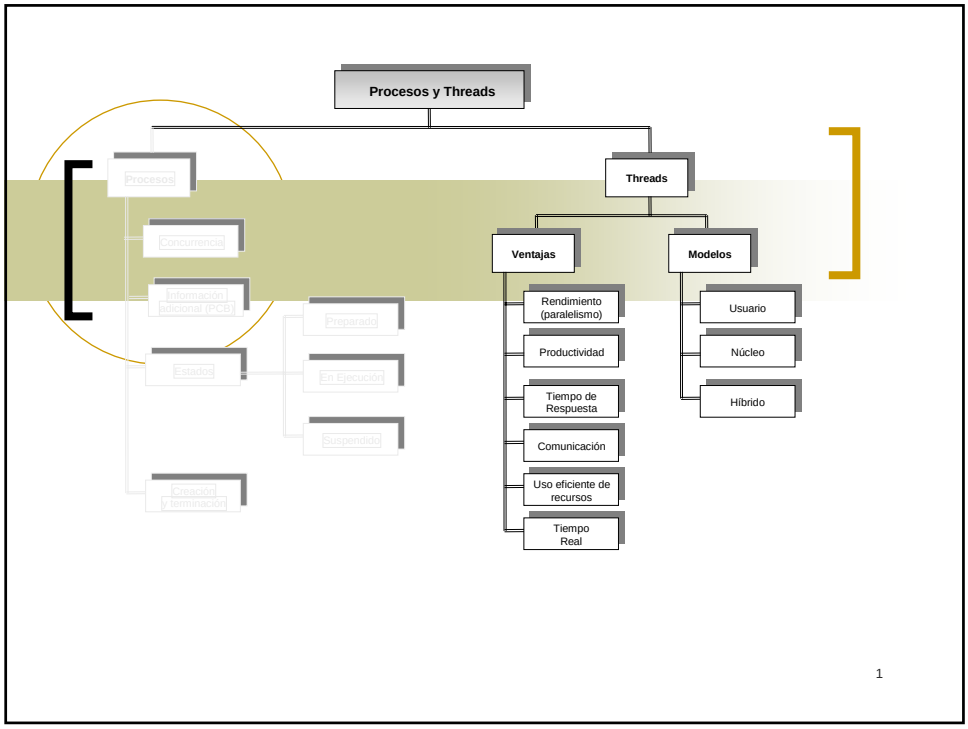


Threads (I)

- Hay SO donde un proceso puede tener internamente ***tareas concurrentes*** llamadas *hilos de ejecución* o **threads**
- Ejemplo:

Un proceso (guiado del robot) puede tener un hilo de ejecución para cada uno de los sensores que detectan obstáculos y otro para la tarea de elegir la trayectoria óptima

Threads (II)

- Un thread se puede ver como una extensión natural del modelo de proceso: se permite la ejecución de varios ***hilos/threads*** (*unidad planificable independiente*) dentro del mismo proceso
- Los distintos threads de un proceso comparten el código, los datos y los recursos del proceso

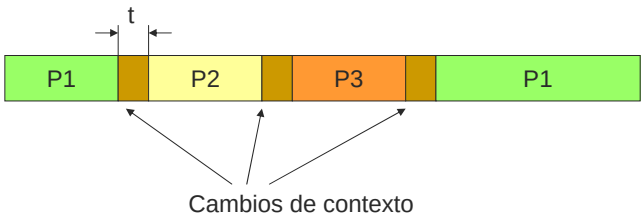
Proporcionan un mecanismo eficiente para la comunicación y sincronización (**comparten recursos**) además de evitar en cierta medida los cambios de contexto

Cambio de Contexto (I)

- Implementación de varios procesos: cambio de contexto
  - La ejecución aparentemente simultánea de varios procesos en un mismo sistema, requiere repartir el tiempo de CPU entre los objetos a ejecutar
  - Eso implica desasignar la CPU (expulsar) al proceso en ejecución y asignarla a un proceso *preparado*. Esta actividad se conoce como ***cambios de contexto***
- Realizar un cambio de contexto lleva consigo:
  - Guardar el estado (contexto) del proceso en ejecución PCB
  - Poner en la CPU el contexto del nuevo proceso
  - Actualizar la información de control de procesos

Cambio de Contexto (II)

- Los cambios de contexto no son trabajo útil e implican una sobrecarga importante si se hacen con frecuencia: reducen la utilización



$$Utilizacion = \frac{T(P1)+T(P2)+T(P3)}{T(P1)+T(P2)+T(P3)+3t}$$

Ventajas del uso de Threads

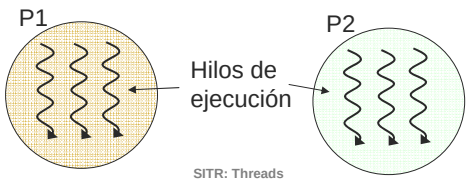
- Aumento del rendimiento en multiprocesamiento hardware (paralelismo)
- Aumento de productividad (uso efectivo de la CPU)
- Disminución del tiempo de respuesta
- Facilita la comunicación entre tareas
- Uso más eficiente de los recursos del sistema
- Simplifica el procesamiento en tiempo real
- Programas mejor estructurados
- Portabilidad

Inconvenientes del uso de Threads

- Mayor complejidad de programación
- No todos los programas se pueden implementar (o son más eficientes) con threads
- La conversión de antiguos programas a su equivalente con threads no es una tarea fácil

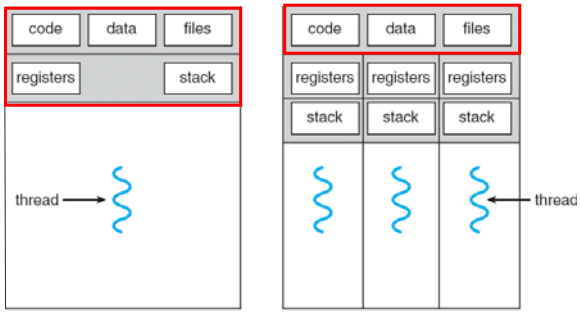
Modelos de Control de Threads (I)

- Las entidades de ejecución vistas hasta ahora son:
  - **Proceso** (*proceso pesado*): unidad de propiedad de los recursos (memoria, manejadores ficheros,…) y planificable
    - Tiene muchos atributos y es lento de crear
    - Cambio de contexto lento
  - **Hilo de ejecución (Thread)**: actividad concurrente dentro de un proceso
    - Todos los hilos de ejecución que pertenecen a un mismo proceso comparten los recursos del mismo (memoria, manejadores de ficheros,…)
    - Tienen pocos atributos y se crean rápidamente
    - Cambio de contexto rápido



Modelos de Control de Threads (II)

- **Compartición de memoria**
  - Los procesos UNIX no comparten memoria (código o datos). Cada uno tiene su propio espacio de direcciones
  - Los hilos de ejecución de un mismo proceso pueden tener rutinas o variables comunes
  - No obstante, cada hilo tendrá su propia pila donde se guardarán las variables locales y argumento de invocación



single-threaded process multithreaded process

9

Modelos de Control de Threads (III)

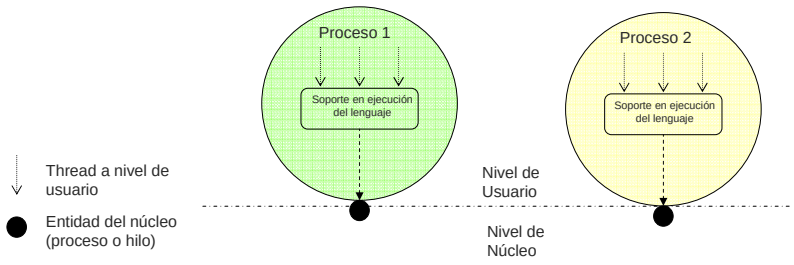
- Existen varias implementaciones en la gestión por parte del SO de los threads o hilos de ejecución:
  - **Hilos a nivel de usuario**
  - **Hilos a nivel de núcleo**
  - **Modelo híbrido**

Modelos de Control de Threads (IV)

- Hilos a nivel de Usuario:
  - Los Hilos se crean a nivel del proceso de usuario por medio de un conjunto de **funciones de biblioteca** o mediante el soporte de ejecución del lenguaje de programación:
    - El Sistema operativo sólo crea un hilo de ejecución en el núcleo por cada espacio de direcciones (proceso)
    - El resto de los hilos son implementados por el ***run-time*** del lenguaje
    - Es la aproximación utilizada en SO que no soportan nativamente hilos
    - Los cambios de contexto entre hilos a nivel de usuario son rápidos, ya que no involucran llamadas al sistema
    - La política de planificación de estos hilos no está restringida a la propia de SO
    - Cuando este tipo de hilos invocan llamadas al sistema bloqueantes, normalmente se bloquea todo el proceso
    - No pueden aprovechar un sistema multiprocesador, ya que el sistema operativo ve un único hilo de ejecución

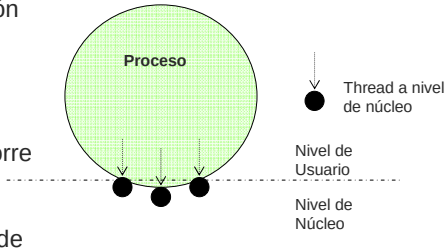
Modelos de Control de Threads (V)

- Esquema de hilos a nivel de usuario:



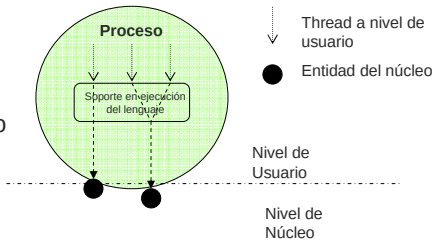
Modelos de Control de Threads (VI)

- Hilos a nivel de núcleo:
  - El SO soporta hilos de ejecución y proporciona un conjunto de ***llamadas al sistema*** para su manipulación
  - El sistema crea un hilo de ejecución (*Proceso Ligero*) para cada hilo a nivel de usuario
  - Son la unidad de planificación del sistema
  - El bloqueo y activación de hilos corre a cargo del núcleo
  - Los cambios de contexto son más lentos que en el caso de los hilos de usuario



Modelos de Control de Threads (VII)

- Modelo Híbrido:
  - Se implementan las dos clases de hilos anteriores
  - El SO permite más de un hilo por proceso
  - El soporte del lenguaje de programación utiliza un hilo del núcleo para implementar un grupo de hilos de usuario
  - Proporciona flexibilidad y el máximo rendimiento potencial al programador de la aplicación



Threads: Conclusión

- Los Threads proporcionan una estructura de programación fiable que ahorra recursos
- La utilización de threads aumenta la productividad y disminuye el tiempo de respuesta (deseable en los SITR)
- Aprovecha la capacidad de los sistemas multiprocesador

- Ejemplo de coste temporal de distintas operaciones sobre los distintos modelos:

Tiempos de los servicios de threads con Solaris 2.3 en una SPARCstation 2	
Operación	**Microsegundos**
Crear thread de usuario	**52**
Crear thread de núcleo	**350**
fork()	**1700**
Sincronizar thread de usuario	**66**
Sincronizar thread de núcleo	**390**
Sincronizar entre procesos	**200**