciones atómicas: durante la ejecución de una sección crítica, **el hilo** que la está ejecutando, por tanto, que ha adquirido el monitor del objeto, **no podrá ser interrumpido ni por otro hilo ni por el SO (aunque agote su cuanto de ejecución, el planificador de hilos no podrá interrumpir la ejecución de ese hilo hasta que no termine la ejecución de la sección crítica), LA SECCIÓN CRITICA SE EJECUTARÁ ENTERA .UNA SECIÓN CRÍTICA ES UNA UNIDAD ATÓMICA.**

UNA SECCIÓN CRÍTICA ES UNA SECCIÓN DE CÓDIGO QUE SE EJECUTA EN EXCLUSIVIDAD POR UN SOLO HILO Y QUE SE EJECUTA ENTERA DE FORMA ATÓMICA.

A.2) Segunda forma: Definir un **bloque de código que se ejecuta sobre un determinado objeto**, en este caso la sección crítica se delimita como un bloque.

Se identifica un bloque sincronizado en relación a un objeto.

```
Synchronized (objeto)
{
    //código que se ejecuta sobre objeto
}
```

CONCEPTO DE MONITOR REENTRANTE

Estamos ante la situación de que un método/bloque definido como sincronizado (es una sección crítica) tiene en su cuerpo que llamar **a otro método de la misma clase** también definido como sincronizado (es otra sección crítica).

Cuando un hilo ejecuta el primer método toma el control del monitor del objeto, y al intentar ejecutar el segundo método, que es llamado desde el código del primero, cabría esperar que el hilo se bloquee intentando de **adquirir el control del monitor del objeto QUE YA POSEE.**

Si fuera así el hilo se auto bloquearía en espera de adquirir el monitor del objeto que él mismo debe ceder.

LA MÁQUINA JAVA PERMITE A UN HILO TOMAR EL CONTROL DEL MONITOR DE UN OBJETO PORQUE YA LO TIENE. PORQUE LOS MONITORES JAVA SON REENTRANTES.

Entonces, dentro del código de una sección crítica podremos hacer una llamada a otra sección crítica del mismo objeto.

Definir una sección crítica soluciona el problema de sincronizar hilos cooperantes que acceden al mismo objeto, pero cuando estos hilos realizan una misma tarea, es decir, son instancias de la misma clase.

Estos hilos, realizan la misma tarea, y en su tarea intentan acceder al mismo objeto/ mismos objetos cuya referencia han recibido.

Determinadas operaciones sobre el objeto/objetos compartidos las ” protegemos”: impedimos que dos o más hilos accedan al mismo tiempo al mismo objeto a realizar esa/esas operaciones, definimos una/varias secciones críticas.

Así en cada instante sólo un hilo de los n hilos que podrán intentar acceder a un objeto para ejecutar una sección crítica del objeto podrá estar en ejecución, todos los demás hilos que intentan ejecutar cualquier sección crítica de ese objeto quedan bloqueados

B) USO DE LOS MÉTODOS wait() , notify() y notifyAll()

Los métodos wait() y notify() nos sirven para sincronizar **hilos que comparten un mismo objeto pero además estos hilos realizan tareas distintas, son instancias de distintas clases**.

En este caso se hace necesario además de establecer secciones críticas **ESTABLECER UNA SECUENCIA DE ACCESO.**

Estamos en la situación de que una tarea depende de otra, entonces, los hilos del proceso no sólo no pueden acceder al mismo tiempo a cualquier sección de código definida como crítica para un objeto ,sino que el programador tiene que diseñar además una secuencia de acceso.

En estos casos tendremos que, además de definir las secciones críticas necesarias para garantizar **la exclusividad y atomicidad de ejecución**, hacer uso de estos tres métodos wait(), notify() y notifyAll() para diseñar una secuencia u orden de acceso.