Hilos, Hebras o Threads

En esta clase:

- Hilos de ejecución o Threads.
- Implementación como Biblioteca Posix Threads (pthread) en Linux

Procesos e Hilos

- Propiedad de los recursos – El proceso incluye un espacio de direccionamiento virtual para ubicar su imagen
- Planificación/Ejecución – Tiene un camino de ejecución que puede estar entrelazado con el de otros procesos.
- El sistema operativo puede tratar ambas características de forma independiente.

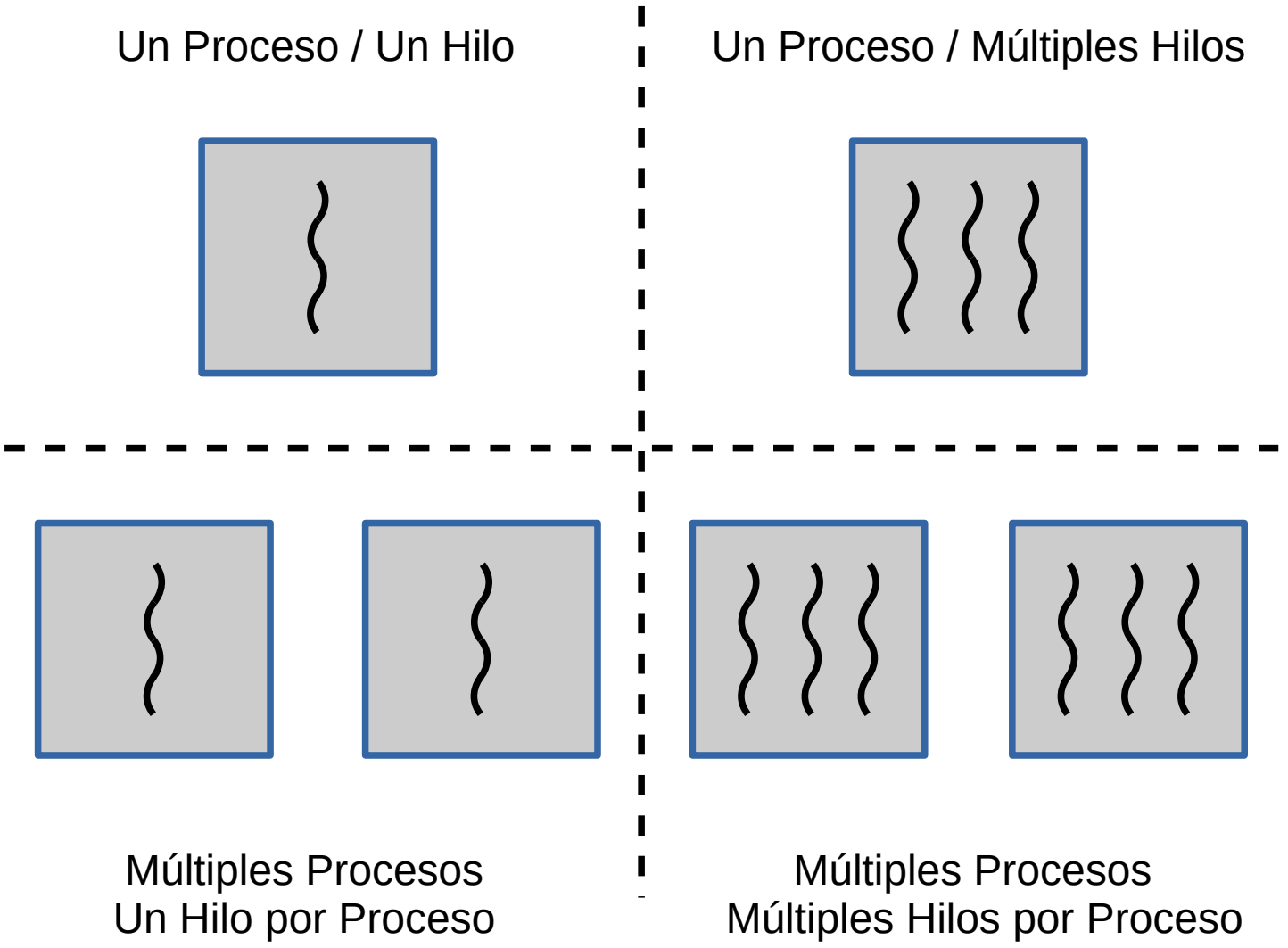
Procesos e Hilos

- La unidad de despacho (ejecución por un procesador) se refiere como *hilo*, *hebra* o *proceso liviano (Thread – Lightweight Process – LWP)*
- La propiedad de los recursos se refiere a un *Proceso* o *Tarea*. *(Task)*

Procesos e Hilos

- Sistemas Operativos con Procesos MultiHebra o de múltiples Hilos:
  - El SO soporta múltiples hilos de ejecución dentro de un único proceso
- Ejemplos:
  - MS-DOS soportaba un único hilo dentro de un único proceso.
  - UNIX original soportaba múltiples procesos de usuario pero un único hilo dentro de cada proceso.
  - Windows, Solaris, Linux, Mach, and OS/2 soportan múltiples hilos dentro de múltiples procesos.

Procesos e Hilos



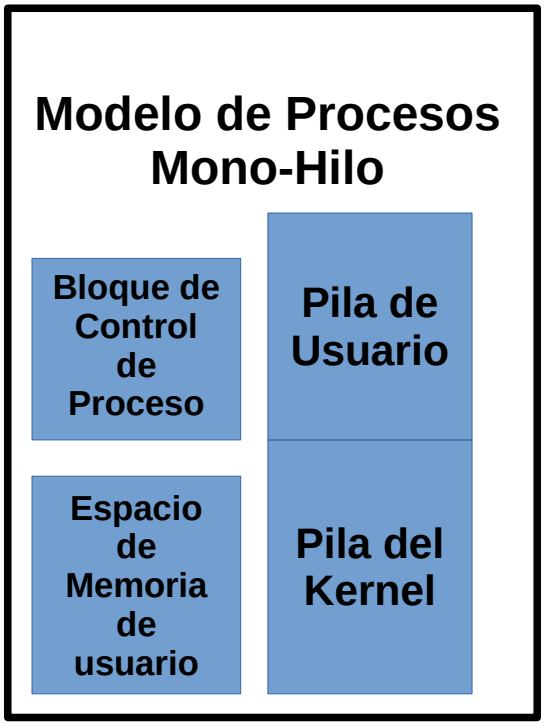
Procesos e Hilos

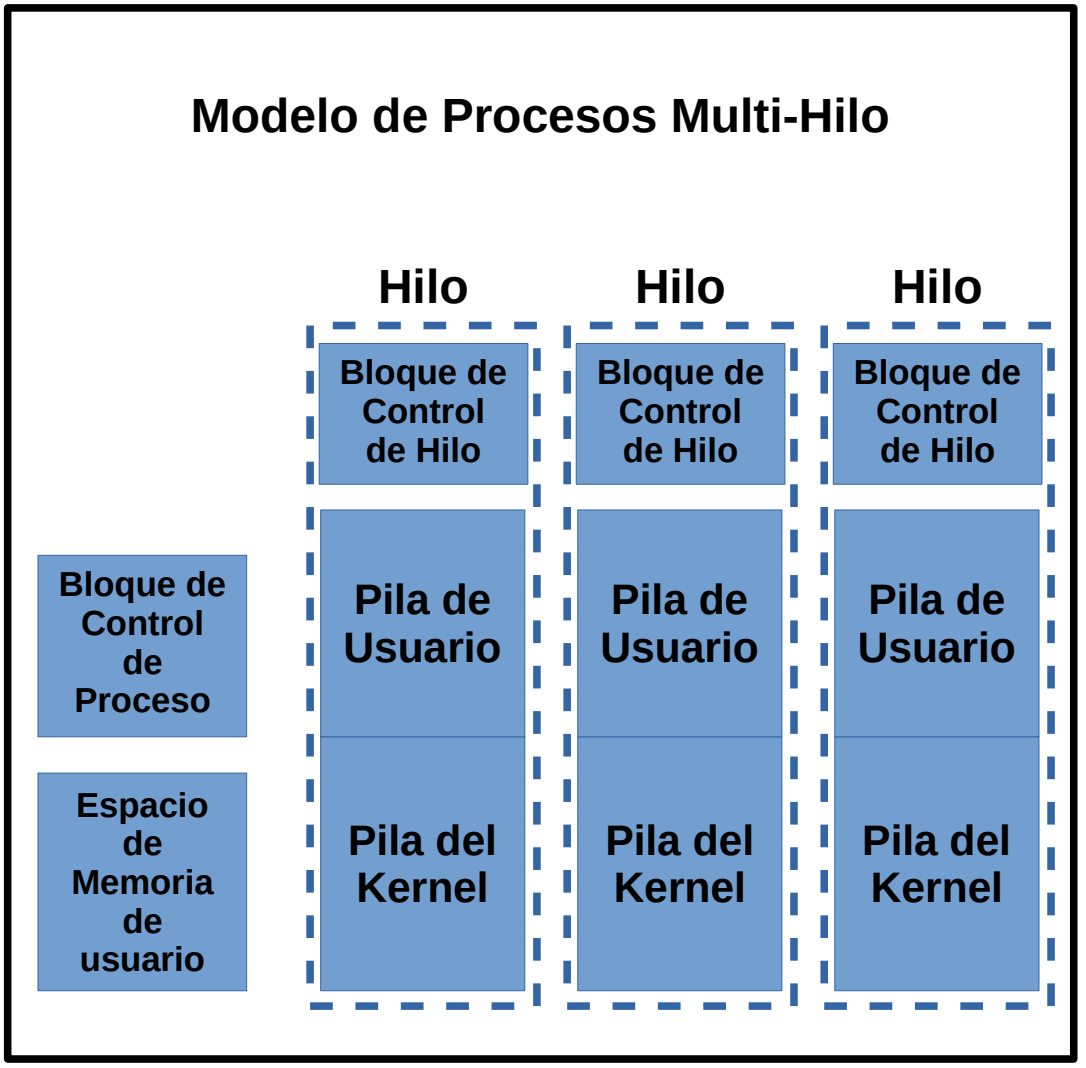
- PROCESO:
  - Está definido en una Imagen de Proceso, que está contenido en un espacio de Memoria Virtual, administrada por el SO
  - El SO le provee acceso protegido a:
    - Procesadores
    - Otros procesos
    - Archivos
    - Recursos de Entrada/Salida (E/S).

Procesos e Hilos

- HILO o HEBRA:
  - Posee un estado de ejecución (Corriendo, Listo, Bloqueado, etc.).
  - El contexto se guarda cuando no está corriendo.
  - Posee una pila de ejecución.
  - Se provee almacenamiento estático para las variables locales de cada hilo
  - Puede acceder a la memoria y recursos del proceso que la originó
  - Todas las hebras de un proceso comparten esto.

Modelos de procesos Mono-Hilo y Multi-Hilo





Beneficios del uso de Hilos

- Lleva menos tiempo crear un nuevo hilo que un proceso
- Lleva menos tiempo terminar un nuevo hilo que un proceso
- Lleva menos tiempo conmutar entre dos hilos de un mismo proceso.
- Desde que las hilos dentro de un proceso comparten memoria y recursos, pueden comunicarse entre sí sin necesidad de recurrir al kernel

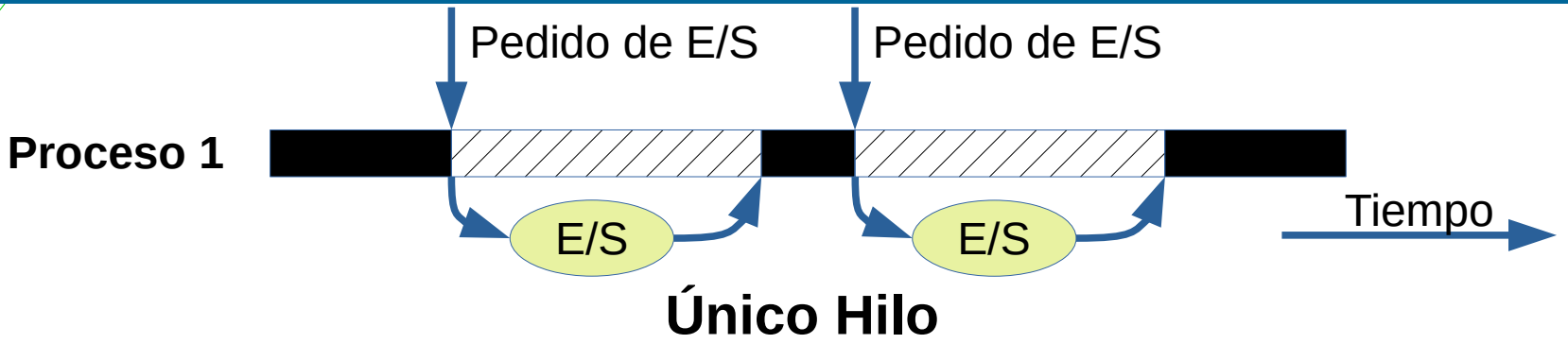
Uso de hebras en un sistema Mono-usuario multiproceso

- Permite:
  - Trabajar en Foreground y Background
  - Procesamiento asincrónico
  - Velocidad de ejecución
  - Estructura modular de programas.
- Suspender un proceso, involucra suspender todas los hilos del proceso, ya que comparten el mismo espacio de direccionamiento
- Cuando termina un proceso, terminan todos los hilos dentro del mismo

Estados de los Hilos

- Estados asociados con un cambio en el estado de los Hilos.
  - Brotando / criando (Spawn)
    - Genera otro hilo
  - Bloqueado
  - Desbloqueado
  - Terminado
    - Liberar contexto de registros y pila de ejecución

Pedido de E/S con uno y múltiples hilos

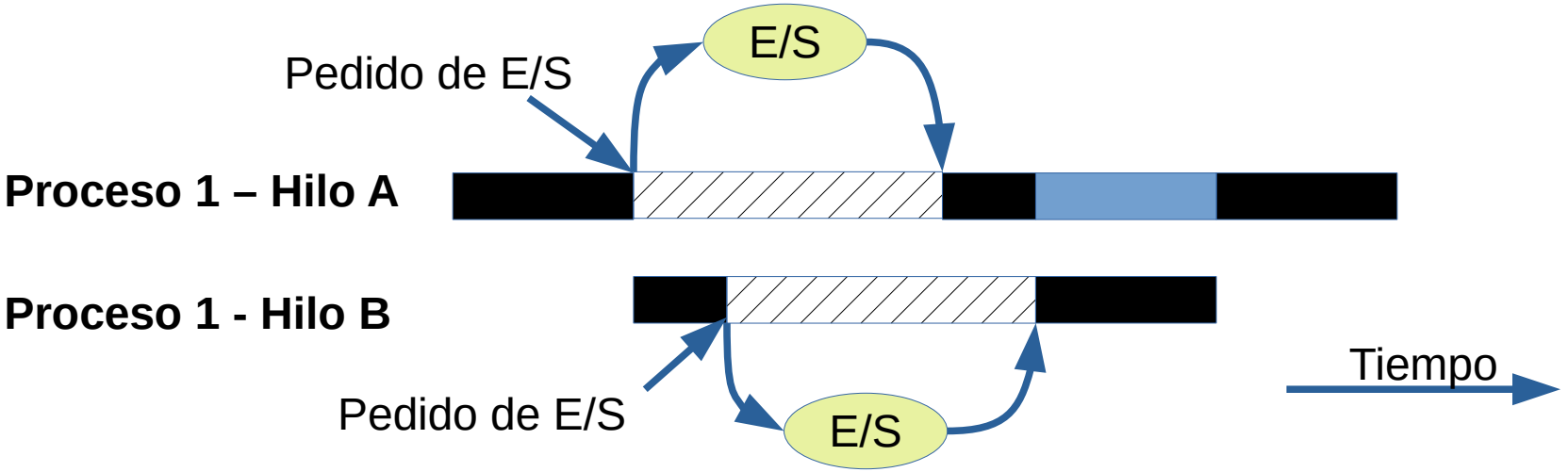


Único Hilo

Bloqueado – Esperando E/S

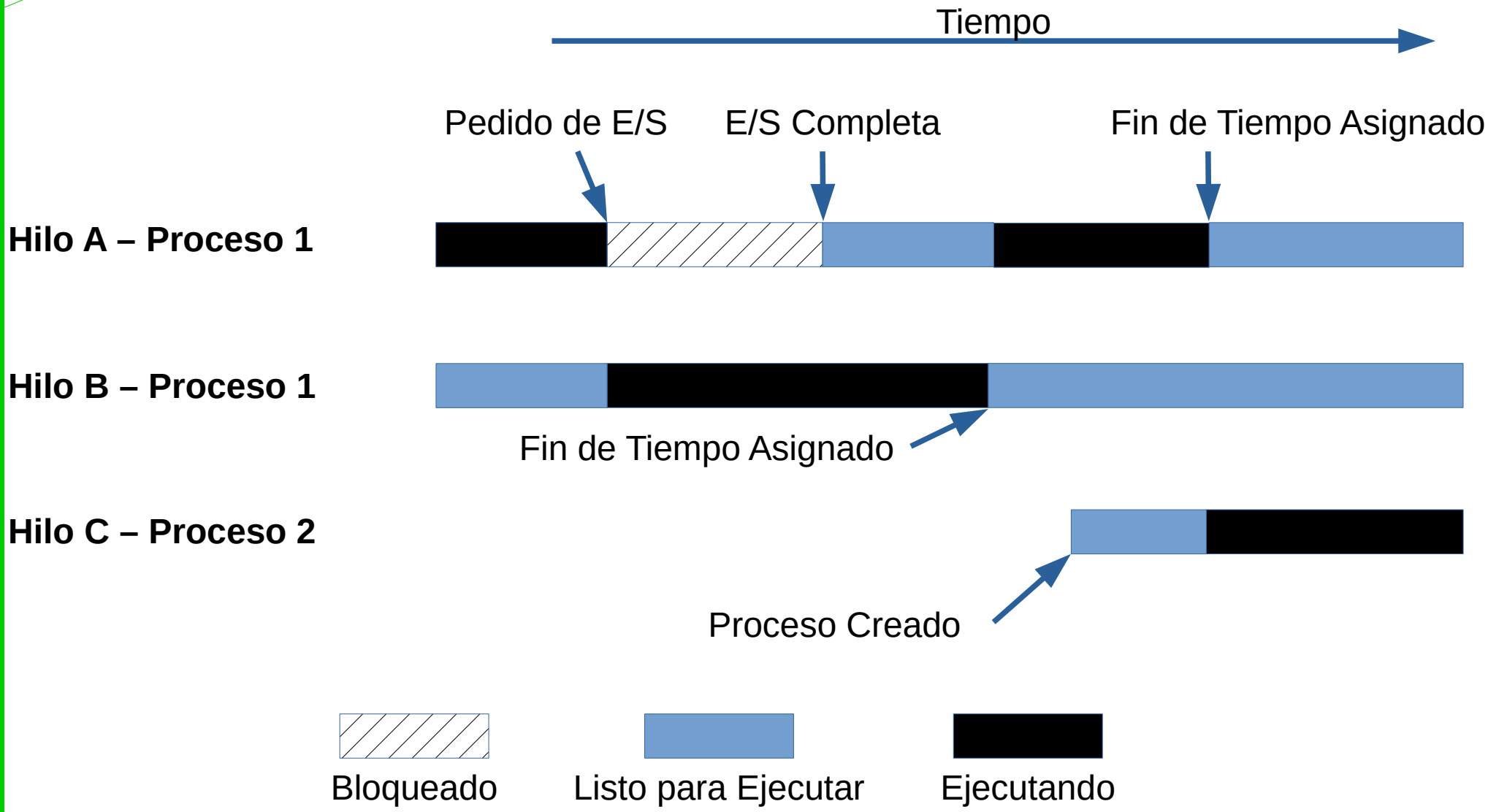
Bloqueado – Esperando Procesador en uso por el Hilo B

Ejecutando



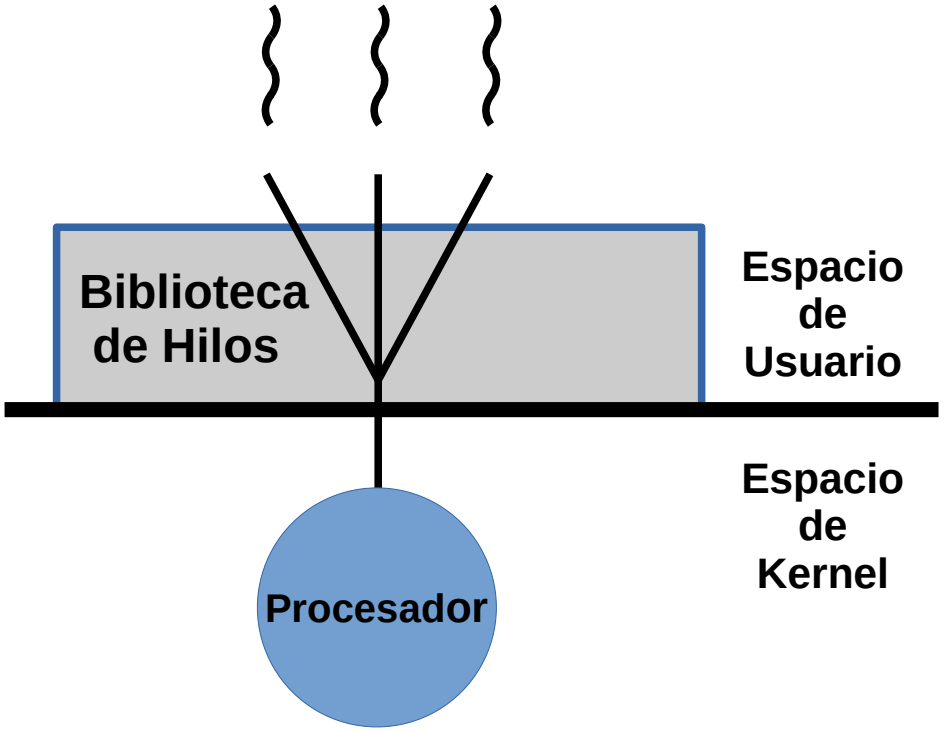
Múltiples Hilos

Multihilo con procesador único

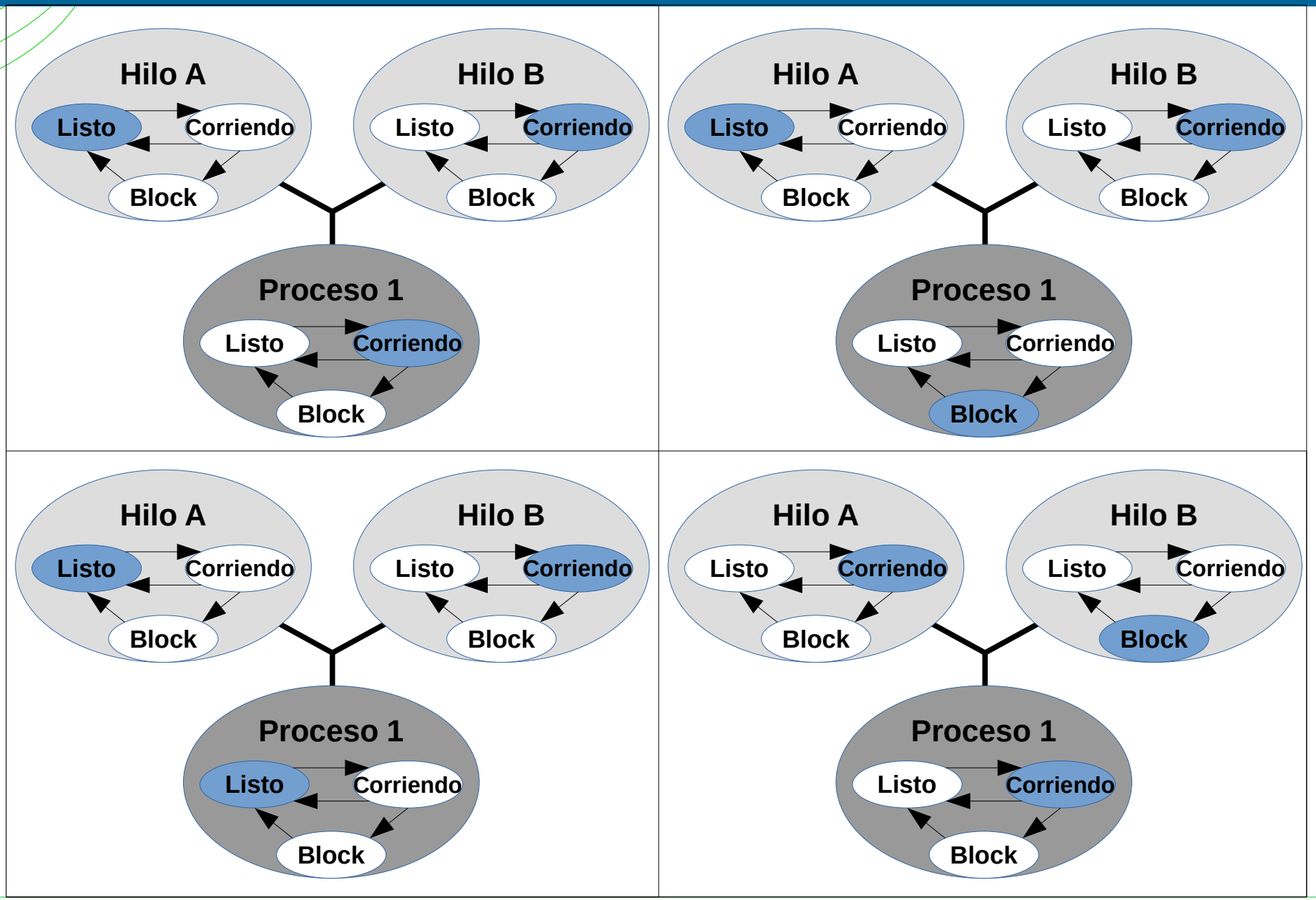


Tipos de Hilos: Hilos a nivel de Usuario

- Hilos a nivel de Usuario
  - Todo el manejo de hilos lo hace la aplicación
  - El kernel no tiene información acerca de la existencia de hilos.

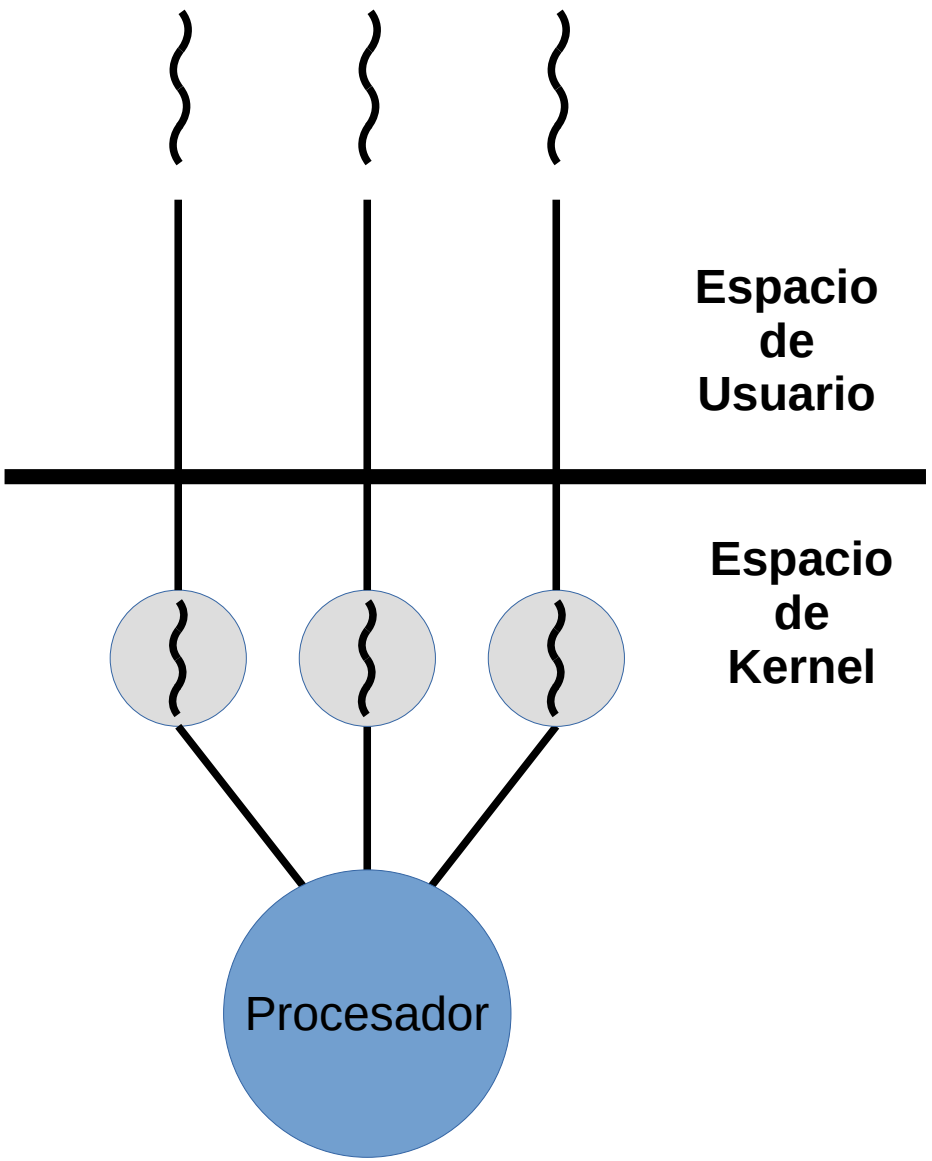


Relación entre procesos e hilos a nivel de Usuario



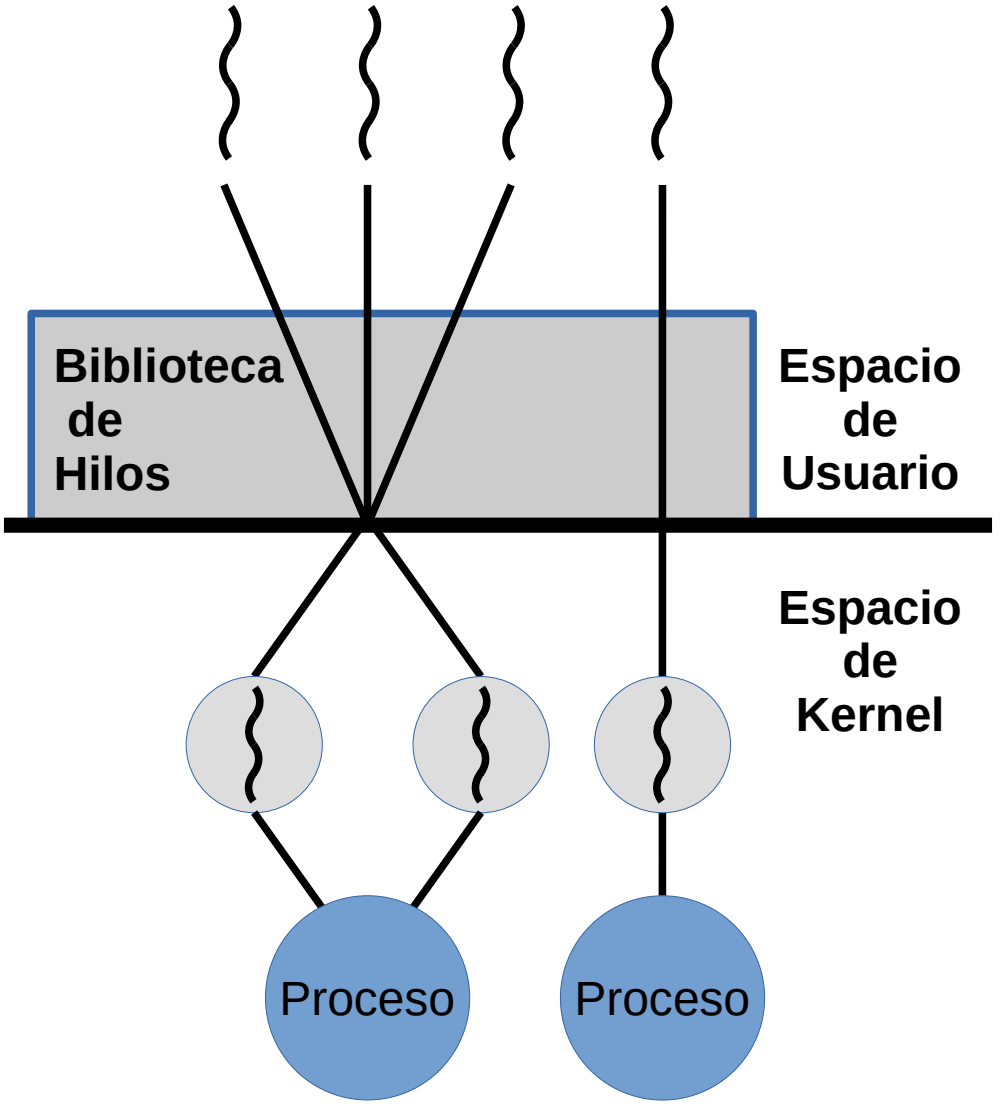
Tipos de Hilos: Hilos a nivel de Kernel

- Un ejemplo es Windows
- El Kernel mantiene información acerca del contexto de los procesos y sus hilos
- La planificación se hace en base a los hilos



Tipos de Hilos: Métodos combinados

- Por ejemplo, Solaris
- La creación de hilos se realiza en el espacio de usuario
- La mayor parte de la tarea de planificación y sincronización se realiza dentro de la aplicación
- Para ello se usan bibliotecas de funciones.



Utilización de Hilos, Hebras o Threads

Biblioteca Pthreads

- La biblioteca de pthreads es una biblioteca que cumple los estándares POSIX y que nos permite trabajar con distintos hilos de ejecución (threads) al mismo tiempo.
- La creación de threads se hace mediante las funciones de la biblioteca pthread mientras que para crear procesos usaremos la llamada al sistema fork(), que se encuentra en todos los sistemas unix.
- Ya que pthreads es una biblioteca POSIX, se podran portar los programas hechos con ella a cualquier sistema operativo POSIX que soporte threads.

  POSIX (Portable Operating System Interface) es un estándar IEEE destinado a mantener compatibilidad entre sistemas Operativos

Uso de la biblioteca pthread

- Para crear programas que hagan uso de la biblioteca pthreads se necesita verificar que esta está instalada en el sistema. La mayoría de las distribuciones Linux la trae y se instala cuando instalamos paquetes de desarrollo de aplicaciones
- Se debe compilar el programa y enlazarlo con la biblioteca pthreads.
- Si estamos usando como compilador GNU gcc se debe usar el siguiente comando:

```
gcc programa_con_pthreads.c -o programa_con_pthreads -lpthread
```

Operaciones con Hilos

- Creación
- Terminación
- Sincronización (Bloqueo, reunión)
- Planificación
- Administración de Datos
- Interacción de Procesos

Propiedades de las hebras

- Una hebra no sabe quien la creó ni mantiene una lista de hebras que crea
- Todas las hebras dentro de un proceso comparten:
  - El mismo espacio de direcciones
  - Instrucciones del proceso
  - La mayoría de los Datos
  - descriptores de archivos abiertos
  - Manejadores de señales y señales
  - CWD (current working directory)
  - Identificador de usuario y grupo

Propiedades de las hebras

- Cada hebra posee un único:
  - Identificador de hebra
  - Conjunto de registros
  - Puntero de pila (Stack Pointer)
  - Pila para variables locales y direcciones de retorno de funciones.
  - Máscara de señales
  - Prioridad
  - Valor de retorno, pero no establecen errno
- Las funciones de pthread devuelven "0" si todo está OK.

Creación y manipulación de Hilos

- Las principales funciones que se utilizan son:
  - pthread_create
  - pthread_join
  - pthread_detach
  - pthread_exit
- También se usan los tipos de datos:
  - pthread_t
  - pthread_attr_t

Otras funciones y tipos de datos que se utilizan en pthreads

- Funciones:
  - `pthread_self()`
  - `pthread_kill()`
  - `pthread_mutex_init()`
  - `pthread_mutex_lock()`
  - `pthread_mutex_unlock()`
  - `pthread_mutex_destroy()`
  - `pthread_mutexattr_init()`
  - `pthread_mutexattr_settype()`
  - `pthread_mutex_trylock()`
- Tipos de datos:
  - `pthread_mutex_t`
  - `pthread_mutexattr_t`

Tipos de datos

- `pthread_t` : tipo predefinido para identificar una hebra, análogo a pid_t para procesos.
- `pthread_attr_t` tipo predefinido para fijar atributos de la hebra

pthread_create

```
int pthread_create (pthread_t *thread,
    pthread_attr_t *attr,
    void *(*start_routine)(void *),
    void *arg
    )
```

- `thread:` devuelve el ID de la hebra
- `attr:` Es un parámetro del tipo `pthread_attr_t` y que se debe inicializar previamente con los atributos que queramos que tenga el thread. Entre los atributos hay la prioridad, el quantum, el algoritmo de planificación que queramos usar, etc. Si pasamos como parámetro aquí `NULL`, la biblioteca le asignará al thread unos atributos por defecto (RECOMENDADO).
- `arg:` Es un puntero al parámetro que se le pasará a la función. Puede ser NULL si no queremos pasarle nada a la función.
- En caso de que todo haya ido bien, la función devuelve un 0 o un valor distinto de 0 cuando hubo algún error.

pthread_create

```
int pthread_create (pthread_t *thread,
    pthread_attr_t *attr,
    void *(*start_routine)(void *),
    void *arg
    )
```

- `start_routine:` Aquí pondremos la dirección de la función que queremos que ejecute el thread. La función debe devolver un puntero genérico (`void *`) como resultado, y debe tener como único parámetro otro puntero genérico.
- La ventaja de que estos dos punteros sean genéricos es que podremos devolver cualquier cosa que se nos ocurra mediante los castings (conversiones) de tipos necesarios.
- Si necesitamos pasar o devolver más de un parámetro a la vez, se puede crear una estructura y meter allí dentro todo lo que necesitemos. Luego pasaremos o devolveremos la dirección de esta estructura como único parámetro. (ver código de ejemplo)

pthread_join

```
int pthread_join(
        pthread_t th,
        void **thread_return
        )
```

- Esta función suspende el thread llamante hasta que no termine su ejecución el thread indicado por `th`. Además, una vez éste último termina, pone en `thread_return` el resultado devuelto por el thread que se estaba ejecutando.
- `th`: Es el identificador del thread que queremos esperar, y es el mismo que obtuvimos al crearlo con `pthread_create`.
- `thread_return:` Es un puntero a puntero que apunta (valga la redundancia) al resultado devuelto por el thread que estamos esperando cuando terminó su ejecución. Si este parámetro es NULL, le estamos indicando a la biblioteca que no nos importa el resultado.
- Devuelve 0 en caso de todo correcto, o valor diferente de 0 si hubo algún error.

pthread_detach

```
int pthread_detach(pthread_t th)
```

- Esta función le indica a la biblioteca que no queremos que nos guarde el resultado de la ejecución del thread indicado por `th`. Por defecto la biblioteca guarda el resultado de ejecución de todos los threads hasta que nosotros hacemos un `pthread_join` para recoger el resultado.
- Es por eso que si no nos interesa el resultado de los threads tenemos que indicarlo con esta función. Así una vez que el thread haya terminado la biblioteca eliminará los datos del thread de sus tablas internas y tendremos más espacio disponible para crear otros threads (IMPORTANTE)
- `th:` Es el identificador del thread
- Devuelve 0 en caso de que todo haya ido bien o diferente de 0 si hubo error.

pthread_exit

`void pthread_exit(void *retval)`

Esta función termina la ejecución del thread que la llama.

`retval`: Es un puntero genérico a los datos que queremos devolver como resultado. Estos datos serán recogidos más tarde cuando alguien haga un `pthread_join` con nuestro identificador de thread.

No devuelve ningún valor.