

## Tema 2. Procesos e hilos

### 1. Procesos

Un proceso es fundamentalmente un programa en ejecución. Un programa o tarea es una unidad inactiva. Un programa no es un proceso.

Definiciones de proceso:

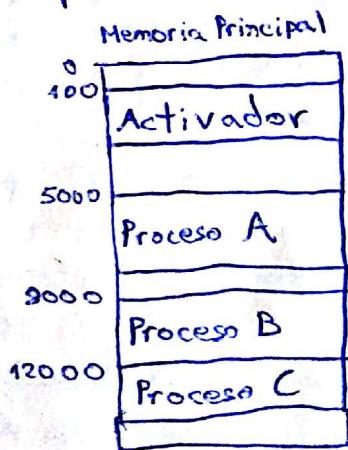
- Un programa en ejecución .
- Una instancia de un programa ejecutándose en un computador
- La entidad que se puede asignar o ejecutar en un procesador.
- Una unidad de actividad caracterizada por un solo hilo secuencial de ejecución, un estado actual, y un conjunto de recursos del sistema asociados.

Un solo procesador puede compartirse entre varios procesos o trabajos sólo si el S.O. cuenta con una política de planificación así como con un algoritmo de planificación.

#### 1.1. Trazas de los procesos

Para que un programa ejecute, se debe crear un proceso para dicho programa. Para que un proceso se ejecute, su conjunto de instrucciones o al menos parte de ellas, deben estar cargadas en memoria principal.

Se puede caracterizar el comportamiento de un determinado proceso, listando la secuencia de instrucciones que se ejecutan para dicho proceso. A la lista se le denomina traza del proceso.



(1)

## 2. Control de procedimientos mediante pila

Una pila es un conjunto ordenado de elementos. El número de elementos de la pila o longitud de la pila es variable. También se conoce como lista LIFO. Cada elemento ordenado contiene información sobre las rutinas que se invocan en un programa.

La mayoría de las veces el bloque de posiciones está parcialmente lleno con elementos y el resto está disponible para el crecimiento de la pila.

Por cada llamada que se realice desde el proceso en ejecución a una función o subrutina en la pila se almacenará lo siguiente:

- La dirección de retorno, que es la dirección de memoria de la siguiente instrucción de código donde retornar después de ejecutarse la función.

- Los datos pasados como parámetros a la función.

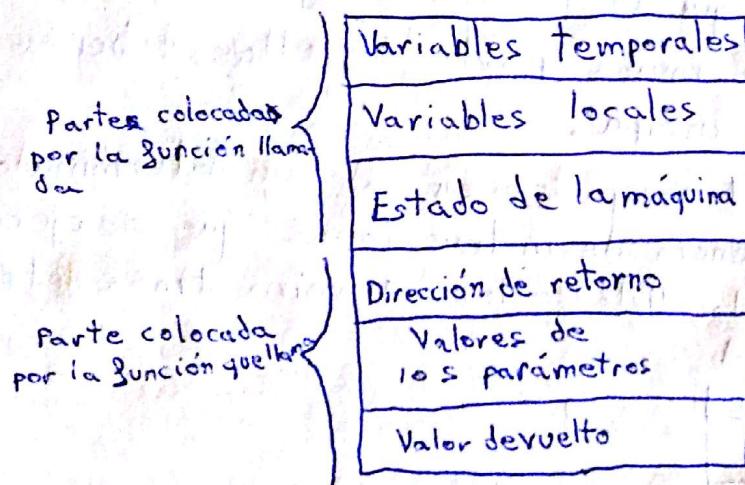
- El valor de retorno de la función (int, float, char...)

- Las variables locales utilizadas por la función llamada.

- " " temporales " " " "

- El estado de la máquina.

### Registro de activación



Durante la ejecución de un proceso éste utiliza varias zonas de memoria principal diferenciadas:

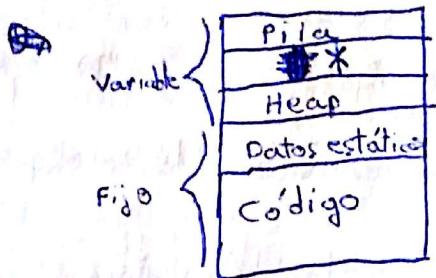
- La pila de llamadas (call stack): Almacenará principalmente los parámetros de las funciones, variables locales y variables de retorno.

El procesador maneja dos registros especiales referentes a la pila:

- Stack Pointer (SP): Dirección de la cima de la pila.

- Frame Pointer (FP): Dirección base del registro de activación de la función activa.

- El área de datos dinámicos o montón (heap)
- El área de datos estáticos: Variables globales y estáticas, reservando el espacio justo ya que se conoce en tiempo de compilación.
- El área del código: Se usa para almacenar el código fuente e instrucciones máquina. Se reserva el espacio justo.



Una vez que el S.O. ha reservado las zonas de memoria y ha cargado el código del ejecutable, el programa está listo para ser ejecutado. El S.O. indicará a la CPU que debe apuntar con su registro de puntero de instrucciones, a la primera instrucción del código (main()) y que debe apuntar con su <sup>puntero</sup> de pila al primer registro de activación reservado en la pila.

### Procedimiento de llamada:

- Se reserva espacio.
- Se almacena el valor de los parámetros que se pasan a la función llamada.
- Se almacena el valor de la dirección de retorno donde ejecutar cuando finalice la función.
- Se almacena el estado de la máquina (incluye SP y FP)
- Se almacenan las variables locales y temporales.
- Comienza la ejecución del código de la función llamada.

### Procedimiento de retorno

- Se asigna el valor devuelto.
- Se restaura el contenido del estado de la máquina. Modifica el valor de FP y SP.
- Se restaura el valor de la dirección de retorno de la función que llama.
- Se devuelve el control a la función que llama.
- Se almacena el valor devuelto en la variable local o temporal de la función que llama.

### 3. Bloque de control de procesos o BCP

Mientras el proceso está en ejecución, se puede caracterizar por una serie de elementos cuyo conjunto se llama contexto de ejecución. Esto es un conjunto de datos por el cual el S.O. es capaz de supervisar y controlar un proceso.

- Identificación: Un identificador único asociado al proceso para distinguirlo del resto de procesos. Puede almacenar el identificador del proceso padre creador de este proceso y el identificador del usuario del proceso, dependiendo del sistema.
- Estado: Si el proceso está actualmente corriendo, está en el estado Ejecución.

• Información de planificación: Nivel de prioridad relativo al resto de procesos.

• Descripción de los segmentos de memoria asignados al proceso.

Espacio de direcciones o límites de memoria asignada al proceso.

• Punteros a memoria.

• Datos de contexto: Presentes en los registros del procesador cuando el proceso está corriendo. Almacena todo lo necesario para poder continuar la ejecución del proceso cuando el planificador del S.O. lo decide.

• Información de estado de E/S y recursos asignados.

• Comunicación entre procesos

• Información de auditoría.

La información de la lista se almacena en una estructura de datos que se llama BCP que contiene suficiente información de forma que es posible interrumpir un proceso cuando está corriendo y posteriormente restaurar su estado de ejecución como si no hubiese sufrido interrupción alguna. El BCP es la herramienta clave que permite al S.O. dar soporte a múltiples procesos.

En algunos sistemas como UNIX cada proceso tiene asociado un BCP que está almacenado en una lista simplemente enlazada, llamada tabla de procesos. Se llama imagen del proceso al conjunta del programa, datos, pila, montículo y BCP. Para ejecutar un proceso, la imagen del proceso completa se debe cargar en memoria principal = al menos en memoria virtual.

La conmutación entre unos procesos y otros es lo que da lo-

gar a la multiprogramación, y el programa que se encarga de comutar se llama activador o dispatcher, proceso que forma parte del núcleo, y cargado en memoria principal, pasa a ejecución cuando se producen alarmas de temporización, cuando hay interrupciones, etc. Es el planificador el que indica al dispatcher, qué proceso es el que debe introducir en la CPU. Los procesos, según en el estado que están, se encuentran en unas colas de planificación u otras. Cada cola se puede planificar por un algoritmo de planificación diferente.

#### 4. Creación de procesos

Hay 4 eventos principales que provocan la creación de procesos:

1. El arranque del sistema: Cuando se arranca un S.O. se crean varios procesos. Algunos en primer plano, otros en segundo plano, los que permanecen en 2º plano para manejar ciertas actividades como e-mail, páginas Web, gestor de impresión, inicialización y control del hardware se conocen como demonios. Los sistemas grandes, tienen docenas de ellos.

2. La ejecución, desde un proceso, de una llamada al sistema para creación de procesos: A menudo, un proceso en ejecución emitirá llamas al sistema para crear uno o más procesos nuevos, para que le ayuden a realizar su trabajo p.e `fork()` e hilos en POSIX. Es útil crear procesos cuando el trabajo a realizar se puede formular fácilmente, en términos de procesos interactivos relacionados entre sí. En un multiprocesador, al permitir que cada proceso se ejecute en una CPU distinta, también se puede hacer el trabajo con mayor rapidez.

3. Una petición de usuario para crear un proceso: En los sistemas interactivos, los usuarios pueden iniciar un programa `terminal` que inicia un proceso y ejecuta el programa seleccionado. Un terminal de GNU/Linux, no es más que un proceso esperando un comando, el cual internamente lo interpretará con una llamada a `fork()`.

4. El inicio de un trabajo por lotes: La última situación se aplica solo a los sistemas de procesamiento por lotes. Cuando el S.O. decide que tiene los recursos para ejecutar un trabajo de la cola, crea un proceso y ejecuta el siguiente trabajo

(3)

de la cola de entrada. Una vez que el S.O. decide crear un proceso procederá de la siguiente manera:

1. Asignar un identificador de proceso único al proceso. Se añade una nueva entrada a la tabla de procesos, que contiene una entrada por proceso y se añade identificador al mismo.
2. Reservar espacio para proceso. El S.O. debe saber cuanta memoria es necesaria para el espacio de direcciones privado y para la pila y montículo de usuario. Si un proceso es creado por otro proceso, el proceso padre puede pasar los parámetros requeridos por el S.O. como parte de la solicitud de creación del proceso.
3. Actualización del BCP. Es necesario actualizar toda la información guardada en el BCP.

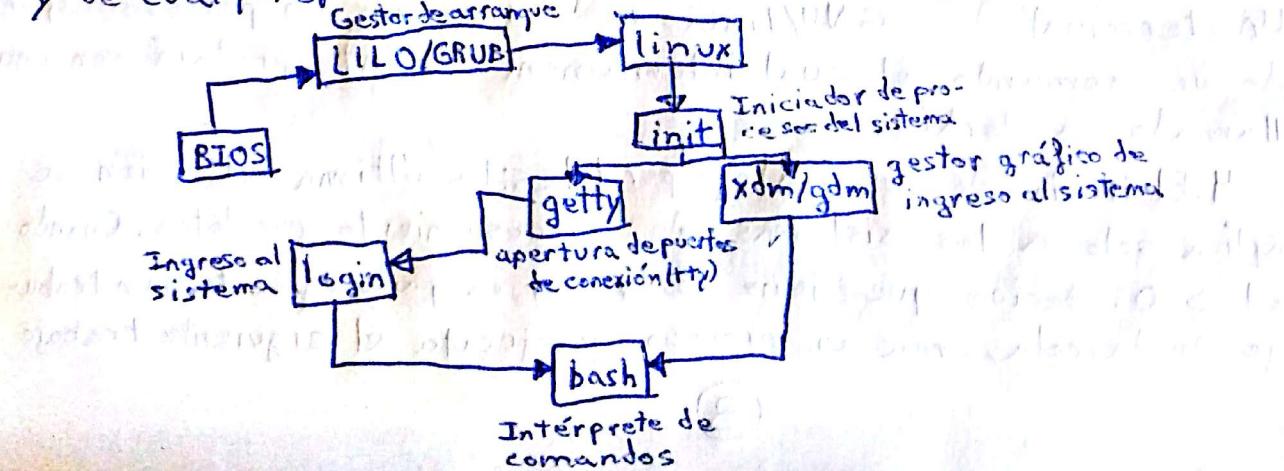
4. Establecer los enlaces apropiados. Por ejemplo si el S.O. mantiene cada cola del planificador como una lista enlazada, éstas deben actualizarse de manera que el nuevo proceso se sitúe en la cola de listos o en la de Listos/Suspendidos.

#### 4. 1. Jerarquía de procesos en GNU/Linux

Existen dos tipos de procesos: los de usuario y los demonios.

Una vez que el Kernel se inicia, se ejecuta init. Init es el proceso responsable de la inicialización de nuevos procesos a nivel de usuarios; es decir, todos los procesos de usuario excepto el proceso swapper descienden del proceso init.

En el momento en el que init finaliza su ejecución, finaliza el proceso de arranque del sistema, realizando una serie de tareas administrativas. Normalmente init realiza el chequeo de los sistemas de archivos, borra contenido de /tmp e inicia la ejecución de varios servicios. Init también se encarga de adoptar procesos huérfanos. Al cerrar el sistema, es init quien se encarga de matar los procesos restantes, desmontar los sistemas de archivos y de cualquier otra cosa que haya sido configurado para hacer.



Init se ejecuta como demonio y por lo general tiene PID

## 5. Terminación de procesos.

Tarde o temprano el nuevo proceso terminará debido a una de las siguientes condiciones:

1. Salida normal (voluntaria). La mayoría de los procesos terminan debido a que han concluido su trabajo. Cuando un compilador compila un programa, añade implícitamente una llamada al sistema para indicar al S.O. su terminación normal. Esta llamada es `exit()` en POSIX.

2. Salida por error (voluntaria). El proceso encuentra un error y termina.

3. Error fatal (involuntaria). Un error fatal producido por el proceso, a menudo debido a un error en el programa. En algunos sistemas, un proceso puede indicar al S.O. que desea manejar ciertos errores por sí mismo. Esto forma parte del tratamiento y captura de excepciones de los programas. Java tiene un tratamiento de excepciones muy potente.

4. Eliminado por otro proceso (involuntaria). Otro proceso ejecuta una llamada al sistema que indica que lo elimine. `Kill()` en POSIX.

## 6. Un modelo de procesos de dos estados.

El primer paso en el diseño de un S.O. para el control de procesos es describir el comportamiento que se desea que tengan los procesos. Se puede construir el modelo más simple posible observando que un proceso está siendo ejecutado por el procesador o no. Cuando el S.O. crea un proceso, se crea el BCP para el nuevo proceso en el sistema en modo No Ejecutando. De cuando en cuando, el proceso actualmente en ejecución se interrumpirá, y el planificador seleccionará otro proceso a ejecutar y selo hará saber al dispatcher. El proceso saliente pasará del Estado Ejecutando a No ejecutando y pasará a Ejecutando un nuevo proceso.

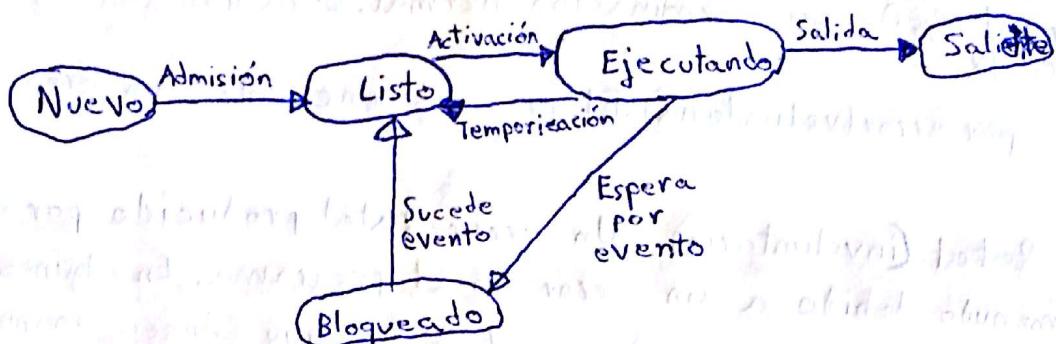
Los procesos que no están ejecutando deben estar en una especie de cola. Existe una sola cola cuyas entradas son punteros al BCP de un proceso en particular.

Un proceso que se interrumpe se transfiere a la cola de procesos en espera. Si el proceso ha finalizado o ha sido abortado, se descarta y sale del sistema.

## 7. Un modelo de procesos de 5 estados.

Utilizando una única cola, el activador no puede seleccionar únicamente los procesos que lleven más tiempo en la cola. Debería recorrer todas las lista buscando los procesos que no estén bloqueados y que lleven en la cola más tiempo.

Una forma más natural para manejar esta situación es dividir el estado de No Ejecutando en dos estados, Listo y Bloqueado.



- **Ejecutando**: Proceso en ejecución

- **Listo**: Se prepara para ejecutar cuando tenga oportunidad.

- **Bloqueado**: No puede ejecutar hasta que se cumpla un evento determinado o se complete una operación de E/S.

- **Nuevo**: Típicamente se trata de un nuevo proceso que no ha sido cargado en memoria principal, aunque su BCP si ha sido creado.

- **Saliente**: Un proceso que ha sido liberado del grupo de procesos ejecutables por el S.O.

Mientras un proceso está en el estado Nuevo, la información relativa al proceso que se necesite por parte del S.O. se mantiene en tablas de control de memoria principal. El proceso en sí mismo no se encuentra en memoria principal. Y no se ha reservado ningún espacio para los datos asociados al programa.

Un proceso sale del sistema en 2 fases. Primero, terminar esta terminación mueve el proceso al estado Saliente. En este punto, el proceso no es elegible de nuevo para su ejecución.

Un programa de utilidad puede requerir extraer información sobre el histórico de los procesos por temas relativos con el rendimiento o análisis de la utilización. Una vez que estos programas han extraído la información necesaria, el S.O. no necesita mantener ningún dato relativo al proceso y el proceso se borra del sistema.

Las posibles transiciones son las siguientes:

• Null → Nuevo: Se crea un nuevo proceso para ejecutar un programa.

• Nuevo → Listo: El S.O. mueve un proceso del estado Nuevo a Listo cuando éste esté preparado para ejecutar un proceso.

• Listo → Ejecutando: Cuando llega el momento de seleccionar un nuevo proceso para ejecutar, el S.O. selecciona uno de los procesos que se encuentre en estado Listo.

• Ejecutando → Saliente: El proceso actual en ejecución se finaliza por parte del sistema operativo tanto si el proceso indica que ha finalizado su ejecución como si éste se aborta.

• Ejecutando → Listo: La razón más habitual para esta transición es que el proceso en ejecución haya alcanzado el máximo tiempo posible de ejecución de forma ininterrumpida; Es de particular importancia el caso en el que el S.O. asigna diferentes niveles de prioridad a diferentes procesos. Un proceso puede dejar voluntariamente de usar el procesador.

• Ejecutando → Bloqueado: Un proceso se pone en estado Bloqueado si solicita algo por lo que debe esperar. Una solicitud al S.O. se realiza habitualmente por medio de una llamada al sistema. Cuando un proceso se comunica con otros un proceso puede bloquearse mientras está esperando a que otro le proporcione datos o esperando un mensaje de ese otro proceso.

• Bloqueado → Listo: Se mueve a Listo cuando sucede el evento por el cual estaba esperando.

• Listo → Saliente: Un padre puede terminar la ejecución de un hijo en cualquier momento

• Bloqueado → Saliente: Lo dicho antes.

### 7.1. Procesos suspendidos.

La necesidad de intercambio o swapping. Los tres principales estados descritos proporcionan una forma sistemática de modelizar el comportamiento de los procesos y diseñar la implementación del S.O.

Existe una buena justificación para añadir otros estados al modelo. Para ver este beneficio de nuevos estados, vamos a suponer un sistema que no utiliza memoria virtual.

Es verdad que la memoria almacena múltiples procesos y el procesador puede asignarse a otro proceso si el que lo usa se queda bloqueado. La diferencia de velocidad entre el procesador y la E/S es tal que sería habitual que todos los procesos en memoria se encontraran a esperas

de dichas operaciones. Incluso en un sistema multiprogramado, el ~~procesador~~ ~~tiempo~~ puede estar ocioso la mayor parte del tiempo.

La memoria principal puede expandirse para acomodar más procesos. Hay 2 fallos en esta solución. Primero, existe un coste asociado a la MP. Segundo, el apetito de los programas a nivel de memoria ha crecido tan rápido como ha bajado el coste de las memorias. Con las memorias actuales se ha conseguido ejecutar procesos de gran tamaño, no más procesos.

Otra solución es el swapping que implica mover parte o todo el proceso de MP al disco. Existe el riesgo de que el problema empeore, pero debido a que la E/S en disco es más rápida que la E/S sobre otros sistemas, el swapping habitualmente mejora el rendimiento.

Con el uso del swapping debemos añadir el estado Suspendido a nuestro modelo de comportamiento de procesos.

Cuando el S.O. ha realizado la operación de swap tiene 2 opciones para seleccionar un proceso y llevarlo a MP; Puede admitir un nuevo proceso que se haya creado o puede traer un proceso que se encuentra suspendido.

Todos los procesos que fueron suspendidos se encontraban en estado Bloqueado antes de su suspensión. Se debe reconocer que todos los suspendidos estaban bloqueados esperando un evento en particular. Cuando el evento sucede, el proceso no está bloqueado y está disponible para su ejecución.

Necesitamos replantear este aspecto del diseño. Hay 2 conceptos independientes aquí: si un proceso está esperando un evento y si un proceso está transferido de memoria a disco. Para estas combinaciones de 2x2 necesitamos 4 estados: Listo, Bloqueado, Listo/Suspendido y Bloqueado/Suspendido.

Las nuevas transiciones más importantes son las siguientes:

- Bloqueado → Bloqueado/Suspendido: Si no hay procesos listos, al menos uno de los que están bloqueados se transfiere al disco para hacer espacio para otro que no esté bloqueado. Esta transición puede realizarse incluso si hay procesos listos, si el S.O. determina que los listos requieren más MP.

- Bloqueado/Suspendido → Listo/Suspendido. Cuando sucede el elemento por el que estaba esperando.

- Listo/Suspendido → Listo. Cuando no hay más listos en MP, el S.O. necesitará traer uno para continuar la ejecución.

• Listo → Listo/Suspendido. El S.O. preferirá suspender procesos bloqueados que un proceso Listo, porque uno bloqueado ocupa memoria y no se puede ejecutar. Sin embargo, puede ser necesario suspender un proceso Listo, si, con ello, se consigue liberar un bloque suficientemente grande de memoria. El S.O. puede decidir suspender un proceso listo de baja prioridad antes que un Bloqueado de alta prioridad.

Otras transiciones interesantes:

• Nuevo → Listo/Suspendido y Nuevo a Listo: Cuando se crea un proceso

• Nuevo → Listo/Suspendido o en Listo: Nuevo se puede poner en la cola de Listo/Suspendido o en Listo. Nuevo se puede poner en la cola de Listo/Suspendido o en Listo. El S.O. puede crear un BCP y reservar el espacio de direcciones ~~del~~ del proceso. La filosofía de creación de procesos diferida, haciéndolo cuanto más tarde, hace posible reducir la sobrecarga del S.O. y realiza las tareas de creación de procesos cuando el sistema está lleno de procesos bloqueados.

• Bloqueado/Suspendido → Bloqueado: Un proceso termina, liberando alguna memoria principal. Hay un proceso en la cola de Bloqueados/Suspendidos con mayor prioridad que todos los procesos en la cola de Listos/Suspendidos y el S.O. tiene motivos para creer que el elemento, que lo bloquea va a ocurrir en breve. Sería razonable traer el proceso bloqueado a memoria por delante de los Listos.

• Ejecutando → Listo/Suspendido: Un proceso en ejecución se mueve a la cola de listos, cuando su tiempo de uso del procesador finaliza. Si el S.O. expulsa el proceso en ejecución, porque un proceso con mayor prioridad en la cola de Bloqueado/Suspendido acaba de desbloquearse, el S.O. puede mover el proceso en ejecución a la cola de Listos/Suspendidos y liberar parte de la MP.

• De cualquier estado → Saliente: Un proceso termina cuando está ejecutando porque ha completado su ejecución o debido a una condición de fallo. En algunos S.O. un proceso puede terminarse por el proceso que lo creó o cuando el proceso padre a su vez ha terminado.

## 8. Hilos (hebras)

Un hilo es una unidad básica de utilización de la CPU. Si un proceso tiene más de un hilo de control, puede realizar más de una tarea a la vez.

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

- Un estado de ejecución por hilo
- Un contexto o BCP de hilo que se almacena cuando no está en ejecución
- Una pila de ejecución
- Un espacio de almacenamiento para variables locales.
- Un espacio de almacenamiento para memoria y recursos de su proceso, compartido con todos los hilos de su mismo proceso.
- Ahora hay varias pilas separadas para cada hilo que contiene los valores de los registros, la prioridad, y otra información relativa al estado del hilo. Todos los hilos de un proceso comparten el espacio de direcciones y tienen acceso a los mismos datos.



### 8.1. Estados de los hilos

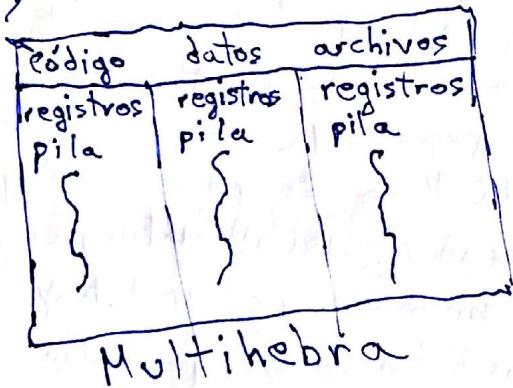
Principales estados de los hilos: Ejecutando, Listo y Bloqueado. Suspender un proceso implica expulsar el espacio de direcciones de un proceso de MP para dejar hueco a otro espacio de direcciones de otro proceso. Ya que todos los hilos de un proceso comparten el mismo espacio de direcciones, todos se suspenden al mismo tiempo.

Suspenso: Hay 4 operaciones básicas relacionadas con los hilos que están asociadas con un cambio de estado del hilo:

• Creación: Un hilo del proceso puede crear otro hilo dentro del mismo proceso proporcionando un puntero a las instrucciones y los argumentos para el nuevo hilo. Al nuevo hilo se le proporciona su propio registro de contexto y espacio de pila y se coloca en la cola de listos.

• Bloqueo: Cuando un hilo necesita esperar un evento se bloquea. El procesador puede pasar a ejecutar otro hilo en estado

• Listo: Una operación realizada por el hilo para indicar que ya no necesita el procesador.



- Desbloqueo: Cuando sucede el evento por el que está esperando, el hilo pasa a la cola de Listos.
- Finalización: Cuando se completa un hilo, se liberan su registro de contexto y pilas.

### 8.2. Motivación

Muchos paquetes de software que se ejecutan en los PC modernos de escritorio son multihilo. Cuatro ejemplos más de uso de hilos para un sistema de multiprocesamiento de un solo usuario y enfocados en conceptos más generales.

• Trabajo en primer plano y en segundo plano: en un programa de hoja de cálculo, un hilo podría mostrar menús y leer la entrada de usuario, mientras otro hilo ejecuta los mandatos de usuario y actualiza la hoja de cálculo.

• Procesamiento asíncrono: Los elementos asíncronos de un programa se pueden implementar como hilos. Se puede crear un hilo cuyo único trabajo sea crear una copia de seguridad periódicamente y que se planifique directamente a través del S.O.

• Velocidad de ejecución: Un proceso multihilo puede computar una serie de datos mientras lee los siguientes de un dispositivo. En un sistema multiprocesador pueden estar ejecutando múltiples hilos de un mismo proceso. Aunque un hilo pueda estar bloqueado por una operación de E/S mientras lee datos, otro hilo puede estar ejecutando.

• Estructura modular de programas: Los programas se pueden diseñar e implementar fácilmente usando hilos.

Por último, ahora muchos Kernel de S.O. son multihilo; hay varias hebras operando en el Kernel y cada una realiza una tarea específica, tal como gestionar dispositivo o tratar interrupciones. GNU/Linux utiliza una hebra del Kernel para gestionar la cantidad de memoria libre en el sistema.

### 8.3. Ventajas.

Las ventajas de la programación multihilo pueden dividirse en 4 categorías principales.

1. Capacidad de respuesta: Permite que un programa continúe ejecutándose incluso aunque parte de él esté bloqueado o realizando una operación muy larga, lo que incrementa la capacidad de respuesta al usuario.

2. Compartición de recursos. Las hebras comparten la memoria y los recursos del proceso al que pertenecen. Permite que una aplicación tenga varias hebras diferentes dentro del mismo espacio de direcciones.

3. Economía y tiempo de ejecución. Se consume mucho más tiempo en crear y gestionar los ~~hilos~~ procesos que las hebras.

- Crear o terminar una hebra es mucho más rápido que un proceso. Cuando termina un proceso se debe eliminar su PCB y todas las estructuras de datos y espacio de memoria de usuario ocupado, mientras que si se termina un hilo, se elimina solo su contexto y pila.

- El cambio de contexto entre hebras es realizado mucho más rápidamente que el cambio de contexto entre procesos, mejorando el rendimiento del sistema. Al cambiar de proceso, el S.O. genera lo que se conoce como overhead, que es tiempo "desperdiciado" por el procesador para realizar un cambio de contexto por medio del dispatcher.

- Si se necesitase comunicación entre hebras también sería mucho más rápido que intercomunicar procesos, ya que los datos están inmediatamente habilitados y disponibles entre hebras. Entre hilos pueden comunicarse entre sí sin la invocación al núcleo.

4. Uso sobre arquitecturas multiprocesador. Las hebras pueden ejecutarse en paralelo en los distintos procesadores. Los mecanismos multihilo en una máquina con varias CPU incrementan el grado de concurrencia.

#### 8.4. Modelos multihilo.

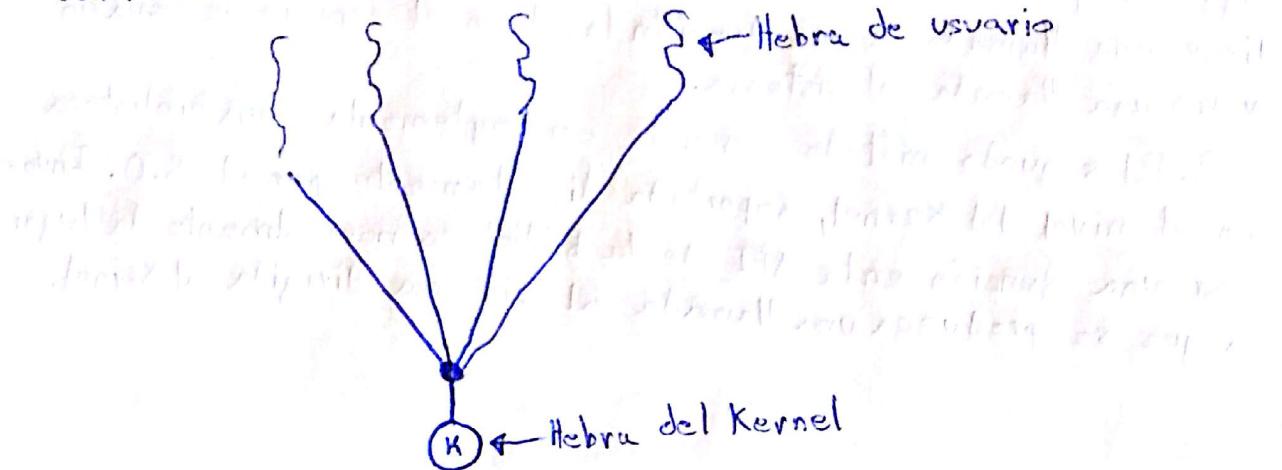
El soporte para hebras puede proporcionarse a nivel de usuario(ULT) o por parte del Kernel(KLT). El S.O. soporta y gestiona directamente las hebras del Kernel. Casi todos los S.O. actuales soportan las hebras del Kernel.

##### 8.4.1. Modelos muchos a uno(ULT)

Este modelo asigna múltiples hebras del nivel de usuario a una hebra del Kernel. El proceso completo se bloquea si una hebra realiza una llamada bloqueante al sistema.

El núcleo no es consciente de esta actividad, los hilos son invisibles para él. Continúa planificando el proceso como una unidad y asigna al proceso un único estado.

Se utilizaban en las aplicaciones que se usaban en S.O. que no podían planificar hilos. El sistema de hebras Green (Bibl. Solaris) y GNU Portable Threads usan este modelo. Son muy portables ya que se pueden ejecutar en cualquier S.O., acepte o no hilos. No se necesita cambios en el nuevo núcleo para darles soporte.



Como desventaja principal, cuando un proceso realiza una operación de E/S bloqueadora, éste pasa a estado bloqueado mientras se espera esa entrada y el S.O. pasa a ejecutar otro proceso. Si esto lo aplicamos a hilos, cuando un hilo a nivel de biblioteca realiza una llamada al sistema bloqueante, no sólo se bloquea ese hilo, sino que se bloquean todos los hilos del proceso. No hay paralelismo real, no se saca ventaja al concepto de multiprogramamiento.

#### 8.4.2. Modelos uno a uno (KLT)

Asigna cada hebra de usuario a una hebra del Kernel. Permite que se ejecute otra hebra mientras una hace una llamada bloqueante al sistema; también permite que se ejecuten múltiples hebras en paralelo sobre varios procesadores.

No hay código de gestión de hilos en la aplicación a nivel de biblioteca solamente una API para acceder a las utilidades de hilos del núcleo. El núcleo mantiene información de contexto del proceso como una entidad y de los hilos individuales del proceso. La planificación realizada por el núcleo se realiza a nivel de hilo.

El único inconveniente de este modelo es que crear una hebra de usuario requiere crear la correspondiente hebra del Kernel, produciéndose un cambio a modo núcleo. La mayoría de implementaciones de este modelo restringen el número de hebras soportadas por el sistema. GNU/Linux y Windows utilizan este modelo.

Existen dos formas principales para gestionar implementar una biblioteca de hebras:

1. El primer método consiste en proporcionar una biblioteca enteramente en el espacio de usuarios sin ningún soporte del Kernel. Esto significa que invocar a una función de la biblioteca es como realizar una llamada a una función local en el espacio de usuario y no una llamada al sistema.

2. El segundo método consiste en implementar una biblioteca en el nivel del Kernel, soportada directamente por el S.O. Invocar una función en la API de la biblioteca normalmente da lugar a que se produzca una llamada al sistema dirigida al Kernel.