Procesos. Hilos

En este módulo se describe el concepto de hilo, distintos modelos de hilos a nivel de usuario y a nivel kernel

Tabla de Contenidos

Autor: Lina García Cabrera

1 Introducción

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Procesos: Hilos (Threads)
El modelo de procesos clásico asume que un **proceso es un programa ejecutándose con un único hijo de control o traza de ejecución**. Este modelo de proceso tiene dos importantes limitaciones:

1. Muchas aplicaciones desean **ejecutar tareas en gran medida independientes que pueden ejecutarse concurrentemente**, pero deben compartir un espacio de direcciones común, además de otros recursos (servidores de bases de datos, monitores de procesamiento de transacciones, protocolos de red de capas medias y altas, etc.). Esos procesos son inherentemente paralelos y necesitan de un modelo de programación que permita el paralelismo.
2. Los procesos tradicionales no pueden **sacar partido de las arquitecturas multiprocesadores**, dado que un proceso sólo puede usar un procesador a la vez. Una aplicación debe crear cierto número de procesos y despacharlos sobre los procesadores disponibles. Estos procesos deben encontrar la forma de compartir memoria y recursos, y sincronizar sus tareas los unos con los otros.

Estas limitaciones se pueden superar con el concepto de **Hilo** o Hebra (*Thread*), surgen de este modo los **sistemas operativos multihilo**. Este módulo introduce conceptos relacionados con estos sistemas operativos: procesos e hilos, modelos de implementación de hilos y bibliotecas de hilos.
Un **hilo** es una traza o secuencia de control dentro de un proceso que ejecuta las instrucciones de un proceso de forma independiente. Se describe por su **pila y bloque de control de hilo** en el que se almacena su **identificación, estado de ejecución y contexto de ejecución** (contador de programa, conjunto de registros).

A comienzos de los 90 el uso de los hilos se reducía a la investigación en universidades y en sectores industriales específicos. A finales de los 90 y en el siglo XXI, el uso de los hilos se ha popularizado gracias a la incorporación del concepto de hilo en el lenguaje Java.

OBJETIVOS

- Conocer el concepto de hilo y los beneficios que aportan los sistemas operativos multihilo.
- Distinguir entre los distintos modelos de implementación de hilos.

2 Motivación

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Procesos: Hilos (Threads)
El uso de múltiples procesos en una aplicación tiene algunas desventajas obvias.

La creación de esos procesos añade una **sobrecarga** substancial, dado que *fork* es una llamada al sistema costosa. Como cada proceso tiene

- su propio espacio de direcciones, debe utilizar mecanismos de comunicación entre procesos, tales como paso de mensajes o memoria compartida.
- Se necesita trabajo adicional para despachar los procesos sobre diferentes máquinas o procesadores, pasar información entre los procesos, esperar su finalización, y recolectar los resultados.
- Finalmente, UNIX no tiene el marco adecuado para compartir ciertos recursos, por ejemplo, conexiones de red. Tal modelo solo está justificado si el beneficio de la concurrencia desplaza al costo de crear y gestionar múltiples procesos.

La abstracción de proceso es inadecuada y, por tanto, se necesitan mejores mecanismos para la computación paralela. Sobre todo cuando las distintas unidades de ejecución tienen relativamente pocas interacciones con otras y, por tanto, pocos requisitos de sincronización. **La abstracción hilo representa una única unidad computacional o de ejecución**. El proceso UNIX tradicional es de un único hilo, significando que toda la computación está serializada dentro de la misma unidad.

3 MultiHilo

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Procesos: Hilos (Threads)

Un proceso es una entidad compuesta que engloba dos conceptos separados y potencialmente independientes: uno relativo a la propiedad de recursos y otro que hace referencia a la ejecución.

- **Unidad que posee recursos**: A un proceso se le asigna un espacio de memoria y, de tanto en tanto, se le puede asignar otros recursos, como dispositivos de E/S o ficheros.
- **Unidad a la que se le asigna el procesador**: Un proceso es un flujo de ejecución (una traza) a través de uno o más programas. Esta ejecución se entremezcla con la de otros procesos. De tal forma que un proceso tiene un estado (en ejecución, listo, etc) y una prioridad de expedición. La unidad planificada y expedida por el sistema operativo es el proceso.

En la mayoría de los sistemas operativos, estas dos características son, de hecho, la esencia de un proceso. Sin embargo, son independientes, y pueden ser tratadas como tales por el sistema operativo.

Esta distinción ha conducido en los sistemas operativos actuales a desarrollar la construcción conocida como **thread**, cuyas traducciones más frecuentes son **hilo**, **hebra** y, por error, **proceso ligero**.
Si se tiene esta división de características:

- **la unidad de asignación de la CPU se conoce como *hilo*, mientras que a**
- **la unidad que posee recursos se le llama *proceso***.

Dentro de un **proceso** puede haber **uno o más hilos de control**, cada uno con:

- Un estado de ejecución (en ejecución, listo, bloqueado).
- Un contexto de procesador, que se salva cuando no esté ejecutándose.
- Una pila de ejecución.
- Algún almacenamiento estático para variables locales.
- Acceso a la memoria y a los recursos de ese trabajo que comparte con los otros hilos.

Un **Hilo** (en inglés, ***thread***) es una unida básica de utilización de la CPU;

- cuenta con un identificador de hilo,
- un contador de programa,
- un conjunto de registros y
- una pila.

Comparte con otros hilos que pertenece al mismo proceso:

- su sección de código,
- sección de datos, y
- otros recursos del sistema operativo como son los archivos abiertos y señales.

Un proceso tradicional (o proceso pesado) tiene un solo hilo de control. Si un proceso tiene múltiples hilos de control, puede realizar más de una tarea a la vez.

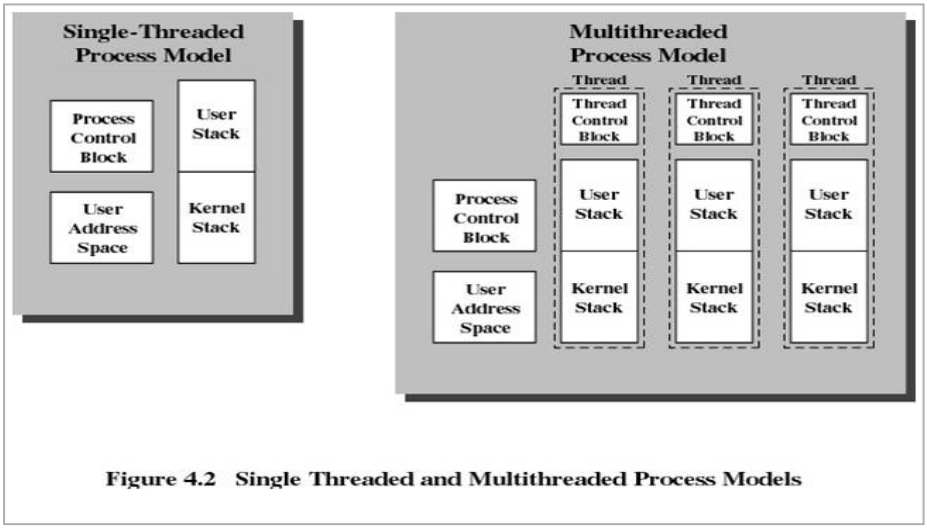


Figure 4.2 Single Threaded and Multithreaded Process Models

Figura 1. Modelo de procesos de un hilo y multihilo.

Los beneficios clave de los hilos se derivan de las implicaciones del **rendimiento**:

- se tarda **menos tiempo** en **crear un nuevo hilo de un proceso que ya existe**,
- en **terminarlo**, y
- en hacer un **cambio de contexto entre hilos de un mismo proceso**.

- Al someter a un mismo proceso a varios flujos de ejecución se mantiene **una única copia en memoria del código**, y no varias.

> Un ejemplo de aplicación que podría hacer uso de los hilos es un **servidor de ficheros de una red de área local**. Cada vez que llega una solicitud de una operación sobre un fichero, se puede generar un nuevo hilo para su gestión. El servidor gestiona multitud de solicitudes, por tanto, se pueden crear y destruir muchos hilos en poco tiempo para dar servicio a estas peticiones. Si el servidor es un multiprocesador, se pueden ejecutar varios hilos de un mismo proceso simultáneamente en diferentes procesadores.

4 Concurrencia y Paralelismo

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Procesos: Hilos (Threads)
Para comprender los diferentes tipos de abstracciones de hilos, debemos distinguir entre concurrencia y paralelismo.

> El **paralelismo** de una aplicación multiprocesador es el **grado real de ejecución paralela alcanzado** y está, por tanto, limitado por el **número de procesadores** físicos disponibles para la aplicación.

> La **concurrencia** de una aplicación es el **máximo paralelismo que puede alcanzar con un número ilimitado de procesadores**. Esta depende de como esté escrita la aplicación, y de cuantos hilos de control puedan ejecutarse simultáneamente, con los recursos adecuados posibles.

La **concurrencia** puede suministrarse a nivel de sistema o de aplicación. El **kernel suministra concurrencia de sistema** reconociendo múltiples hilos de control (también denominadas hilos calientes) dentro de un proceso y planificándolos independientemente. El kernel multiplexa esos hilos entre los procesadores disponibles. Una aplicación puede beneficiarse de la concurrencia de sistema incluso sobre un monoprocesador, debido a que si un hilo se bloquea sobre un suceso o recurso, el kernel planifica otro hilo.

La **aplicación puede suministrar concurrencia de usuario** a través de bibliotecas de hilos a nivel de usuario. Tales hilos de usuario, o corrutinas (también denominadas hilos fríos), no son reconocidas por el kernel y deben ser planificadas y manejadas por la propia aplicación. Estas no suministran verdadera concurrencia o paralelismo, ya que esas hilos no pueden ejecutarse en paralelo. Sin embargo, suministran un modelo de programación natural para aplicaciones concurrentes.

Utilizando ***llamadas al sistema no bloqueantes***, las aplicaciones pueden mantener simultáneamente varias iteraciones en progreso. Las hilos de usuario simplifican la programación capturando el estado de tales iteraciones en variables locales por hilo (sobre la pila del hilo) en lugar de en una tabla de estado global.

Cada uno de los modelos de concurrencia tiene un valor limitado por sí mismo:

- Las hilos se usan tanto como herramientas de organización como para explotar múltiples procesadores.
- Las hilos kernel permiten la ejecución paralela sobre multiprocesadores, pero no son aconsejables para estructurar aplicaciones de usuario.

> Por ejemplo, una aplicación servidora podría querer crear miles de hilos, una por cada cliente. Las hilos kernel consumen recursos valiosos tales como memoria física (dado que muchas implementaciones requieren que las estructuras de hilos permanezcan en memoria), y por tanto, no son aconsejables para tal programa. En contra, un mecanismo a nivel de usuario puro es útil sólo para estructurar aplicaciones y no permite la ejecución paralela del código.

Muchos sistemas implementan un modelo de **concurrencia mixto** que combina **concurrencia de sistema y de usuario**:

- El kernel reconoce múltiples hilos en un proceso, y
- la biblioteca añade hilos de usuario que no son vistas por el kernel.

Las **hilos de usuario suministran sincronización entre rutinas concurrentes** en un programa sin la sobrecarga de hacer llamadas al sistema, y además son deseables incluso en sistemas sin núcleos multihilo. Es más, siempre es buena idea reducir el tamaño y la responsabilidad del kernel, y dividir la

funcionalidad de soporte de hilos entre el kernel y la biblioteca de usuario es consistente con esta estrategia.

5 Modelos MultiHilo

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Procesos: Hilos (Threads)
Existen dos grandes categorías para la implementación de hilos: **los hilos a nivel de usuario** (ULT, *User Level Thread*) y **los hilos a nivel de núcleo** (KLT, *Kernel Level Thread*).

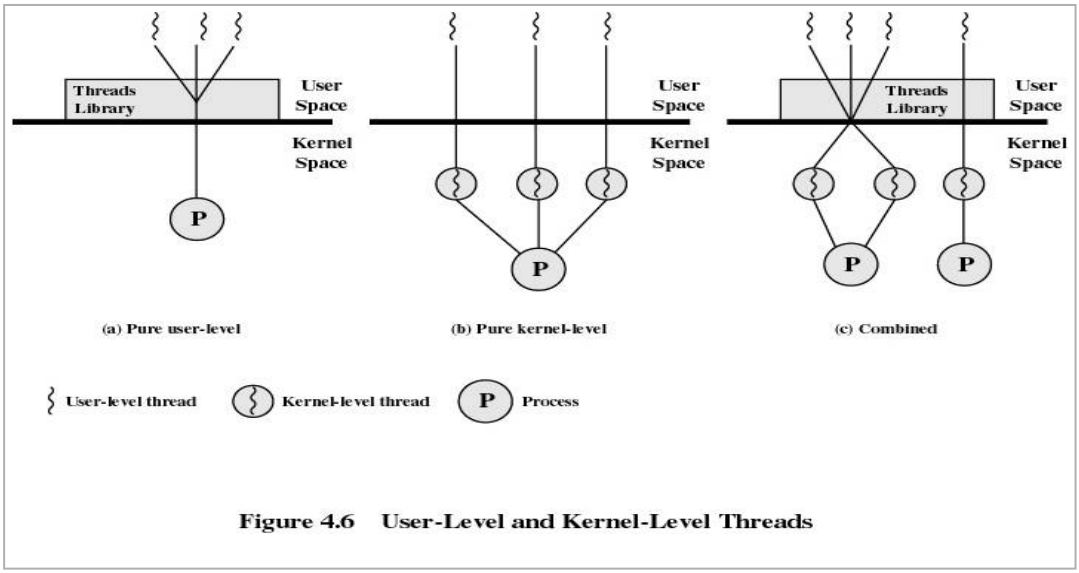


Figura 2. Hilo a nivel de usuario y a nivel del kernel.

- Los **hilos a nivel de usuario** son los que utilizamos para programar y pueden crearse desde lenguajes de programación como Java.
- Los **hilos a nivel de núcleo** o sistema son del propio sistema operativo y pueden dar soporte a los hilos de usuario.

5.1 Hilos a nivel de Usuario

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Procesos: Hilos (Threads)

En una aplicación ***User Level Thread*** (**ULT**) pura, todo el trabajo de gestión de hilos lo realiza la aplicación y el núcleo no es consciente de la existencia de hilos.

Se puede **programar una aplicación como multihilo con una biblioteca de hilos**. Tales bibliotecas suministran todas las funciones para crear, sincronizar, planificar, y manejar hilos, sin asistencia especial del *kernel*. Las interacciones entre hilos no involucran al *kernel* por lo que son muy rápidas (aunque algunas características de las bibliotecas necesitan la intervención del *kernel*).

Por defecto, una aplicación comienza su ejecución con un único hilo. **Esta aplicación y su hilo pertenecen a un único proceso, gestionado por el núcleo**. La **aplicación cuando esté en estado en-ejecución** puede crear un nuevo hilo que se ejecuta dentro del mismo proceso.

La creación la hace la biblioteca de hilos. El flujo de control pasa a esa utilidad mediante una llamada a procedimiento. La biblioteca de hilos crea una estructura de datos para el nuevo hilo y cede el control a uno de los hilos del proceso que esté Preparado o Listo (elegido mediante algún **algoritmo de planificación**). Cuando el control pasa a la biblioteca, se salva el contexto del hilo actual, que se restaura cuando el control pasa de la biblioteca al hilo. La **biblioteca actúa como un kernel en miniatura para las hilos que ella controla**.

Todas las operaciones anteriores, se hacen en el espacio de usuario y dentro del mismo proceso. El núcleo del sistema operativo mantiene la responsabilidad de la conmutación entre procesos, porque solamente él tiene el privilegio de modificar los registros de gestión de memoria.

Los hilos de usuario no son entidades verdaderamente planificables, y el kernel no tiene conocimiento de ellos. El **kernel sencillamente planifica el proceso base**, que por turnos utiliza las funciones de biblioteca para planificar sus hilos. Cuando un proceso es apropiado, lo son sus hilos. De la misma forma, **si un hilo de usuario hace una llamada al sistema bloqueante, éste bloquea al proceso base, todos los hilos se bloquean**.

Los hilos de usuario tienen varias ventajas:

- Para intercambiar hilos no hay que cambiar al modo núcleo porque todas las estructuras están en el espacio de usuario de un mismo proceso y mejora el rendimiento.
- Se puede hacer una planificación específica para cada aplicación.
- Se pueden ejecutar en cualquier sistema operativo sin cambiar su núcleo.

Existen dos desventajas claras en el uso de hilos de usuario:

- En un sistema operativo la mayoría de las llamadas al sistema son bloqueantes. Cuando un hilo se bloquea, se bloquean todos los hilos del proceso.
- En un sistema con varios procesadores no se puede aprovechar esta ventaja. El núcleo asigna un proceso a un procesador cada vez. No se puede ejecutar en paralelo los hilos de un proceso.

Una forma de **superar el problema del bloqueo de hilos** es mediante la **técnica del recubrimiento** (*jacketing*); convertir una llamada bloqueadora en no bloqueadora. No se hacen llamadas al sistema directamente sino que se invoca a una función de recubrimiento de la aplicación. Esta función comprueba antes si supondrá el bloqueo del hilo. En este caso, pasa el hilo a Preparado y cede el control a otro hilo.

5.2 Hilos a nivel de Kernel

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Procesos: Hilos (Threads)
En una aplicación ***Kernel Level Thread*** (**KLT**) pura, la gestión de los hilos la realiza el kernel:

- Todos los hilos de la misma aplicación pertenecen a un único proceso.
- El **núcleo mantiene la información de contexto del proceso como un todo y la de cada hilo dentro del proceso**.
- El núcleo planifica los hilos.
- Las propias funciones del núcleo pueden ser multihilo.

Los hilos pueden hacer llamadas al sistema y bloquearse por E/S o recursos. En un sistema multiprocesador, un proceso puede disfrutar de los beneficios del verdadero paralelismo, ya que **cada hilo puede despacharse para ejecutarse en un procesador diferente**. Las esperas de recursos y E/S bloquean hilos individuales, no al proceso entero.

Los hilos a nivel de kernel también tienen limitaciones:

- La mayoría de las operaciones, tales como creación, destrucción, y sincronización de hilos, **necesitan llamadas al sistema**. Las llamadas al sistema son relativamente **costosas** por varias razones:
  - cada llamada al sistema requiere dos cambios de modo - uno de modo usuario a kernel en la invocación, y otro a la inversa a la finalización.
  - En cada cambio de modo, el hilo cruza un dominio de protección.
  - El kernel debe copiar los parámetros de la llamada al sistema del espacio de usuario al del kernel y validarlos para protegerse frente a procesos maliciosos o erróneos.
  - De la misma forma, al retorno de la llamada, el kernel debe copiar los datos de vuelta al espacio de usuario.

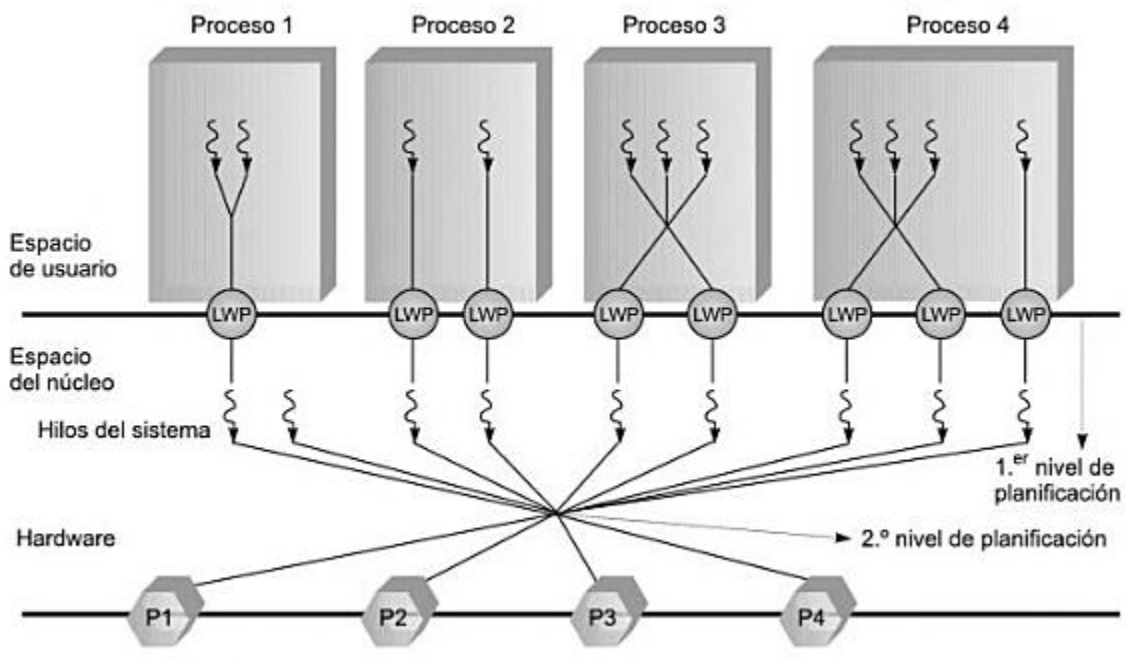
- Cuando los hilos acceden frecuentemente a datos compartidos, la sobrecarga de sincronización puede anular cualquier beneficio de rendimiento. La mayoría de los sistemas multiprocesadores suministran cerrojos que pueden adquirirse a nivel de usuario si no están bloqueados por otros hilos. Si un hilo desea un recurso que no está disponible actualmente, puede ejecutar una espera ocupada (iterar mientras el recurso queda libre), de nuevo sin la intervención del kernel. La espera ocupada es razonable para recursos que

están bloqueados por cortos periodos de tiempo; en otro caso, es necesario bloquear el hilo. Bloquear el hilo necesita de la intervención del kernel y, por tanto, aumenta su coste.

5.3 Planificación de Hilos

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Procesos: Hilos (Threads)
Algunos sistemas operativos ofrecen un mecanismo que combina hilos a nivel de usuario ULT e hilos a nivel de kernel KLT (como en Solaris). La creación de hilos, y la mayor parte de la planificación y sincronización de los hilos de la aplicación se realiza en el espacio de usuario. Los múltiples ULT de una aplicación se asocian con varios (el mismo o menor número) KLT.

Los sistemas operativos como Solaris definen el concepto de **procesador lógico**. **Los hilos de usuario se ejecutan en un procesador lógico**.



Planificación de procesos e hilos en Solaris ITES Paraninfo Programación Concurrente José Tomás Palma Méndez,M. C. Garrido Carrera,F. Sánchez Figueroa, A. Quesada Arencibia

De esta forma tenemos una **planificación a dos niveles**:

- Un primer nivel para asignar los hilos de usuario a los procesadores lógicos y
- otro para asignar los procesadores lógicos al procesador o procesadores físicos.

Hay tres técnicas distintas para hacer planificación de hilos sobre los recursos del núcleo (sobre las CPU's):

- Muchos hilos en un procesador lógico.
- Un hilo por procesador lógico.
- Muchos hilos en muchos procesadores lógicos.

5.3.1 Muchos hilos en un procesador lógico

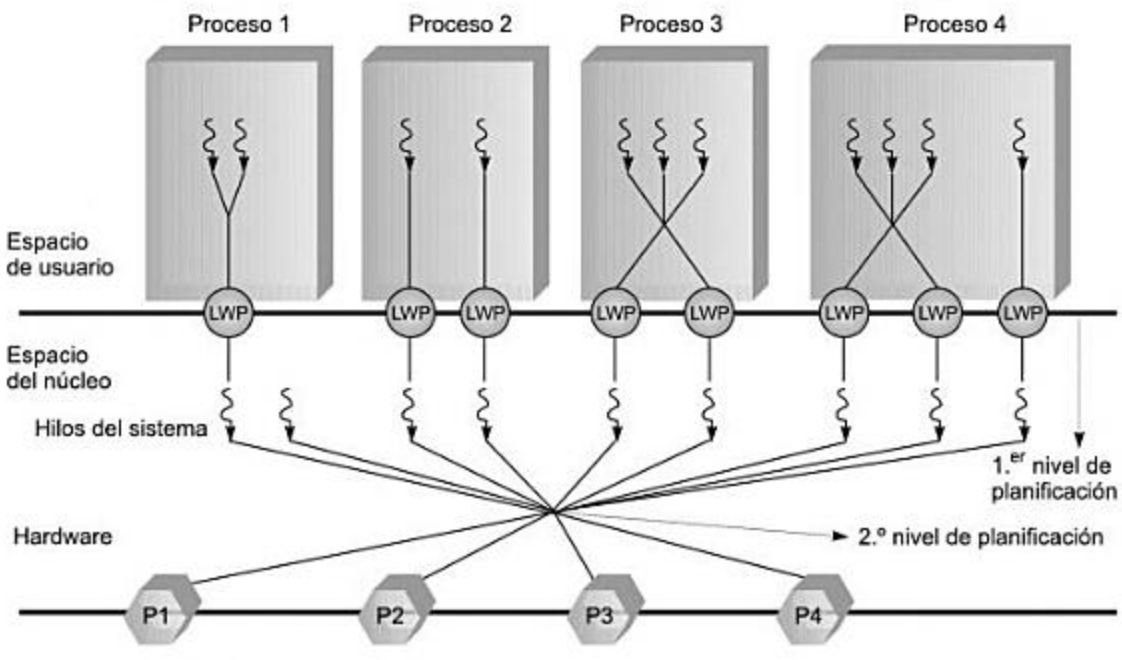
2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Procesos: Hilos (Threads)

Modelo **Muchos-a-uno**. Todos los hilos creados en el espacio de usuario compinten para ejecutarse en el único procesador lógico asignado a ese proceso. Un proceso no se beneficia de tener varios

procesadore físicos.

- Una llamada bloqueante (una E/S) provoca el bloqueo del proceso.
- Sin embargo, la creación de hilos y la planificación se hacen en el espacio de usuario sin usar recursos del núcleo, es más rápido.

Sería el caso del proceso 1 en la figura.



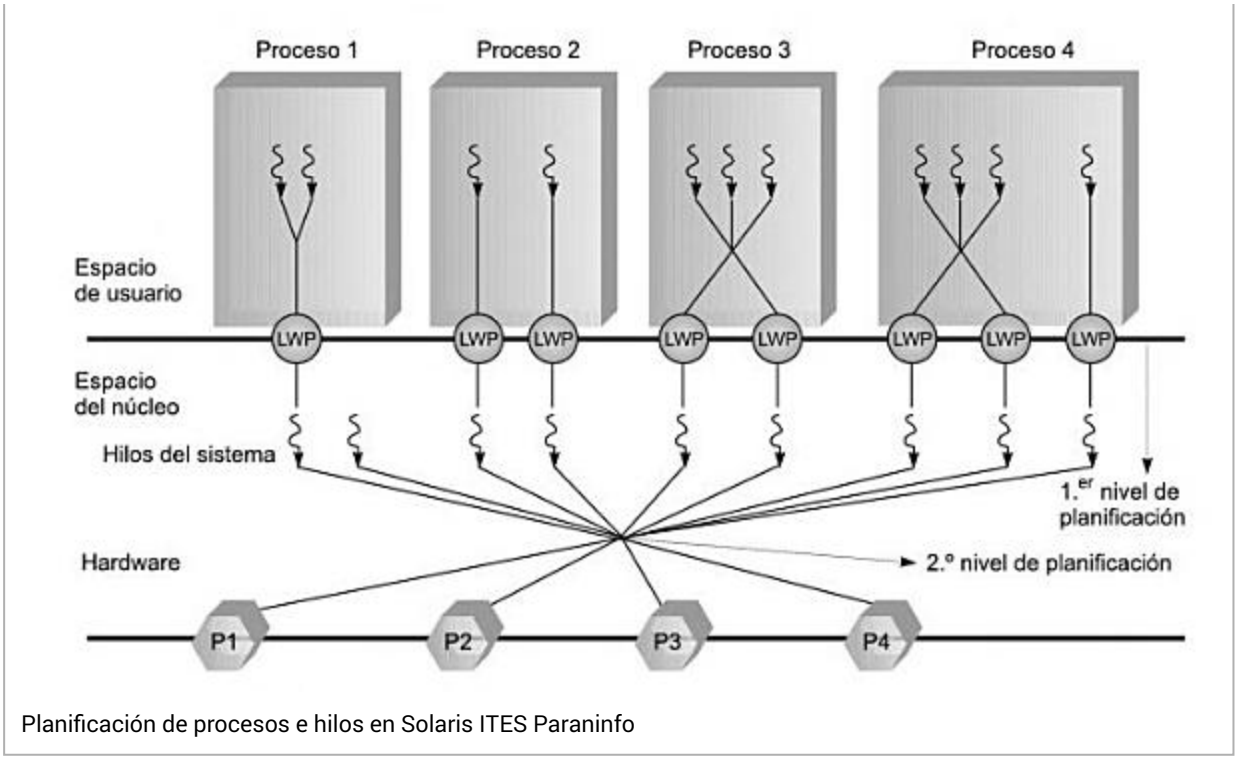
Planificación de procesos e hilos en Solaris ITES Paraninfo

5.3.2 Un hilo por procesador lógico

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Procesos: Hilos (Threads)

Modelo **Uno-a-Uno**. **Se asigna un procesador lógico a cada hilo de usuario**. Muchos hilos se pueden ejecutar simultáneamente en diferentes procesadores físicos. Sin embargo, la creación de hilos conlleva la creación de un procesador lógico y, por tanto, una llamada la sistema. Como cada procesador lógico toma recursos del núcleo, está limitado el número de hilos que se pueden crear.

Cualquier MVJ basada en librerías que utilizan este modelo como Win32, OS/2, LinuxTreads de Xavier Leroy usan este modelo.
Sería el caso del proceso 2 de la figura.



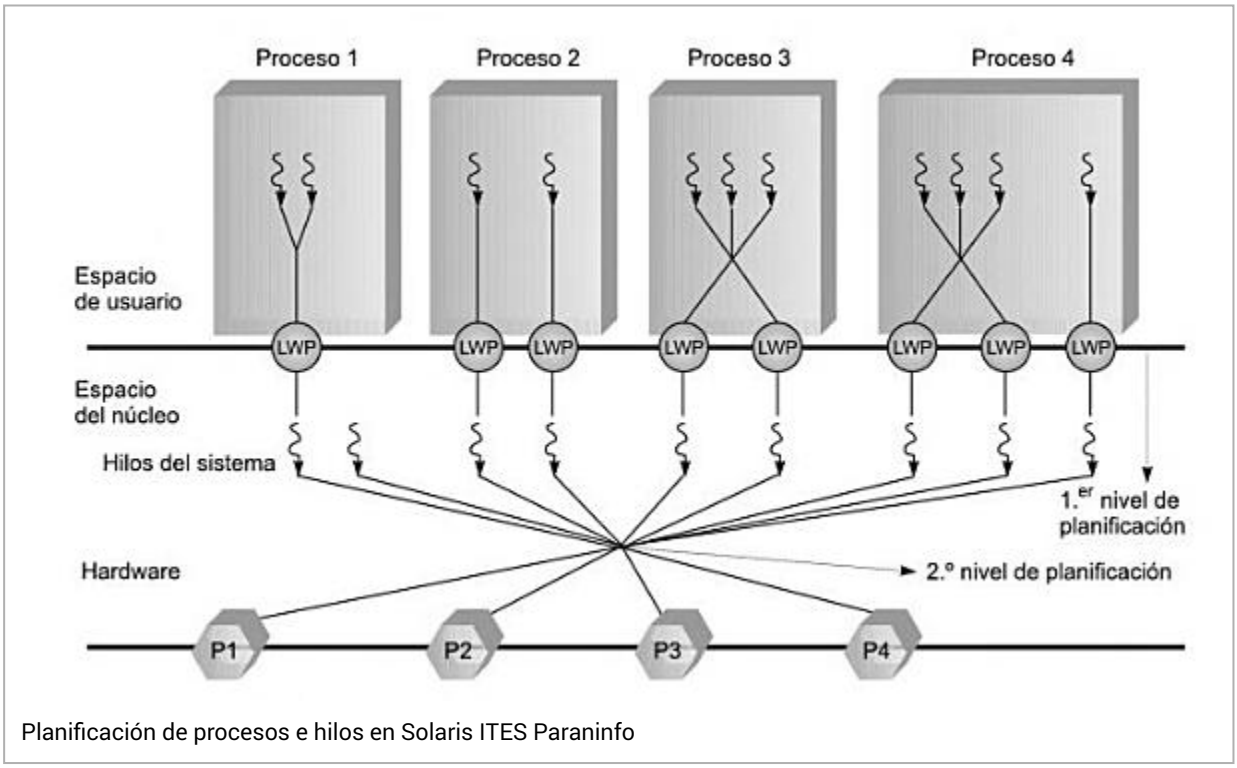
Planificación de procesos e hilos en Solaris ITES Paraninfo

5.3.3 Muchos hilos en muchos procesadores lógicos

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Procesos: Hilos (Threads)

Modelo **Muchos-a-Muchos (estricto)**. Un número de hilos se multiplexa en un número de procesadores lógicos igual o menor. Muchos hilos pueden ejecutarse en paralelo en diferentes CPUs y las llamadas al sistema de tipo bloqueante no bloquean al proceso entero.

Sería el caso del proceso 3 de la figura.



Planificación de procesos e hilos en Solaris ITES Paraninfo

El modelo de dos niveles

Modelo Muchos-a-Muchos (no estricto). Como el anterior pero ofrece la posibilidad de hacer un enlace a un hilo específico con un procesador lógico. Probablemente sea el mejor modelo.

Sistemas operativos como Digital UNIX, Solaris, IRIX, HP-UX, AIX usan este modelo.

En el caso de Java, las máquinas virtuales MVJ sobre estos SSOO tienen la opción de usar cualquier combinación de hilos enlazados o no. La elección del modelo de hilos es una elección al nivel de implementación de los programadores de la MVJ. Java en sí mismo no tiene el concepto de procesador lógico.

Sería el caso del proceso 4 de la figura.

6 Bibliotecas de hilos

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Procesos: Hilos (Threads)
Una biblioteca de hilos proporciona al programador una API para crear y gestionar hilos. Existen dos formas principales de implementar una biblioteca de hilos: **en el espacio del usuario** y **en el espacio del kernel**.

- Si la biblioteca está en el **espacio del usuario**, no existe soporte por parte del sistema operativo. El código y las estructuras de datos de la biblioteca están en el espacio del usuario. Por tanto, invocar una función de la biblioteca es como hacer una llamada a una función.
- Si la biblioteca está en el **espacio del kernel**, el código y las estructuras de datos de la biblioteca están en el espacio del kernel. Invocar una función en la API provoca una llamada al sistema.

6.1 Pthreads

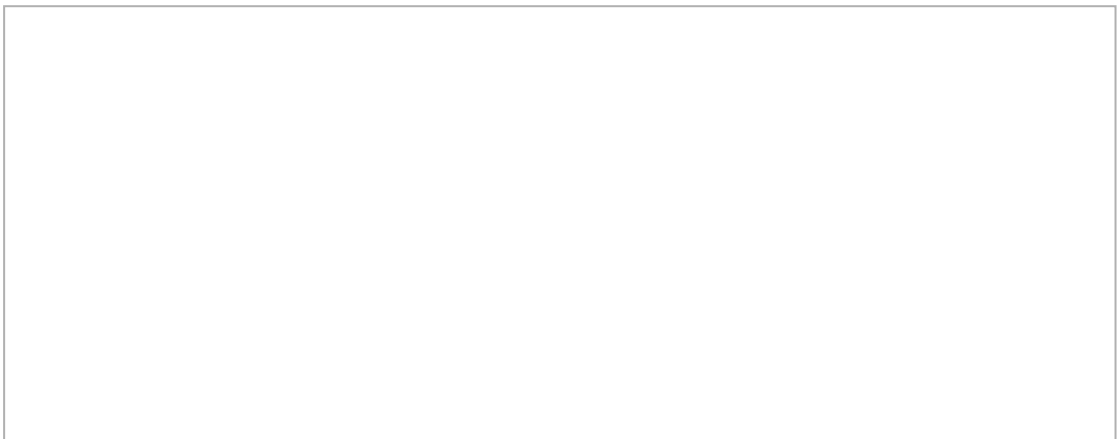
2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Procesos: Hilos (Threads)

`Pthreads`, la extensión de hilos del estándar POSIX, puede proporcionarse como biblioteca a nivel de usuario o de kernel. Se trata de una especificación no de una implementación y cada sistema operativo la puede implementar como desee (Solaris, Linux, Mac OS X).

El programa C de la figura muestra un ejemplo de un programa multihilo que calcula el sumatorio de un entero no negativo en un hilo. El programa comienza con una sola hebra de control `main()`. Después de la inicializaciones, `main()` crea el segundo hilo que comienza con la función `runner()`. Ambos comparten la variable global `sum`.

Todos los programas deben incluir el fichero `pthread.h`. Con `pthread_t tid` se declara el hilo. Cada hilo tiene una serie de atributos `pthread_attr_t attr` como el tamaño de pila y la información de planificación. Para crear otro hilo se llama a `pthread_create()`. Se le pasa el hilo, los atributos y el nombre de la función que ejecutará, `runner()` y, por último, el parámetro de entrada que se escribió en la línea de órdenes, `argv[1]`.

Después de crear el hilo sumatorio, el hilo padre espera a que se complete llamando a `pthread_join`.



```c
#include <pthread.h>
#include <stdio.h>

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
   pthread_t tid; /* the thread identifier */
   pthread_attr_t attr; /* set of thread attributes */

   if (argc != 2) {
      fprintf(stderr,"usage: a.out <integer value>\n");
      return -1;
   }
   if (atoi(argv[1]) < 0) {
      fprintf(stderr,"%d must be >= 0\n",atoi(argv[1]));
      return -1;
   }

   /* get the default attributes */
   pthread_attr_init(&attr);
   /* create the thread */
   pthread_create(&tid,&attr,runner,argv[1]);
   /* wait for the thread to exit */
   pthread_join(tid,NULL);

   printf("sum = %d\n",sum);
}

/* The thread will begin control in this function */
void *runner(void *param)
{
   int i, upper = atoi(param);
   sum = 0;

   for (i = 1; i <= upper; i++)
      sum += i;

   pthread_exit(0);
}
```

Figura 3. Programa C multihilo usando API Pthreads.