

Sistemas Operativos

Threads - I



Sistemas Operativos

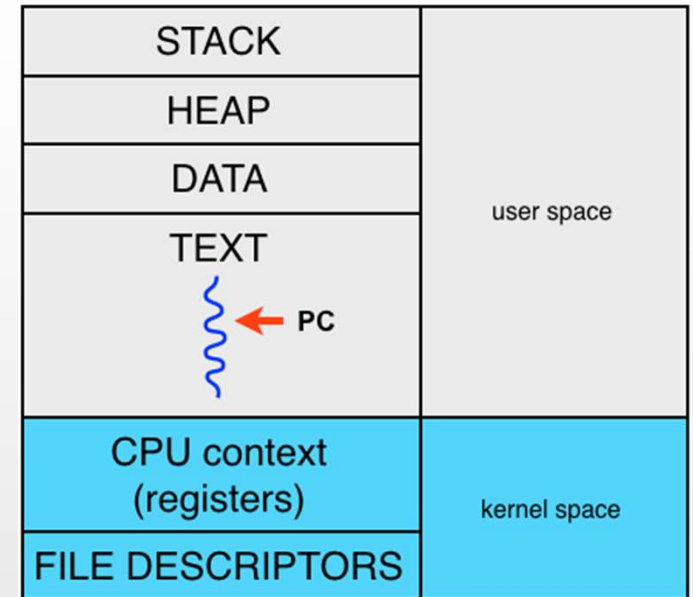
- ✓ Versión: Abril 2025
- ✓ Palabras Claves: Threads, Hilos, ULT, KLT, Procesos, Concurrencia, Paralelismo, Multithreading

Algunas diapositivas han sido extraídas de las ofrecidas para docentes desde el libro de Stallings (Sistemas Operativos) y el de Silberschatz (Operating Systems Concepts)

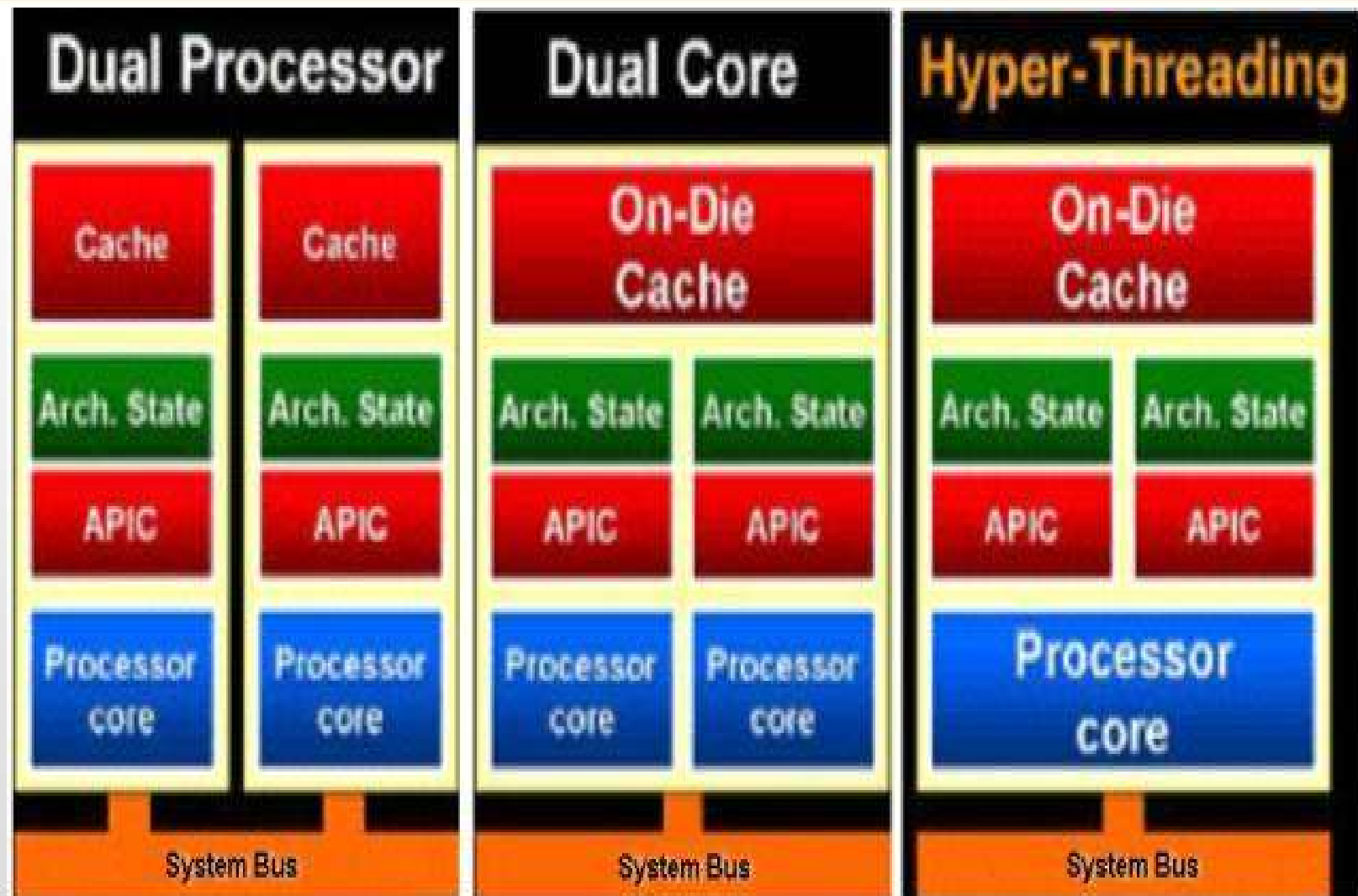


Primeros SO – Procesos

- ✓ Unidad básica de utilización de CPU: Proceso
- ✓ Programa en Ejecución
- ✓ Unidad de asignación de los recursos
- ✓ Conceptos relacionados con proceso:
 - ✓ Espacio de direcciones
 - ✓ Punteros a los recursos asignados (stacks, archivos, etc.)
 - ✓ Estructuras asociadas: PCB, tablas
- ✓ En los primeros sistemas operativos, cada proceso tenía un espacio de direcciones y **un solo hilo de control**.
 - ✓ Un único flujo secuencial de ejecución.
 - ✓ Se ejecuta una instrucción y cuando finaliza se ejecuta la siguiente
- ✓ Para ejecutar otro proceso, se debe llevar adelante un cambio de contexto

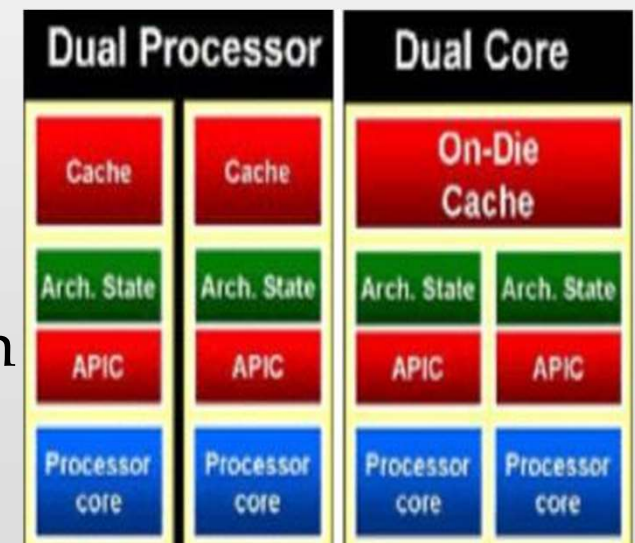


Evolución de Hardware



Evolución de Hardware

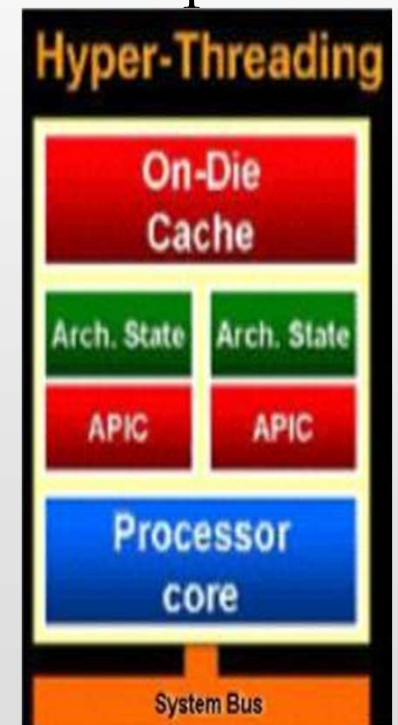
- ✓ **Sistemas Dual-processor (DP):** tiene 2 procesadores físicos en el mismo chasis. Pueden estar en la misma motherboard o no. Cache y controlador independientes.
- ✓ **Sistemas Dual-core:** una CPU con dos cores por procesador físico. Un circuito integrado tiene 2 procesadores completos. Los 2 procesadores combinan cache y controlador.
- ✓ En ambos casos, las APIC (Advanced Programmable Interrupt Controllers) están separadas por procesador. De esta manera proveen administración de interrupciones por procesador.



Evolución de Hardware

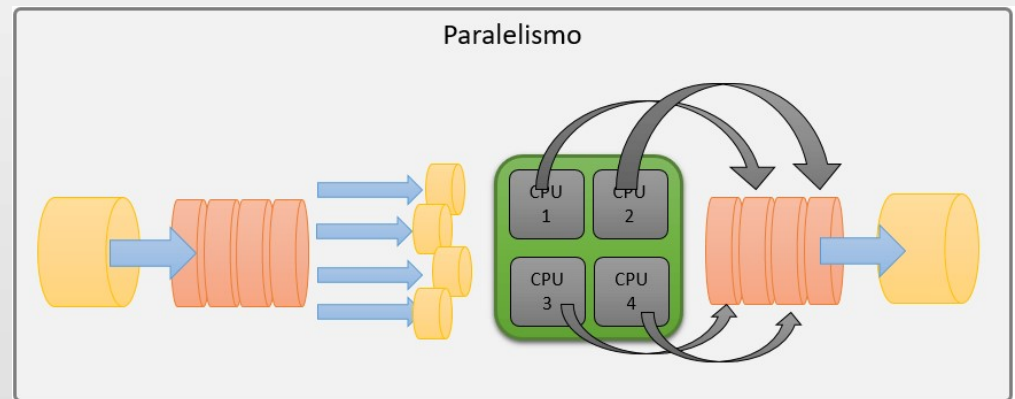
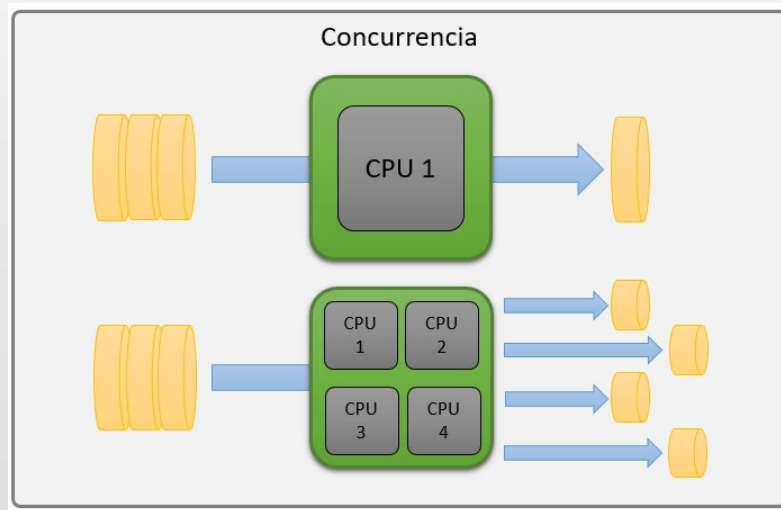
☑ Multithreading Simultáneo

- ✓ Generalmente conocido como Hyper Threading, lo cual es propietario de las CPU Intel
- ✓ Permite que el software programado para ejecutar múltiples hilos (multi-threaded) procese los hilos en paralelo dentro de un único procesador.
- ✓ La tecnología simula dos procesadores lógicos dentro de un único procesador físico
 - ♦ Duplica solo algunas “secciones” de un procesador
 - ♦ Registros de Control (MMU, Interrupciones, Estado, etc)
 - ♦ Registros de Proposito General (AX, BX, PC, Stack, etc.)
- ✓ Resultado: mejora en el uso del procesador (entre 20 y 30%)



Evolución del Software

- ✓ Es común dividir un proceso en diferentes “tareas” que, independientemente o colaborativamente, solucionan el problema
- ✓ Es común contar con un pool de procesadores para ejecutar nuestros procesos simultáneamente



Evolución del Software

- ✓ Procesador de texto: ingreso de caracteres, auto-guardado, análisis ortográfico/gramatical
- ✓ Navegador web que descarga distintos recursos de una página al mismo tiempo
- ✓ Acceso simultáneo a diferentes fuentes de E/S
- ✓ Tendencia de los procesadores actuales a contar con varios núcleos (multiprocesadores)
- ✓ Se incrementa el rendimiento de aplicaciones y se evitan bloqueos



Evolución del Software

```
#include <pthread.h>

void * myFunction(void * var) // Función que ejecutará el thread
{
    // Código a ejecutar por el/los threads
    return 0;
}

int main(int argc, char *argv[])
{
    int *res;
    pthread_t mythread; // Variable que almacena la información sobre el thread
    pthread_create(&mythread, NULL, myFunction, NULL); // Creamos el thread
    pthread_join(mythread, (void *)&res); // espera por la terminación del hilo.
    printf("El thread %d devolvio el valor %d\n", i, *res);
    return 0;
}
```



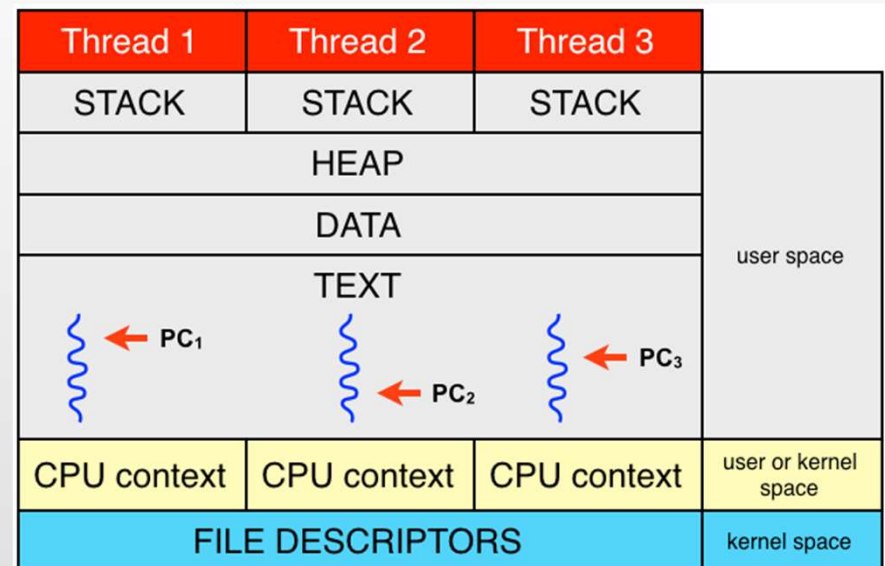
Evolución del Software

- ☑ Brindan herramientas que nos permiten separar las diferentes “tareas” de los programas en unidades de ejecución diferentes:
 - ✓ Java – heredar de “Thread”, implementar la interface “Runnable”
 - ✓ Delphi – Heredar de “TThread”
 - ✓ C#, C, etc
 - ✓ Ruby – Thread.new{CODIGO}
 - ✓ PHP – Heredar de Thread
 - ✓ Javascript – HTML5 Web Workers



Evolución de los Sistemas Operativos

- ✓ Hay situaciones en las que es conveniente tener varios hilos de control en el mismo espacio de direcciones:
- ✓ Ejecutarlos en cuasi-paralelo, como si fueran procesos (casi) separados
- ✓ Compartir el espacio de direcciones

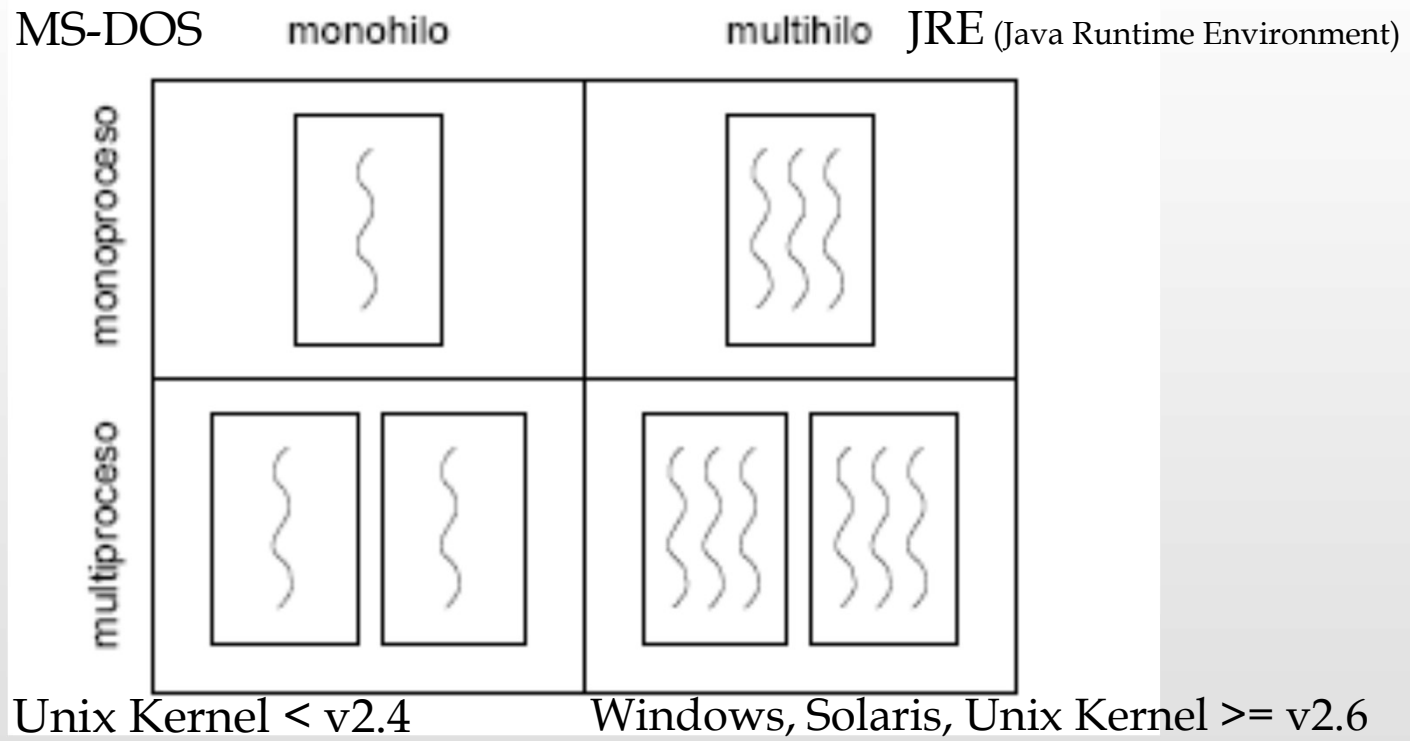


- ✓ Unidad básica de utilización de CPU: Hilo
- ✓ Unidad básica de asignación de RECURSOS: Proceso



Evolución de los Sistemas Operativos

☑ SO Monothreading vs. Multithreading



SO Actuales - Threads

☑ Proceso:

- ✓ Espacio de direcciones
- ✓ Unidad de propiedad de recursos
- ✓ Conjunto de threads (eventualmente uno)

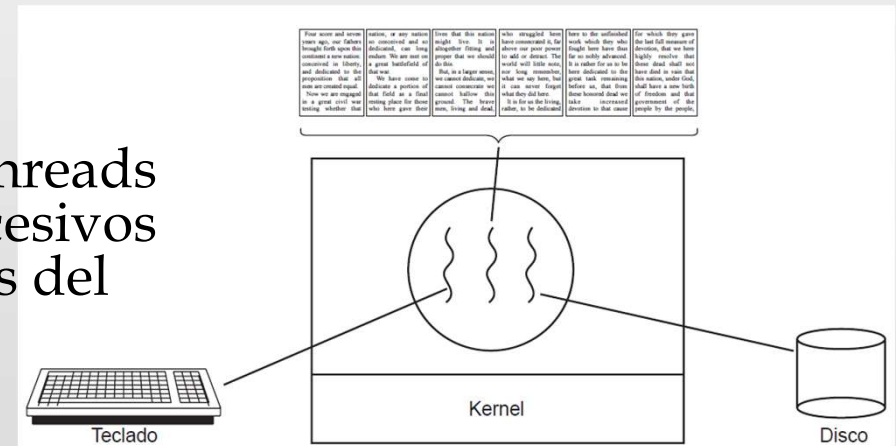
☑ Thread:

- ✓ Unidad de trabajo (hilo de ejecución)
- ✓ Contexto del procesador
- ✓ Stacks de Usuario y Kernel
- ✓ Variables propias
- ✓ Acceso a la memoria y recursos del PROCESO

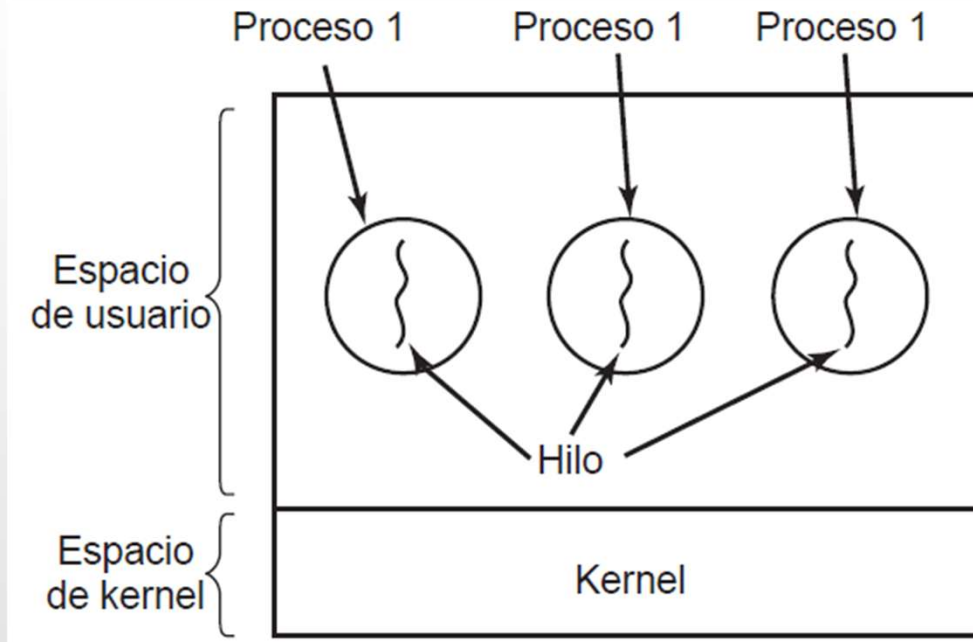


Procesos e Hilos

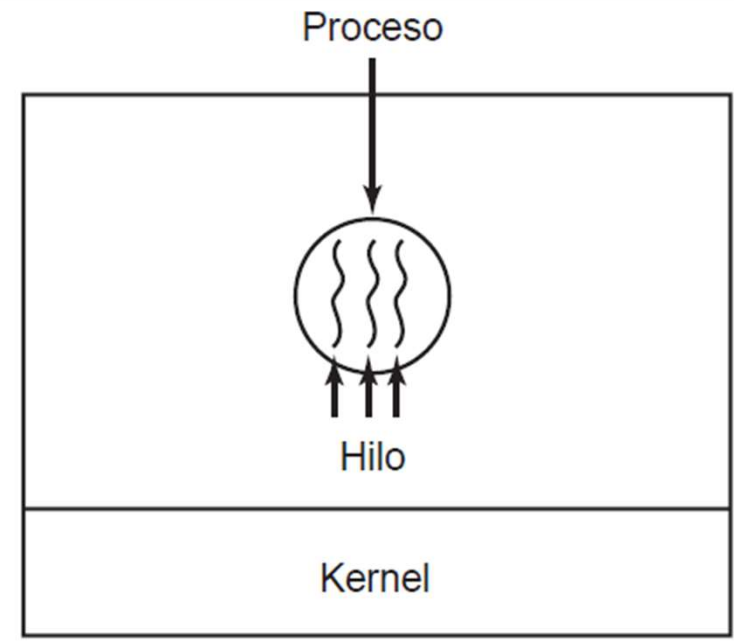
- ☑ Porqué dividir una aplicación en threads?
 - ✓ Respuestas percibidas por los usuarios, paralelismo/ejecución en background
 - ♦ Ejemplo: El servicio de impresión de un editor de texto se ejecuta en background y nos permite seguir editando
 - ✓ Aprovechar las ventajas de múltiples procesadores
 - ♦ Con n CPUs pueden ejecutarse n threads al mismo tiempo
 - ♦ **Pregunta: ¿Dada una aplicación con un único thread, agregar un nuevo procesador hará que esta se ejecute más rápido?**
 - ✓ Características complejas
 - ♦ Sincronización
 - ♦ Escalabilidad: cantidad de threads por proceso? (# de CPU, excesivos cambios de contexto de hilos del mismo proceso...)



Threads



- ✓ 3 Procesos con 1 hilo de ejecución cada uno
- ✓ 3 espacios de direcciones distintos



- ✓ 1 Procesos con 3 hilos de ejecución
- ✓ 1 único espacio de direcciones

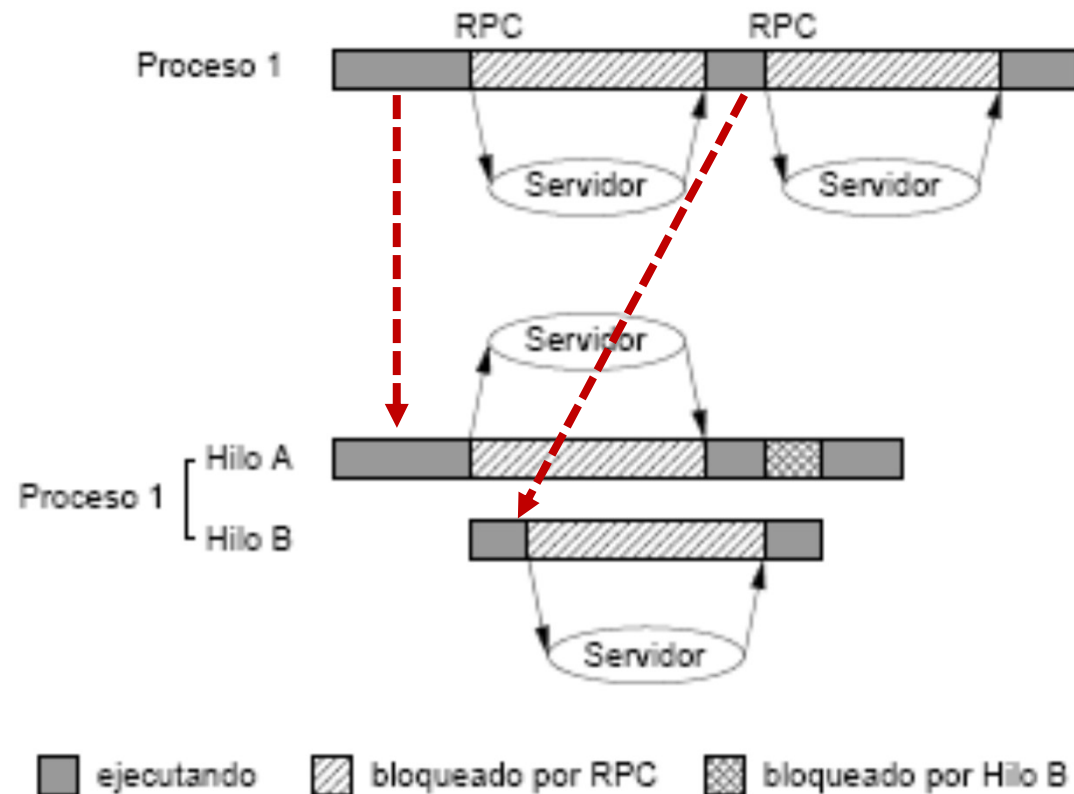


Threads - Ventajas

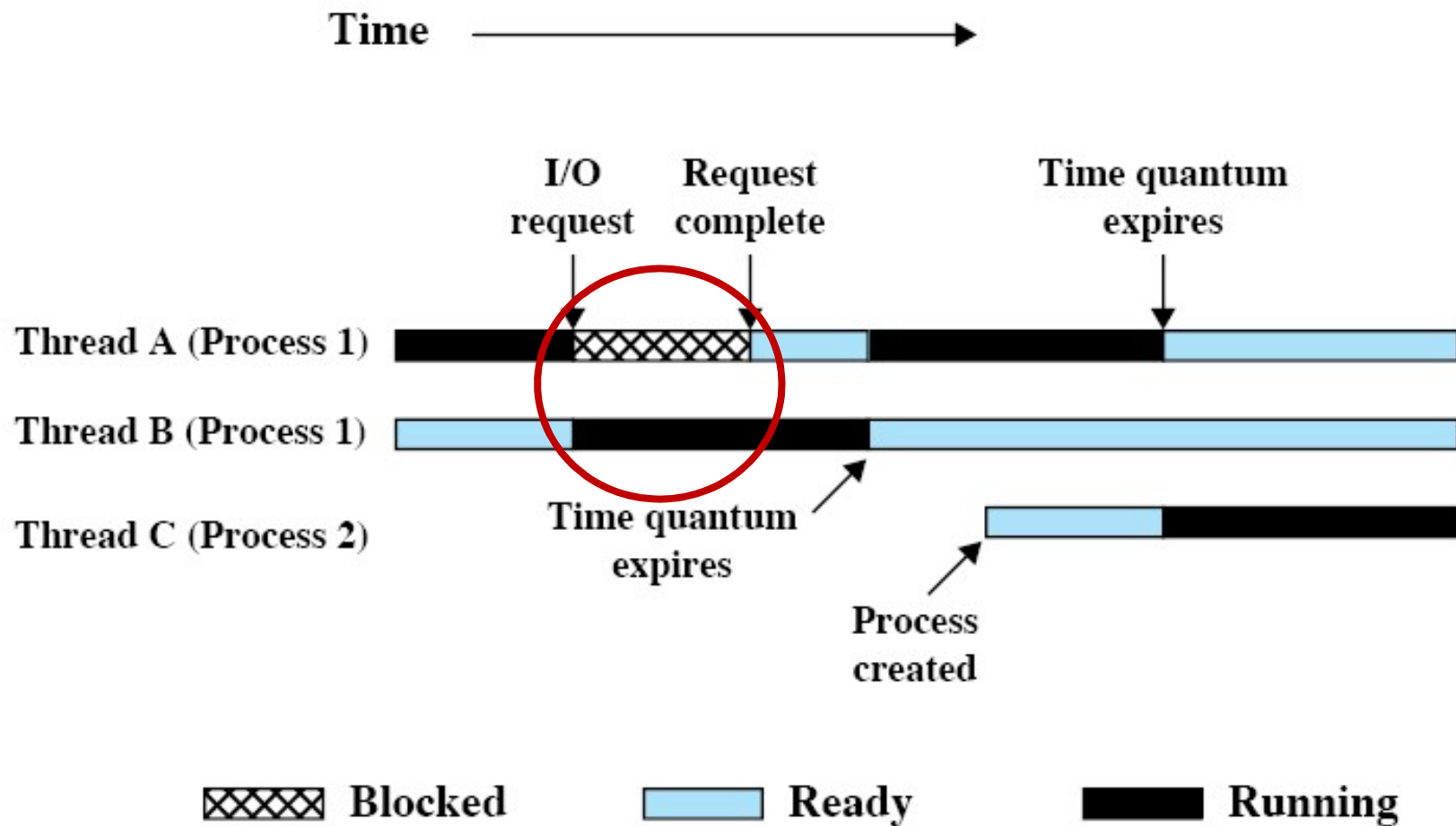
- ✓ Sincronización de Procesos
- ✓ Mejorar tiempos de Respuesta
- ✓ Compartir Recursos
- ✓ Economía
- ✓ Analicemos uso de RPC, o servidor de archivos



Threads– Ejemplo 1



Threads – Ejemplo 2



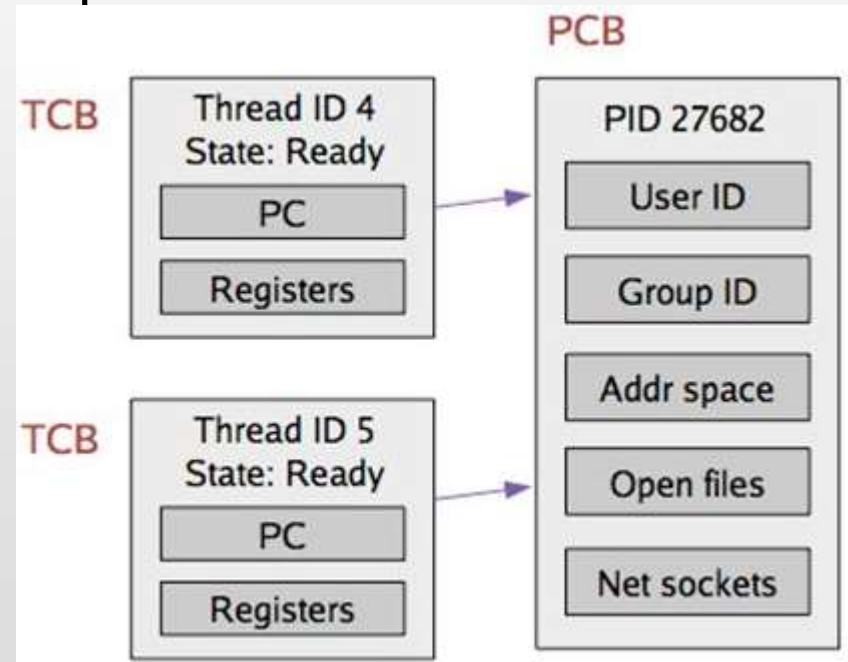
Estructura de un hilo

✓ Cada hilo cuenta con:

- Un estado de ejecución
- Un contexto de procesador
- Stacks (uno en modo usuario y otro en modo kernel)
- Acceso a memoria y recursos del proceso:

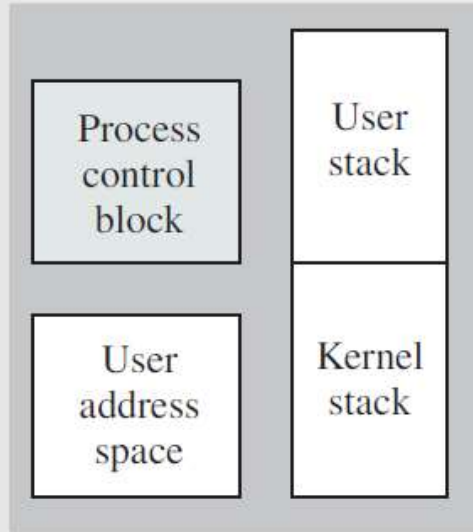
- Archivos abiertos
- Señales
- Código
- Todos estos datos se compartirán con el resto de los hilos del proceso.

➤ TCB - Thread Control Block

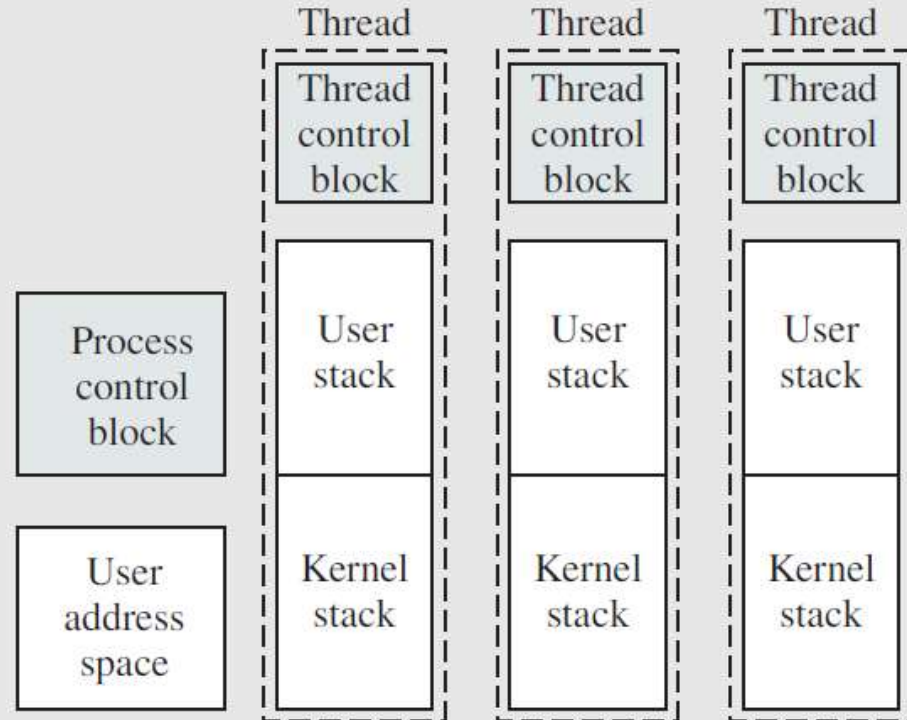


Estructura de un hilo

Single-threaded
process model



Multithreaded
process model



Análisis:

☑ El uso de hilos en reemplazo de los procesos posee ventajas:

➤ Context switch:

- **Procesos:** El SO debe intervenir con el fin de salvar el ambiente del proceso saliente y recuperar el ambiente del nuevo
- **Hilos:** El cambio de contexto solo se realiza a nivel de registros y no espacio de direcciones. Lo lleva a cabo el proceso sin necesidad de intervención del SO

➤ Creación:

- **Procesos:** Implica la creación de un nuevo espacio de direcciones, PCB, PC, etc. Lo lleva a cabo el SO
- **Hilos:** Implica la creación de una TCB, registros, PC y un espacio para el stack. Lo hace el mismo proceso sin intervención del SO



Análisis:

☑ El uso de hilos en reemplazo de los procesos posee ventajas:

➤ **Dstrucción:**

- **Procesos:** El SO debe intervenir con el fin de salvar el ambiente del proceso saliente y eliminar su PCB
- **Hilos:** La tarea se realiza dentro del proceso sin necesidad de intervención del SO

➤ **Planificación:**

- **Procesos:** Es llevada a cabo por el sistema operativo. El cambio implica cambios de contexto continuos
- **Hilos:** Es responsabilidad del desarrollador quien debe planificar sus hilos. Es menos costoso, pero puede traer desventajas aparejadas



Análisis:

☑ El uso de hilos en reemplazo de los procesos posee ventajas:

➤ **Protección:**

- **Procesos:** El SO garantiza la protección a través de distintos mecanismos de seguridad. La comunicación entre ellos implica el uso de técnicas mas avanzadas
- **Hilos:** La protección debe darse desde el lado del desarrollo. Todos los hilos comparten el mismo espacio de direcciones. Un hilo podría bloquear la ejecución de otros



Análisis:

- ☑ ¿Qué realiza el siguiente código?
- ☑ ¿Cuáles son los problemas que podría generar?

```
void print_message_function( void *ptr );
main()
{
pthread_t thread1, thread2;
char *message1 = "Hello";
char *message2 = "World";
    pthread_create( &thread1, pthread_attr_default,
                    (void*)&print_message_function, (void*) message1);
    pthread_create(&thread2, pthread_attr_default,
                    (void*)&print_message_function, (void*) message2);
    exit(0);
}
void print_message_function( void *ptr )
{
char *message;
    message = (char *) ptr;
    printf("%s ", message);
}
```



Análisis:

☑ ¿El siguiente código soluciona el problema anterior?

```
void print_message_function( void *ptr );
main()
{
pthread_t thread1, thread2;
char *message1 = "Hello";
char *message2 = "World";
    pthread_create( &thread1, pthread_attr_default,
        (void*)&print_message_function, (void*) message1);
    sleep(10);
    pthread_create(&thread2, pthread_attr_default,
        (void*)&print_message_function, (void*) message2);
    sleep(10);
    exit(0);
}
void print_message_function( void *ptr )
{
char *message;
    message = (char *) ptr;
    printf("%s ", message);
}
```



Análisis:

- ✓ La solución al problema anterior es la correcta utilización de funciones para la sincronización de hilos:
 - ✓ pthread_delay_np
 - ✓ Uso de semáforos
 - ✓ Más información en:
<http://www.lix.polytechnique.fr/~liberti/public/computing/parallel/threads/threads-tutorial/sect4.html>



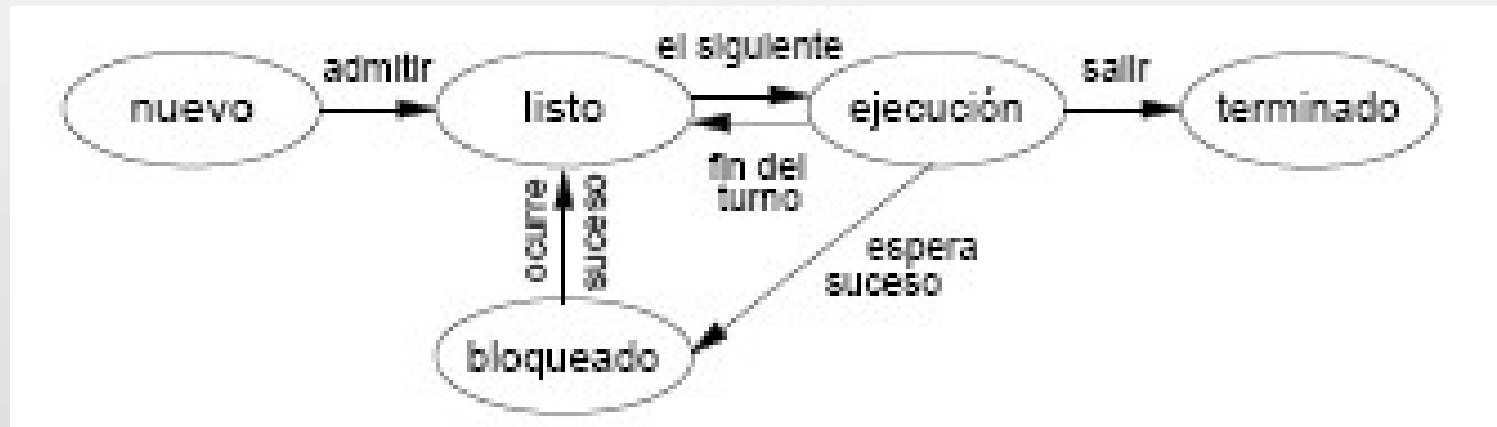
Hilos

- ✓ En los sistemas operativos tradicionales, cada proceso tiene un espacio de direcciones y un solo hilo de control.
- ✓ Hay situaciones en las que es conveniente tener varios hilos de control en el mismo espacio de direcciones que se ejecutan en cuasi-paralelo, como si fueran procesos (casi) separados (excepto por el espacio de direcciones compartido)
- ✓ Unidad básica de utilización de CPU: Hilo
- ✓ Contexto del procesador
- ✓ Stacks de Usuario y Kernel
- ✓ Variables propias



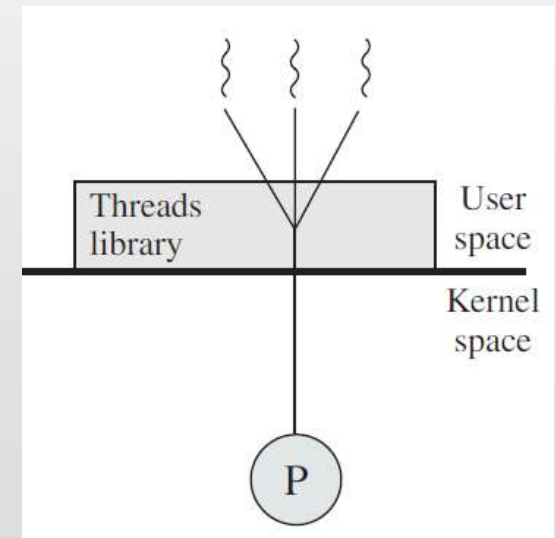
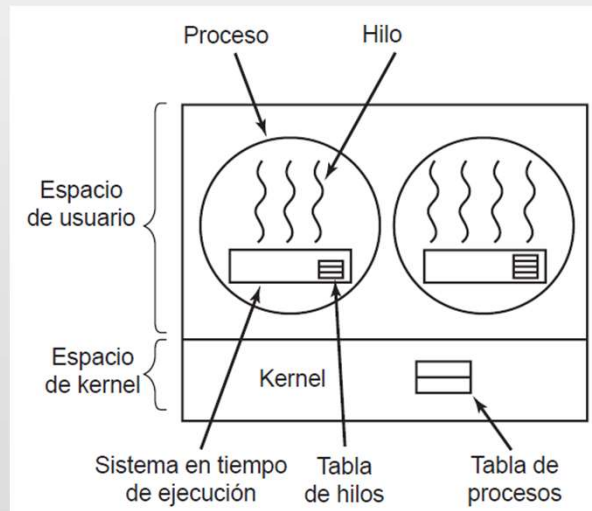
Estados de un Thread

- ✓ Estados Mínimos:
 - Ejecución, Listo y Bloqueado
- ✓ Planificación: sobre los Threads
- ✓ Eventos sobre procesos afectaran todos sus Threads



Tipos - Usuario – ULT

- ✓ User Level Thread
- ✓ La aplicación, en modo usuario, se encarga de la gestión
 - Por medio de una Biblioteca de Threads
- ✓ La Biblioteca deberá brindar funciones para:
 - ✓ Crear, destruir, planificar, etc.
- ✓ El Kernel “no se entera” de la existencia de Threads.
- ✓ Ejemplos:
 - ✓ Java VM
 - ✓ POSIX Threads
 - ✓ Solaris Threads



Tipos - Usuario – ULT

☑ Ventajas

- ✓ Intercambio entre hilos: comparten el espacio de direcciones
- ✓ Planificación independiente: cada proceso los planifica como mas le conviene
- ✓ Podrían reemplazarse llamadas al sistema bloqueantes por otras que no bloqueen
- ✓ Portabilidad: pueden correr en distintas plataformas
- ✓ No requiere cambios para su “existencia”
- ✓ No es necesario que el SO soporte hilos



Tipos - Usuario – ULT

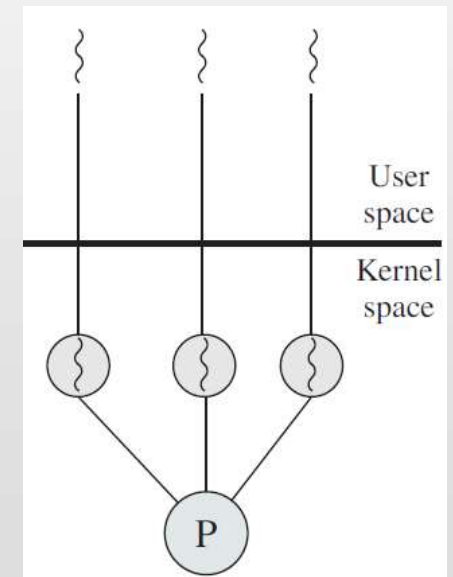
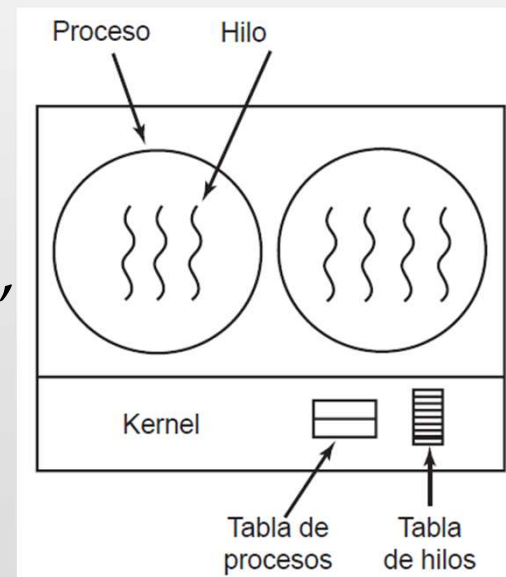
☑ Desventajas

- ✓ No se puede ejecutar hilos del mismo proceso en distintos procesadores
- ✓ Si un hilo produce un Page Fault, todo el proceso se bloquea
- ✓ Un hilo podría monopolizar el uso de la CPU por parte del proceso
- ✓ Bloqueo del proceso durante una System Call bloqueante



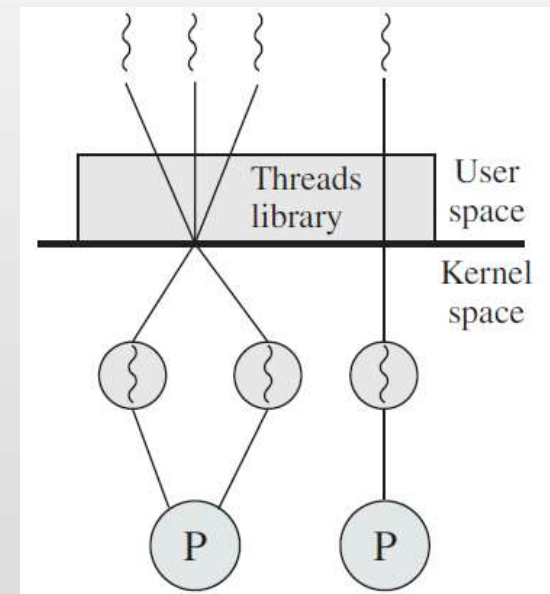
Tipos - Kernel – KLT

- ✓ Kernel Level Thread
- ✓ La gestión completa se realiza en modo Kernel
- ✓ Ventajas
 - ✓ Se puede multiplexar hilos del mismo proceso en diferentes procesadores
 - ✓ Independencia de bloqueos entre Threads de un mismo proceso
- ✓ Desventajas
 - ✓ Cambios de modo de ejecución para la gestión
 - Planificación, creación, destrucción, etc.
- ✓ Ejemplos:
 - ✓ Windows NT/2000
 - ✓ Linux



Tipos de Threads - Combinaciones

- ✓ Es posible combinar ULT y KLT
- ✓ En este tipo de sistemas, la creación de hilos se realiza a nivel de usuario y los mismos son mapeados a una cantidad igual o menor de KLT.
- ✓ La sincronización de hilos en este modelo, permite que un hilo se bloquee y otros hilos del mismo proceso sigan ejecutándose
- ✓ Permite que hilos de usuario mapeados a distintos KLT puedan ejecutarse en distintos procesadores.
- ✓ Este enfoque aprovecha las ventajas de ambos tipos



Modelos de Multithreading

☑ Relación entre ULT y KLT

☑ Tipos

✓ Uno a Uno

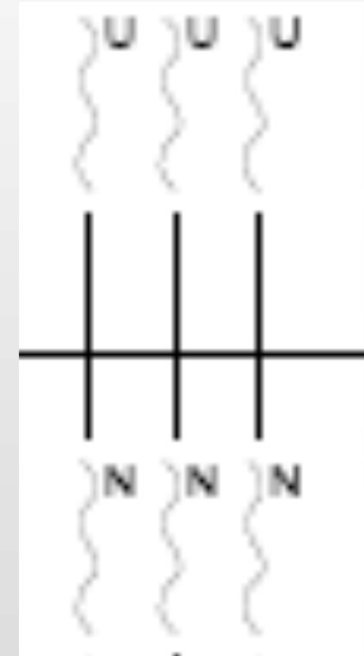
✓ Muchos a Uno

✓ Muchos a Muchos



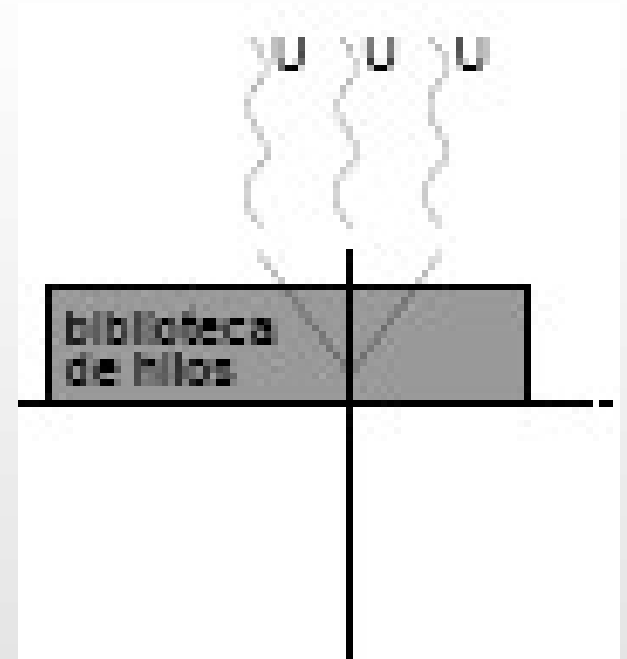
Modelos - Uno a Uno

- ✓ Cada ULT mapea con un KLT
- ✓ Cuando se necesita un ULT se debe crear un KLT
- ✓ Si se bloquea un ULT, otro hilo del mismo proceso puede seguir ejecutándose
- ✓ La concurrencia y/o paralelismo es máximo, ya que cada hilo puede correr en un procesador distinto
- ✓ Introduce un costo alto, ya que cada vez que se crea un hilo de usuario se debe crear un KLT



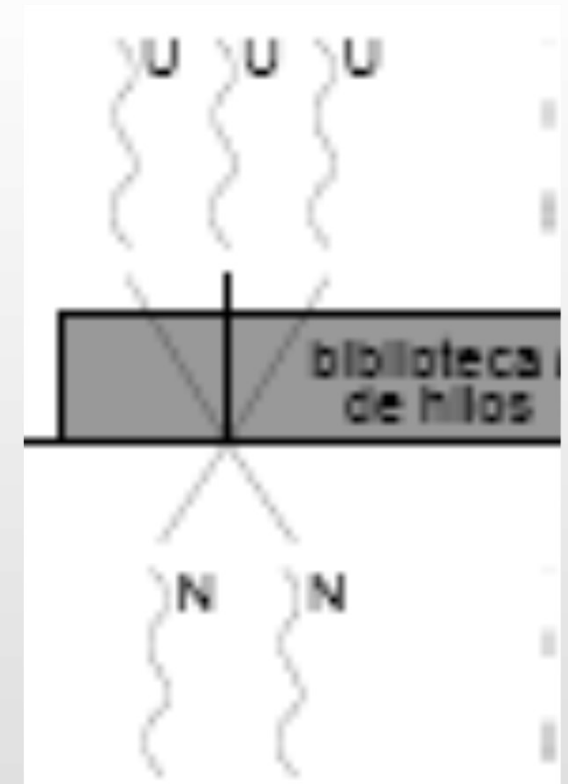
Modelos - Muchos a Uno

- ✓ Muchos ULT mapean a un único KLT
- ✓ Usado en sistemas que no soportan KLT
- ✓ Si se bloquea un ULT, se bloquea el proceso
- ✓ Java sobre un sistema que no soporta KLT



Modelos - Muchos a Muchos

- ✓ Muchos ULT mapean a muchos KLT
- ✓ Este modelo multiplexa los ULT en KLT, logrando un balanceo razonable:
 - ✓ No tiene el costo del modelo 1:1
 - ✓ Minimiza los problemas de bloqueo del modelo M:1



Modelos

Threads: Processes	Description	Example Systems
1:1	Each thread of execution is a unique process with its own address space and resources.	Traditional UNIX implementations
M:1	A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.	Windows NT, Solaris, Linux, OS/2, OS/390, MACH
M:N	Combines attributes of M:1 and 1:M cases.	TRIX

