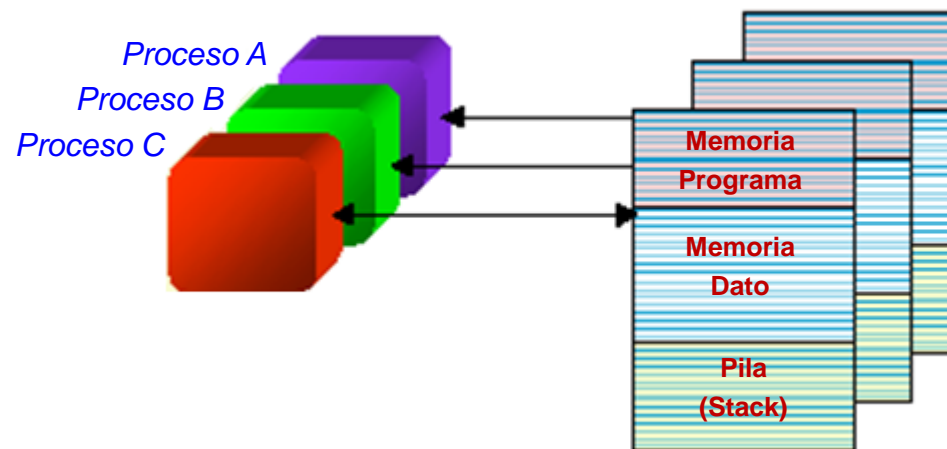


Hilos (Threads), Concurrencia

Procesos e Hilos

- Cada programa está compuesto por uno o varios procesos.
 - ✓ Cada proceso dispone de su propio espacio virtual protegido de memoria para **datos globales**, **datos dinámicos** y **código**, además de **recursos** solicitados al sistema operativo (ejemplo ficheros).
 - ✓ Cada proceso se ejecuta en uno o varios hilos (threads).
 - ✓ Cada hilo corresponde a una línea de ejecución secuencial de instrucciones y dispone de **pila propia** (datos locales, retornos de rutinas) e información de **contexto propia** (registros de la CPU, estado).

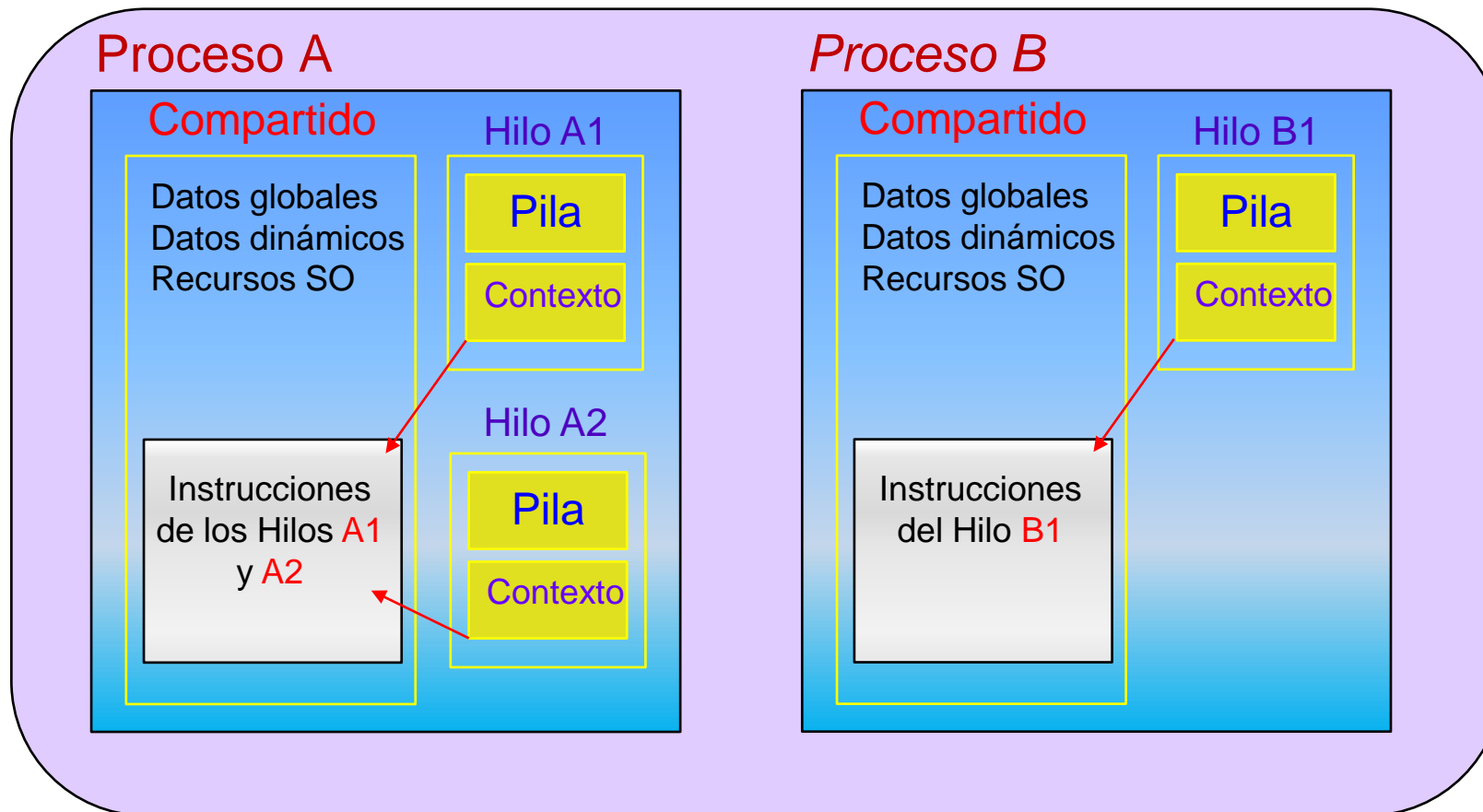


Hilos (Threads)

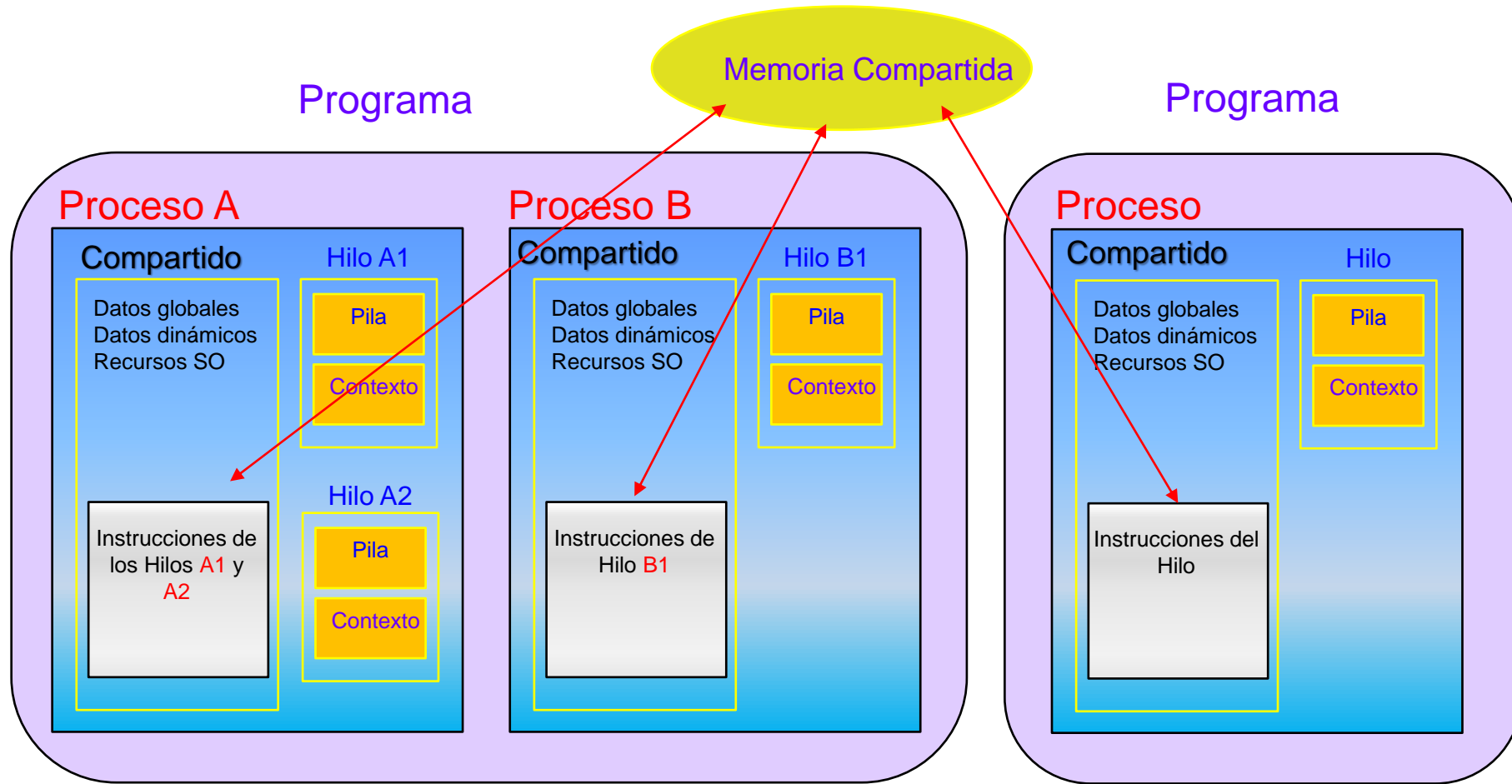
- Un **proceso** tradicional (**proceso pesado**) se compone de una tarea con un hilo de ejecución
- Es definido dentro de los sistemas operativos como el **subproceso** o unidad más pequeña y compacta que influye en la planificación de los tiempos de procesamiento entre los diferentes procesos.
- Lo que es propio de cada hilo es el **contador de programa**, la **pila de ejecución** y **el estado de la CPU** (incluyendo el valor de los registros).
- Cada hilo comparte con los otros hilos cooperantes código, datos y recursos del SO
- El **código**, **los datos** y los **recursos** son poseídos por otra entidad conocida como tarea (**TASK**).
- Cada hilo sólo puede pertenecer a una tarea

Procesos e Hilos

- Todos los hilos **comparten el mismo espacio de direcciones** y otros **recursos** como pueden ser **archivos** abiertos.



Interacción entre Procesos

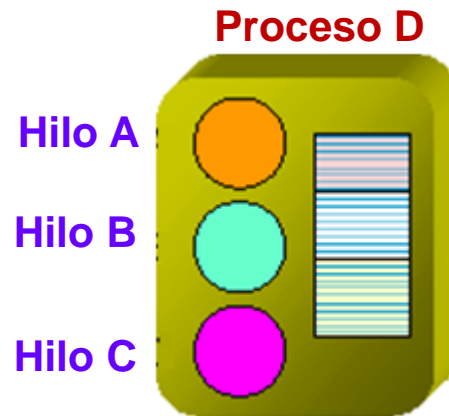


¿Como interaccionan los hilos o procesos?

- ✓ **Servicios** ofrecidos por el sistema operativo como intermediario.
- ✓ **Datos:** colas de mensajes, memoria compartida.
- ✓ **Sincronización:** semáforos, monitores, etc.
- ✓ **Señales**

Hilos (Threads)

- Al cambiar de un proceso a otro el sistema operativo (mediante el dispatcher) genera lo que se conoce como **overhead**, que es tiempo desperdiciado por el procesador para realizar un **cambio de contexto**
- En los hilos, como **pertenecen a un mismo proceso**, al realizar un cambio de hilo, la conmutación de un hilo a otro en la misma tarea requiere un coste mínimo, ya que solo es necesario salvar los registros y conmutar la pila



Hilos (Threads)

- Cualquier **modificación de un recurso** desde un hilo **afecta** al entorno del **resto de los hilos del mismo proceso**. Por lo tanto, es necesario sincronizar la actividad de los distintos hilos para que no interfieran unos con otros o **corrompan** estructuras de **datos**.



Hilos (Threads)

- El proceso **sigue en ejecución** mientras al menos **uno de sus hilos** de ejecución siga activo.
- Cuando el proceso es terminado, todos sus hilos de ejecución también terminan. En el momento en el que todos los hilos de ejecución finalizan, el proceso no existe más y todos sus recursos son liberados.



Sincronización de Hilos

- Una ventaja de la programación multihilo es que los programas operan con mayor velocidad en sistemas de computadores con múltiples CPUs (sistemas multiprocesador o a través de grupo de máquinas) ya que los hilos del programa se prestan verdaderamente para la ejecución concurrente. En tal caso el programador necesita ser cuidadoso para evitar **condiciones de carrera** (problema que sucede cuando diferentes hilos o procesos alteran datos que otros también están usando), y otros comportamientos no intuitivos.
- Los threads son muy adecuadas para sistemas distribuidos y sistemas multiprocesador (cada hilo se puede ejecutar en un procesador)
- Los SOTR brindan servicios para hilos análogos a los de los procesos
- Los hilos pueden ser usuario o como llamadas al sistema

Hilos (Threads)

- Un ejemplo de la utilización de hilos es tener un hilo atento a la interfaz gráfica (iconos, botones, ventanas), mientras otro hilo hace una larga operación internamente. De esta manera el programa responde de manera más ágil a la interacción con el usuario. También pueden ser utilizados por una aplicación servidora para dar servicio a múltiples clientes de procesos.

Hilos (Threads)

- Casos en los que es conveniente utilizar hilos:
 - ✓ Servicio para muchos usuarios a la vez, como por ejemplo un servidor Web
 - ✓ La comunicación a través de la red (sockets)
 - ✓ Realice la operación que consume tiempo
 - ✓ La distinción entre las tareas por prioridad
 - ✓ Para que la interfaz de usuario pueda continuar para "responder" a consultas de los usuarios, mientras que se lleva a cabo la tarea de fondo.

Lenguaje que incluya soporte de ejecución RT

- Los **lenguajes** que permiten concurrencia pueden crear su propio soporte de ejecución (en inglés run-time support) que sustituye (o puede sustituir) al sistema operativo.
- Estos **lenguajes de programación** tienen características de diseño expresamente creadas para permitir a los programadores lidiar con hilos de ejecución (como Java).
- **Otros** (la mayoría) desconocen la existencia de hilos de ejecución y éstos deben ser creados mediante llamadas de biblioteca especiales que dependen del sistema operativo en el que estos lenguajes están siendo utilizados (como es el caso del C y del C++).

soporte de ejecución RT +lenguaje convencional

- Es posible utilizar un lenguaje “convencional” con un soporte de ejecución específico (en inglés **run time support**) que sustituye (o puede sustituir) al sistema operativo.

Implementación en un SO de tiempo Real

- Utilizan un conjunto de **llamadas al sistema** (primitivas) que permiten:
 - ✓ Crear procesos con prioridades
 - ✓ Planificación basada en prioridades
 - ✓ Fijar procesos en memoria (no intercambio)
 - ✓ E/S rápida y eficiente
 - ✓ Memoria compartida
 - ✓ Mecanismo de concurrencia de bajo nivel

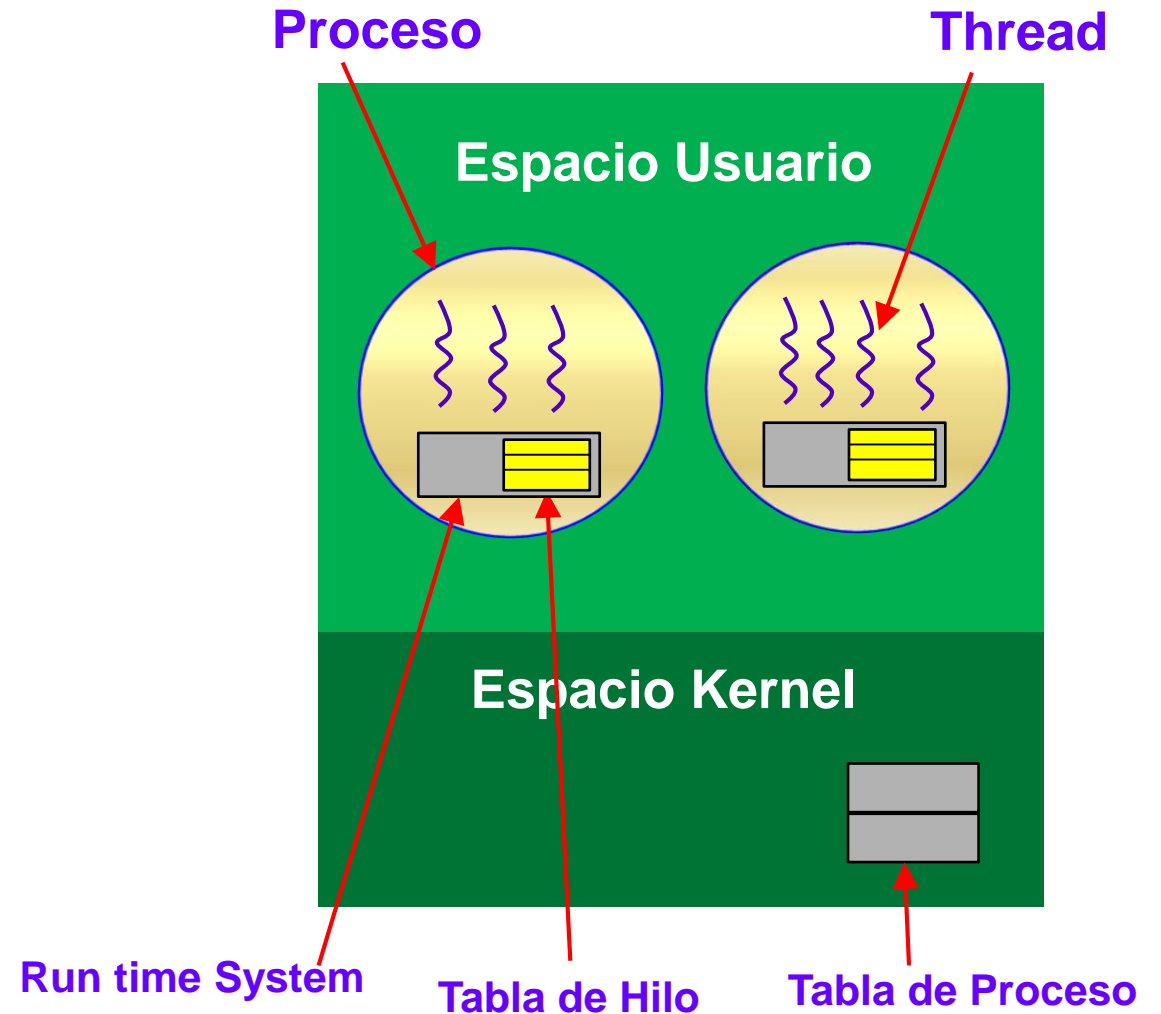
Basados en UNIX (RT UNIX), QNX, RTEMS

Hilos en **espacio usuario**

- Ventajas:
 - ✓ La conmutación entre hilos se puede realizar rápidamente sin ayuda del S.O.
 - ✓ La planificación puede hacerla la aplicación
 - ✓ Portabilidad entre SO diferentes
- Inconvenientes:
 - ✓ Si el S.O. no sabe de la existencia de hilos en una tarea, el bloqueo de un hilo produce el bloqueo del resto de hilos de la misma tarea
 - ✓ Dos hilos de una misma tarea no se pueden ejecutar en procesadores diferentes

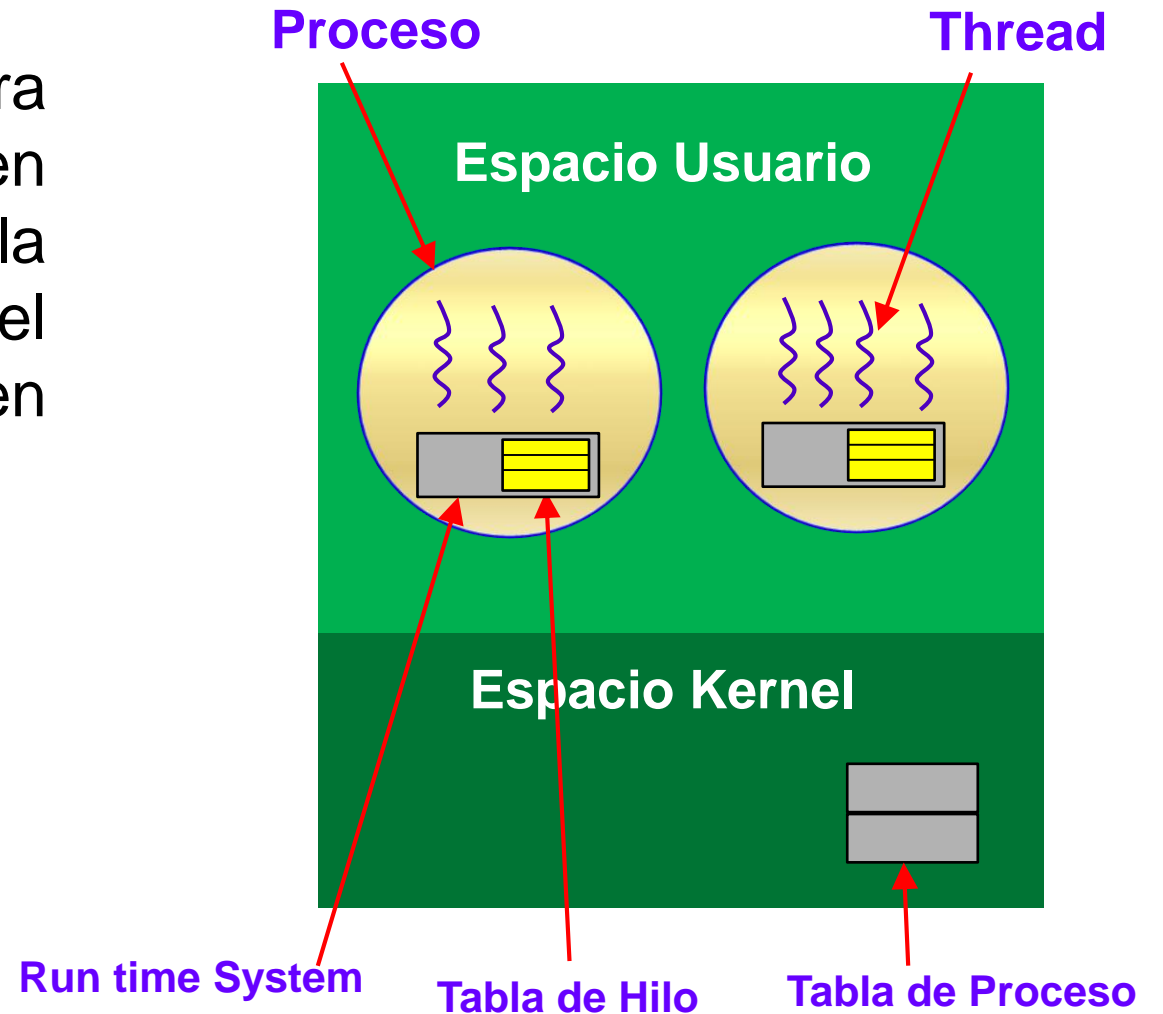
Hilos en **espacio usuario**

- La administración de los hilos es realizada por una librería de hilos a nivel usuario
- Los hilos corren encima de un *run-time system*, el cual es una colección de procedimientos que administra los threads.
- Si los hilos son administrados en espacio usuario, cada proceso necesita su propia **tabla de hilos**



Hilos en **espacio usuario**

- La información necesaria para restaurar un hilo esta almacenada en la **tabla del hilo**, exactamente de la misma manera como el kernel almacena información del proceso en la **tabla del proceso**
- Examples:
 - ✓ POSIX Pthreads
 - ✓ Mach C-threads
 - ✓ Solaris threads

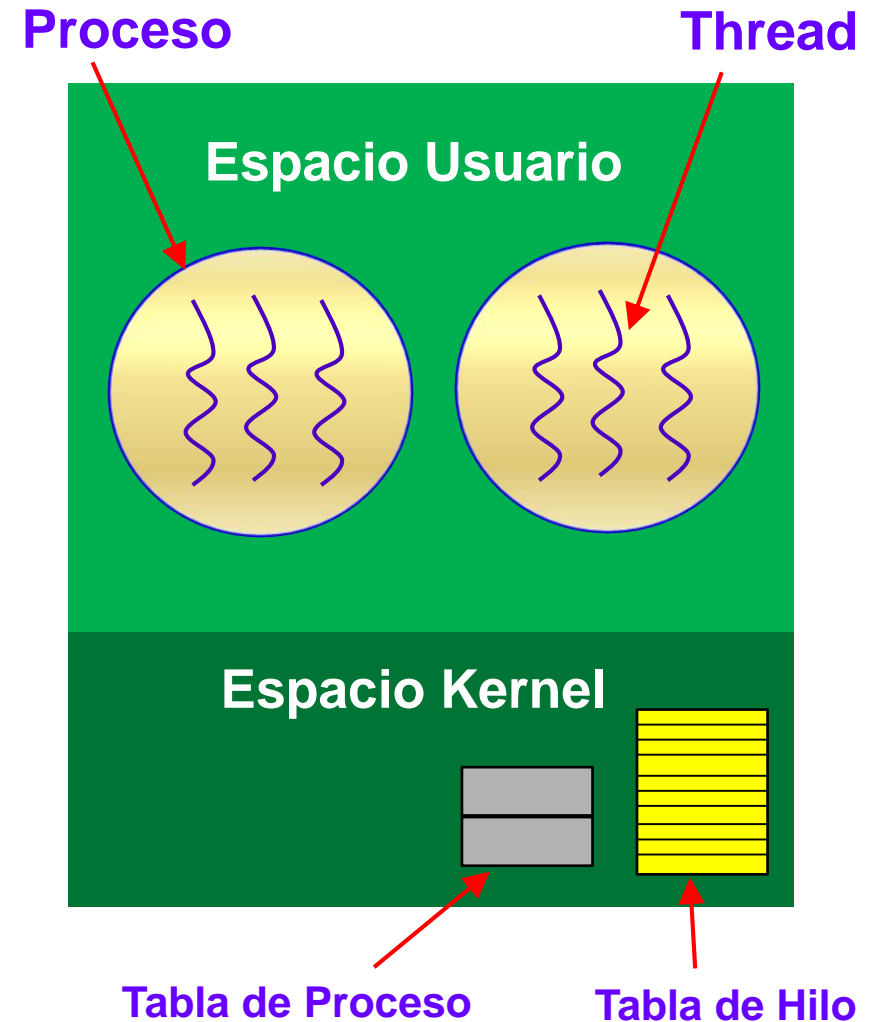


Hilos soportados por el S.O.

- Ventajas:
 - ✓ Si el S.O. soporta hilos, el bloqueo de uno de ellos en una tarea no afecta al resto
 - ✓ Puede tomar ventaja de múltiples CPUs
- Inconvenientes:
 - ✓ La creación y destrucción es mas costosa
 - ✓ La conmutación de un hilo a otro se hace vía interrupciones (mayor sobrecarga)

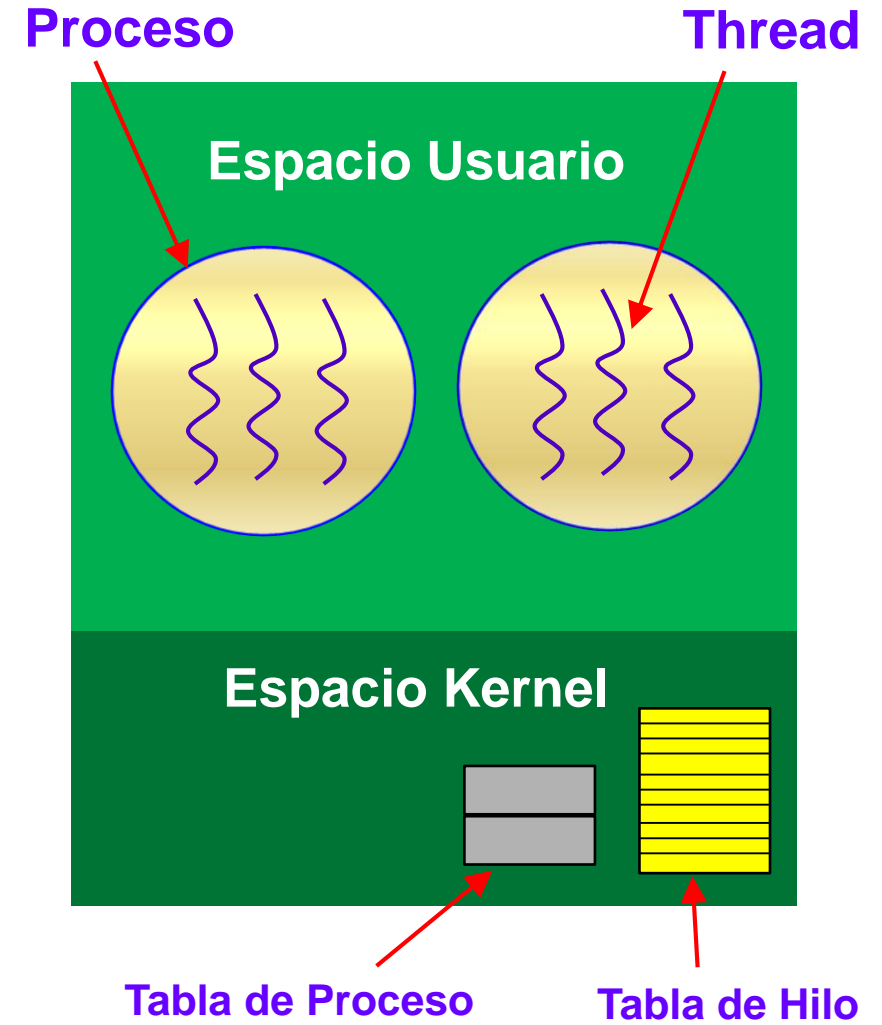
Hilos en **espacio kernel**

- Ningún *run-time system* es necesario
- No hay una Tabla de hilo para cada proceso. El kernel tiene una tabla de hilos, que mantiene las propiedades de **todos** los hilos en el sistema. Cuando se crea un hilo se hace una llamada al kernel el cual actualiza dicha tabla.
- La información es la misma como con los hilos en espacio usuario pero ahora la información están en el kernel en lugar del espacio usuario (run-time system).



Hilos en **espacio kernel**

- Esta información es un subconjunto de la información que tradicionalmente mantiene el kernel acerca de cada uno de sus procesos con un solo hilo.
- Ejemplos
 - ✓ Windows 95/98/NT/2000
 - ✓ Solaris
 - ✓ Tru64 UNIX
 - ✓ BeOS
 - ✓ Linux



Ventajas de los hilos

- Los hilos se **crean** más rápidamente que los procesos:
Se tarda mucho menos tiempo en crear un hilo nuevo en un proceso existente que en crear un proceso. Algunas investigaciones llevan al resultado que esto es así en un factor de 10.
- Los hilos se **destruyen** más rápidamente que los procesos
Se tarda mucho menos en terminar un hilo que un proceso, ya que cuando se elimina un proceso se debe eliminar el **BCP** del mismo, mientras que un hilo se elimina su contexto y pila.

Ventajas de los hilos

- El tiempo de conmutación (**cambios de contexto**) entre hilos de la misma tarea (proceso) es más rápida que la conmutación entre procesos
- Los hilos tienen espacio de direcciones comunes esto hace mas fácil coordinar actividades
- Menor sobrecarga de comunicaciones debido a que todos los hilos de una tarea comparten memoria.
- Los hilos son útiles en aplicaciones que tiene que hacer muchas cosas concurrentemente:
 - ✓ **Un hilo espera por una I/O**
 - ✓ **Otro hilo en el mismo proceso esta haciendo algo de procesamiento (algoritmo)**

Ventajas de los hilos

- Los hilos aumentan la eficiencia de la comunicación entre programas en ejecución. En la mayoría de los sistemas en la comunicación entre procesos debe intervenir el núcleo para ofrecer protección de los recursos y realizar la comunicación misma. En cambio, entre hilos pueden comunicarse entre sí sin la invocación al núcleo. Por lo tanto, si hay una aplicación que debe implementarse como un conjunto de unidades de ejecución relacionadas, es más eficiente hacerlo con una colección de hilos que con una colección de procesos separados.

Hilos vs Procesos

Ventajas de los procesos

- Espacios de memoria virtual **independientes** significa disponer de un alto nivel de seguridad. Un proceso con problemas no puede corromper estructuras de datos de otros.
- Codificados de forma independiente, preparados para ser ejecutados en **computadores diferentes**.

Realización

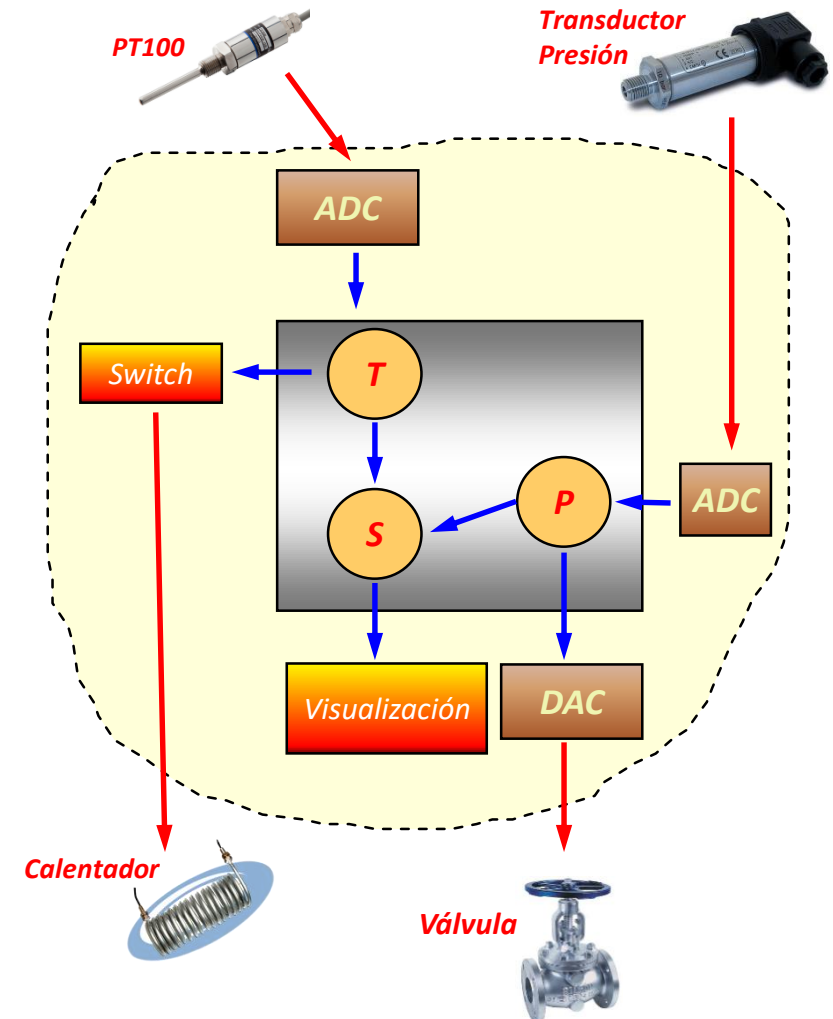
- La mayoría de los S.O. modernos soportan threads (OS/2, Mach, W2K, chorus, linux, etc).
- ¿Cómo programar con hilos?:
 - Bibliotecas estándar
 - POSIX Threads
 - Sun threads, etc.

*Cómo
programar
con hilos ?*



Aplicación: sistema de control

- El **objetivo** global del sistema es el de **mantener** la **temperatura** y **presión** de algún proceso químico dentro de **unos límites preestablecidos**.




Arquitectura de software

- Hay tres actividades concurrentes:
 - ✓ Control de presión.
 - ✓ Control de temperatura.
 - ✓ Visualización.
- Se puede hacer de tres formas:
 1. Con **un solo programa secuencial** (ejecutivo cíclico). No hace falta sistema operativo
 2. Con **tres procedimientos secuenciales** ejecutados como procesos de un **sistema operativo**
 3. Con **un solo programa en un lenguaje concurrente** . No hace falta sistema operativo, pero sí **un núcleo de ejecución**.

1. Solución con ejecutivo cíclico

Con *un solo programa secuencial* (ejecutivo cíclico). No hace falta sistema operativo

```
with Data_Types; use Data_Types;
with IO; use IO;
with Control_Procedures; use Control_Procedures;
procedure Controller is
  TR : Temperature;
  PR : Pressure;
  HS : Heater_Setting;
  VS : Valve_Setting;
begin
  loop
    Read (TR);
    Control_Temperature (TR, HS);
    Write (HS);
    Write (TR);
    Read (PR);
    Control_Pressure (PR, VS);
    Write (VS);
    Write (PR);
  end loop;
end Controller;
```



1. Solución con ejecutivo cíclico

Problemas:

- La temperatura y la presión se muestrean con la misma frecuencia.
- Mientras se espera leer una variable no se puede hacer nada con la otra. Si un sensor falla, deja de leerse el otro también.
- Se basa en una espera ocupada. Además ambas actividades deberían estar desacopladas.
- *El principal problema es que dos actividades independientes están acopladas entre sí*
- Se utiliza un único programa que ignora la concurrencia lógica de T, P y S
=> No es necesario sistema operativo.

2. Solución con un lenguaje concurrente

- Con **un solo programa en un lenguaje concurrente** . No hace falta sistema operativo, pero sí **un núcleo de ejecución**.

```
with Data_Types; use Data_Types;
with IO; use IO;
with Control_Procedures; use Control_Procedures;
procedure Controller is

    task Temperature_Controller;
    task Pressure_Controller;

    task body Temperature_Controller is
        TR : Temperature; HS : Heater_Setting;
    begin
        loop
            Read (TR); Control_Temperature (TR, HS); Write (HS); Write (TR);
        end loop;
    end Temperature_Controller;

    task body Pressure_Controller is
        PR : Pressure; VS : Valve_Setting;
    begin
        loop
            Read (PR); Control_Pressure (PR, VS); Write (VS); Write (PR);
        end loop;
    end Pressure_Controller;
```

- Es **más fácil** de escribir y leer
- El código **refleja** directamente el **paralelismo** de la aplicación

2. Solución con un lenguaje concurrente

- **Problemas:**
- Ambas tareas envían datos a la pantalla
 - ✓ hay que **gestionar el acceso a este recurso compartido**
 - ✓ si la **escritura** no se hace de forma **exclusiva** por cada tarea, el **resultado es ilegible**
- Hace falta un mecanismo para la gestión de recursos
 - ✓ **sincronización** de tipo exclusión mutua
 - ✓ hace falta el apoyo del **entorno de ejecución**

Hilos de ejecución

- ¿Cómo se programa con hilos? Se tienen básicamente dos posibilidades:
 - a. Utilizando un lenguaje de programación convencional y llamadas al sistema (o funciones de biblioteca)
 - ✓ Ejemplo: Hilos POSIX utilizados desde C, (enlazando con la librería pthreads, opción -lpthread)
 - b. Utilizando construcciones lingüísticas (o clases) de un lenguaje de programación concurrente
 - ✓ Ejemplo: Tareas en Ada95, threads en Java
- En este caso, el compilador traduce las construcciones lingüísticas a:
- ✓ Llamadas al sistema (hilos a nivel de núcleo)
 - ✓ Llamadas a bibliotecas propias de soporte de hilos (hilos a nivel de usuario)