Tema-2.pdf

cdl_99

Sistemas Operativos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Tema 2 Libros

1. Diseño e implementación de procesos e hilos

¿Que es un proceso?

Un **proceso** es una instancia de un programa en ejecución. Cada proceso tiene su propio espacio en memoria y un identificador único (PID). Es gestionado por el sistema operativo, que lo coordina para que realice sus tareas sin interferir con otros procesos.

En cualquier instante puntual de tiempo, *mientras el proceso está en ejecución,* este proceso se puede caracterizar por una serie de elementos, incluyendo los siguientes:

- **Identificador:** identificador unico asociado a un proceso para asi poder distinguirlo del resto de procesos.
- **Estado**
- **Prioridad**
- **Contador de programa**
- **Punteros a la memoria**
- **Datos de contexto:** datos que estan presentes en los registros del procesador cuando el proceso está corriendo.
- **Información de estado de E/S**
- **Infromación de auditoría:** puede incluir la cantidad de tiempod e procesador y de tiempo de reloj utilziados, asi como los limites de tiempo, registros contrables, etc.

Toda esta informacion se almacena en el bloque de control de proceso, la cual es creada y gestionada por el sistema operativo.

El punto más significativo en relación al bloque de control de proceso, o BCP, es que contiene suficiente información de forma que es posible interrumpir el proceso cuando está corriendo y posteriormente restaurar su estado de ejecución como si no hubiera habido interrupción alguna. El BCP es la herramienta clave que permite al sistema operativo dar soporte a múltiples procesos y proporcionar multiprogramación.

Multitarea

Dependiendo del número de procesos y de usuarios que puedan ejecutar simultáneamente, un sistema operativo puede ser:

- **Monotarea o monoproceso**: solamente permite que exista un proceso en cada instante. Si se quieren ejecutar varios procesos, o tareas, hay que lanzar

la ejecución de la primera y esperar a que termine antes de poder lanzar la siguiente. El ejemplo típico de sistema operativo monoproceso es el MS-DOS. La ventaja de estos sistemas operativos es que son muy sencillos.

- **Multitarea o multiproceso:** permite que coexistan varios procesos activos a la vez. El sistema operativo se encarga de ir repartiendo el tiempo del procesador entre estos procesos, para que todos ellos vayan avanzando en su ejecución.
- **Monousuario:** está previsto para dar soporte a un solo usuario. Estos sistemas pueden ser monoproceso o multiproceso. En este último caso el usuario puede solicitar varias tareas al mismo tiempo.
- **Multiusuario:** da soporte a varios usuarios que trabajan simultáneamente desde varios terminales. A su vez, cada usuario puede tener activos más de un proceso, por lo que el sistema, obligatoriamente, ha de ser multitarea. Los sistemas multiusuario reciben también el nombre de tiempo compartido, puesto que el sistema operativo ha de repartir el tiempo de la computadora entre los usuarios, para que las tareas de todos ellos avancen de forma razonable.

Base de la multitarea

La multitarea se basa en las tres características siguientes:

- Paralelismo real entre E/S y procesador.
- Alternancia en los procesos de fases de E/S y de procesamiento.
- Memoria principal capaz de almacenar varios procesos.

Planificador y activador

El planificador (*scheduler*) forma parte del núcleo del sistema operativo. Entra en ejecución cada vez que se activa el sistema operativo y su misión es seleccionar el proceso que se ha de ejecutar a continuación.
El activador (*dispatcher*) también forma parte del sistema operativo y su función es poner en ejecución el proceso seleccionado por el planificador.

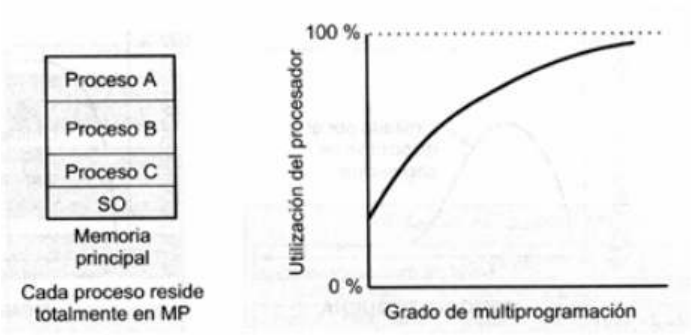
Ventajas de la multitarea

- Facilita la programación. Permite dividir las aplicaciones en varios procesos, lo que beneficia a su modularidad.
- Permite prestar un buen servicio, puesto que se puede atender a varios usuarios dc forma eficiente, interactiva y simultánea.
- Aprovecha los tiempos muertos que los procesos pasan esperando a que se completen sus operaciones de E/S.
- Aumenta el uso de la UCP, al aprovechar los espacios de tiempo que los procesos están bloqueados.

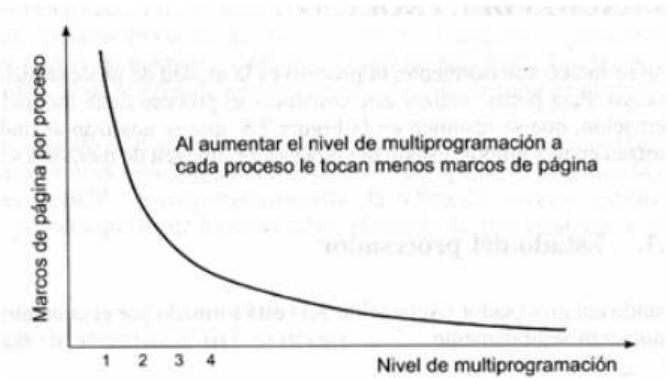
Grado de multiprogramación y necesidades de memoria principal

El **grado de multiprogramación** es el número de procesos activos en un sistema. A mayor número de procesos, más probable es que siempre haya uno listo para ejecutarse, lo que mejora el uso del procesador. Sin embargo, esto aumenta las necesidades de memoria.

En **sistemas sin memoria virtual**, los procesos deben caber completamente en la memoria principal, lo que limita el grado de multiprogramación. Aunque el rendimiento mejora con más procesos activos, está limitado por la cantidad de memoria disponible.



En sistemas **con memoria virtual**, los procesos solo tienen en memoria principal su conjunto residente, lo que permite alojar más procesos. Sin embargo, al aumentar el número de procesos, el conjunto residente de cada uno disminuye, lo que puede causar fallos de página frecuentes. Cada fallo de página consume tiempo de procesador y de FIS, ya que se requiere mover páginas. Esto hace que, a medida que aumentan los fallos, el sistema gaste más tiempo resolviéndolos en lugar de ejecutar procesos. Al principio, el aumento de multiprogramación mejora el rendimiento, pero después de un cierto límite, el sistema pierde eficiencia debido al tiempo dedicado a la paginación.



Se denomina hiperpaginación (*trashing*) a la situación de alta paginación producida cuando los conjuntos residentes de los procesos son demasiado pequeños.

Cuando la memoria principal disponible es pequeña, se llega a la situación de hiperpaginación antes de alcanzar una cota alta de utilización del procesador.

Para aumentar el rendimiento de un sistema que esté en esta situación es necesario añadir más memoria principal. Cuando la memoria es grande se llega a saturar el procesador con menos procesos de los que caben en memoria. En este caso se puede aumentar el
rendimiento del sistema manteniendo la memoria pero aumentando la potencia del procesador o añadiendo otro procesador.

Estado de los procesos

Para que un programa pueda ejecutarse, es necesario que el sistema cree un proceso para él. Desde la perspectiva del procesador, este solo sigue una secuencia de instrucciones que le indica el registro llamado **contador de programa**. Este contador va señalando qué instrucciones ejecutar en cada momento. A medida que el procesador avanza, el contador puede pasar por secciones de código que pertenecen a distintos programas, que en realidad son parte de diferentes procesos.

Para cada programa individual, su ejecución es simplemente una serie de instrucciones propias que el procesador sigue paso a paso.

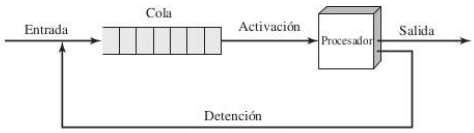
Un modelo de proceso de dos estados

La responsabilidad princiapl del sitema operativo es controlar la ejecucion de los procesos; esto incluye determinar el patron de entrelazado para la ejecución y asignar recursos a los procesos, El primer paso en el diseño de una sistema operativo para el control de procesos es describir el comportmiento que se desea que tengan los procesos.

En el modelo más simple, un proceso puede estar en dos estados: **Ejecutando** o **No Ejecutando**. Cuando el sistema operativo crea un proceso, le asigna un bloque de control de proceso (BCP) y lo coloca en estado No Ejecutando, esperando su turno.

Para gestionar los procesos, el sistema operativo utiliza una **cola** que mantiene el orden de los procesos que esperan su oportunidad de ejecución. Cuando un proceso en ejecución se detiene (o finaliza), se mueve a esta cola, y el **activador** del sistema selecciona otro proceso de la cola para que empiece a ejecutarse. Así, el sistema operativo siempre sabe el estado de cada proceso y su ubicación en memoria.

(a) Diagrama de transiciones de estados



(b) Modelos de colas

Figura 3.5. Modelo de proceso de dos estados.

Creación y terminación de procesos

Cuando se va a añadir un nuevo proceso a aquellos que se están gestionando en un determinado momento, el sistema operativo construye las estructuras de datos que se usan para manejar el proceso y reserva el espacio de direcciones en memoria principal para el proceso. Estas acciones constituyen la creación de un nuevo proceso.

Eventos comunes que llevan a la creación de un proceso:

Nuevo proceso de lotes	El sistema operativo dispone de un flujo de control de lotes de trabajos, habitualmente una cinta un disco. Cuando el sistema operativo está listo para procesar un nuevo trabajo, leerá la siguiente secuencia de mandatos de control de trabajos.
Sesión interactiva	Un usuario desde un terminal entra en el sistema.
Creado por el sistema operativo para proporcionar un servicio	El sistema operativo puede crear un proceso para realizar una función en representación de un programa de usuario, sin que el usuario tenga que esperar (por ejemplo, un proceso para controlar la impresión).
Creado por un proceso existente	Por motivos de modularidad o para explotar el paralelismo, un programa de usuario puede ordenar la creación de un número de procesos.

Cuando un proceso lanza otro, al primero se le denomina proceso padre, y al proceso creado se le denomina proceso hijo. Habitualmente, la relación entre procesos necesita comunicación y cooperación entre ellos

Todo sistema debe proporcionar los mecanismos mediante los cuales un proceso indica su finalización, o que ha completado su tarea. Un trabajo por lotes debe incluir una instrucción `HALT` o una llamada a un servicio de sistema operativo especifica para su terminación. En el caso anterior, la instrucción `HALT` generará una interrupción para indicar al sistema operativo que dicho proceso ha finalizado.

1/6

Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

ING BANK NV se encuentra adherido al Sistema de Garantía de Depósitos Holandés con una garantía de hasta 100.000 euros por depositante. Consulta más información en ing.es

Que te den **10 € para gastar** es una fantasía. ING lo hace realidad.

Abre la **Cuenta NoCuenta** con el código **WUOLAH10**, haz tu primer pago y llévate 10 €.

Quiero el cash

Consulta condiciones **aquí**

do your thing

Finalización normal	El proceso ejecuta una llamada al sistema operativo para indicar que ha completado su ejecución
Límite de tiempo excedido	El proceso ha ejecutado más tiempo del especificado en un límite máximo. Existen varias posibilidades para medir dicho tiempo. Estas incluyen el tiempo total utilizado, el tiempo utilizado únicamente en ejecución, y, en el caso de procesos interactivos, la cantidad de tiempo desde que el usuario realizó la última entrada.
Memoria no disponible	El proceso requiere más memoria de la que el sistema puede proporcionar.
Violaciones de frontera	El proceso trata de acceder a una posición de memoria a la cual no tiene acceso permitido.
Error de protección	El proceso trata de usar un recurso, por ejemplo un fichero, al que no tiene permitido acceder, o trata de utilizarlo de una forma no apropia, por ejemplo, escribiendo en un fichero de sólo lectura.
Error aritmético	El proceso trata de realizar una operación de cálculo no permitida, tal como una división por 0, o trata de almacenar números mayores de los que la representación hardware puede codificar.
Límite de tiempo	El proceso ha esperado más tiempo que el especificado en un valor máximo para que se cumpla un determinado evento.
Fallo de E/S	Se ha producido un error durante una operación de entrada o salida, por ejemplo la imposibilidad de encontrar un fichero, fallo en la lectura o escritura después de un límite máximo de intentos (cuando, por ejemplo, se encuentra un área defectuosa en una cinta), o una operación inválida (la lectura de una impresora en línea).
Instrucción no válida	El proceso intenta ejecutar una instrucción inexistente (habitualmente el resultado de un salto a un área de datos y el intento de ejecutar dichos datos).
Instrucción privilegiada	El proceso intenta utilizar una instrucción reservada al sistema operativo.
Uso inapropiado de datos	Una porción de datos es de tipo erróneo o no se encuentra inicializada.
Intervención del operador por el sistema operativo	Por alguna razón, el operador o el sistema operativo ha finalizado el proceso (por ejemplo, se ha dado una condición de interbloqueo).
Terminación del proceso padre	Cuando un proceso padre termina, el sistema operativo puede automáticamente finalizar todos los procesos hijos descendientes de dicho padre.
Solicitud del proceso padre	Un proceso padre habitualmente tiene autoridad para finalizar sus propios procesos descendientes.

Modelo del proceso de cinco estados

El sistema usa una cola tipo FIFO y aplica una estrategia **round-robin** para asignar tiempo de CPU a cada proceso, permitiendo que todos tengan su turno. Sin embargo, no todos los procesos en espera están listos para ejecutar; algunos están bloqueados esperando que se complete una operación de E/S. Por eso, usar una sola cola resulta ineficiente, ya que el activador tendría que buscar entre procesos bloqueados y no bloqueados para encontrar uno listo.

Para resolverlo, el estado de **No Ejecutando** se divide en dos subestados: **Listo** y **Bloqueado**, facilitando que el sistema distinga los procesos preparados para ejecutar de aquellos en espera de un evento externo. Esto organiza mejor los procesos y permite una gestión más precisa.

Estos cinco estados en el nuevo diagrama son los siguientes:

- **Ejecutado**: el proceso esta actualmente en ejecucion. Para este tema asumimos que el computador tiene un unico precesador, de forma que solo un proceso puede estar en este estado en un isntante determinado.
- **Listo**: un proceso que se prepara para ejecutar cuando tenga oportunidad.

- **Bloqueado**: un proceso que no peude ejecutar hasta que se cumpla un evento determiando o se complete una operacion E/S.
- **Nuevo**: un proceso que se acaba de crear que aún no ha sido admitido en el grupo de procesos ejecutables por el sistema operativo.
- **Saliente**: un proceso que ha sido liberado del grupo de procesos ejecutables por el sistema operativo, debido a que ha sido detenido o que ha sido abortado por alguna razon.

Figura 3.6. Modelo de proceso de cinco estados.

La Figura 3.6 indica que tipos de eventos llevan a cada transición de estado para cada proceso, las posibles transiciones son las siguientes:

- **Null** → **Nuevo**: Se crea un proceso para ejecutar un programa.
- **Nuevo** → **Listo**: cuando está listo para ejecutar, respetando límites de memoria y cantidad de procesos activos.
- **Listo** → **Ejecutando**: El planificador selecciona un proceso en estado Listo para que comience a ejecutarse.
- **Ejecutando** → **Saliente**: El proceso actual finaliza su ejecución y sale del sistema.
- **Ejecutando** → **Listo**: Un proceso en ejecución pasa a Listo cuando alcanza su tiempo máximo de CPU o si otro proceso con mayor prioridad se vuelve listo para ejecutar.
- **Ejecutando** → **Bloqueado**: Un proceso se bloquea si debe esperar algún recurso, servicio o comunicación que no está disponible inmediatamente.
- **Bloqueado** → **Listo**: Un proceso bloqueado pasa a Listo cuando el evento que estaba esperando ocurre.
- **Listo** → **Saliente** y **Bloqueado** → **Saliente**: Un proceso puede terminar si su proceso padre lo finaliza, o si el padre finaliza y todos sus procesos hijos se cierran también.

Procesos suspendidos

El modelo básico de tres estados (Listo, Ejecutando, Bloqueado) es útil, pero puede ser limitado, sobre todo en sistemas sin memoria virtual. En tales

casos, todos los procesos deben estar en memoria principal, lo que puede provocar que, debido a las operaciones de E/S (que son mucho más lentas), el procesador esté inactivo incluso en sistemas multiprogramados.

Una solución es **expandir la memoria** para alojar más procesos, aunque esto puede ser costoso y poco efectivo. La alternativa es el **swapping**, que traslada temporalmente procesos bloqueados al disco, situándolos en un estado de **Suspendido** y liberando memoria principal para otros procesos. Aunque el swapping es una operación de E/S, generalmente mejora el rendimiento porque la E/S en disco suele ser más rápida que en otros dispositivos.

Con el uso de **swapping**, es necesario añadir el estado **Suspendido** al modelo de procesos. Cuando todos los procesos en memoria están bloqueados, el sistema puede mover uno al estado Suspendido, liberando espacio en memoria para otros procesos.

Tras realizar un swap, el sistema operativo puede optar por traer de vuelta un proceso suspendido en lugar de uno nuevo, evitando sobrecargar el sistema. Sin embargo, como los procesos suspendidos estaban previamente bloqueados, el sistema debe asegurarse de que el proceso suspendido esté listo para ejecutarse cuando vuelva a memoria.

De esta forma, necesitamos replantear este aspecto del diseño. Hay dos conceptos independientes aquí: si un proceso está esperando a un evento (Bloqueado o no) y si un proceso está transferido de memoria a disco (suspendido o no). Para describir estas combinaciones de 2 x 2 necesitamos cuatro estados:

- **Listo:** El proceso está en memoria principal disponible para su ejecución.
- **Bloqueado:** El proceso está en memoria principal y esperando un evento.
- **Bloqueado/Suspendido:** El proceso está en almacenamiento secundario y esperando un evento.
- **Listo/Suspendido:** El proceso está en almacenamiento secundario pero está disponible para su ejecución tan pronto como sea cargado en memoria principal.

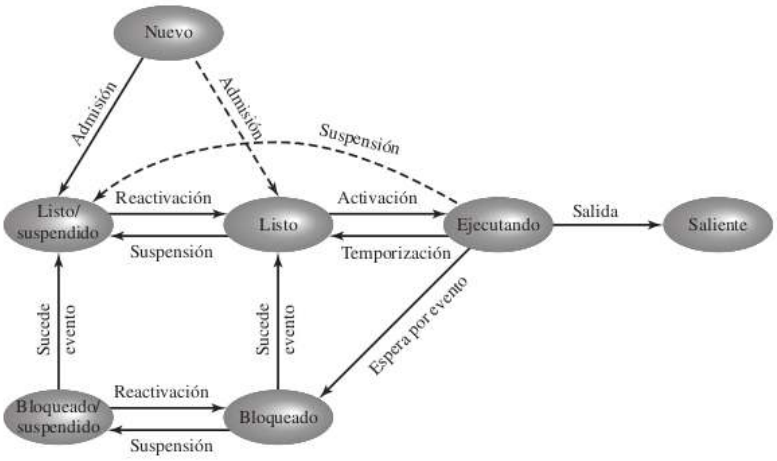
> 📌 Antes de analizar un diagrama de transiciones que incluya los estados de **Suspendido**, es importante aclarar que esta explicación asume que no se utiliza **memoria virtual**. En ese caso, un proceso está completamente en memoria principal o fuera de ella.

Con memoria virtual, sin embargo, un proceso puede ejecutarse aunque solo una parte esté en memoria principal. Cuando un proceso necesita acceder a una parte que no está en memoria principal, el sistema puede traer esa porción desde el disco. Esto podría parecer suficiente para evitar el **swapping** explícito (trasladar un proceso completo entre memoria y disco), ya que el hardware de gestión de memoria permite transferencias automáticas entre memoria y disco.

Sin embargo, si hay demasiados procesos activos parcialmente cargados en memoria, el rendimiento puede caer drásticamente. Por eso, el sistema operativo aún necesita realizar **swapping completo y explícito** de procesos para optimizar el rendimiento, incluso con memoria virtual.

A continuación se muestran las transiciones de estado de este modelo, incluyendo algunas nuevas y opcionales, representadas por líneas punteadas.

(a) Con un único estado Suspendido



(b) Con dos estados Suspendido

Las nuevas transiciones son las siguientes:

- **Bloqueado → Bloqueado/Suspendido**: Se transfiere un proceso bloqueado al disco si se necesita espacio, especialmente si los procesos listos o en ejecución requieren más memoria.
- **Bloqueado/Suspendido → Listo/Suspendido**: Un proceso en espera se mueve a Listo/Suspendido cuando ocurre el evento que esperaba.
- **Listo/Suspendido → Listo**: Si no hay procesos listos en memoria o un proceso suspendido tiene mayor prioridad, se trae a memoria un proceso Listo/Suspendido.
- **Listo → Listo/Suspendido**: El sistema prefiere suspender procesos bloqueados, pero puede suspender un proceso listo para liberar memoria o priorizar procesos de mayor urgencia.

Otras transiciones interesantes a tener en cuenta son las siguientes:

- **Nuevo → Listo/Suspendido y Nuevo → Listo**: Al crear un proceso, el sistema puede añadirlo a la cola de Listos o Listos/Suspendidos, dependiendo del espacio en memoria principal. La creación anticipada de procesos ayuda a mantener un número suficiente de procesos no bloqueados, aunque puede llenar la memoria y hacer necesaria la cola de Listo/Suspendido.
- **Bloqueado/Suspendido → Bloqueado**: Se trae un proceso Bloqueado/Suspendido a memoria si tiene mayor prioridad y es probable que pronto se desbloquee, incluso si hay procesos Listos/Suspendidos.
- **Ejecutando → Listo/Suspendido**: Si un proceso en ejecución es desplazado por otro de mayor prioridad, puede moverse directamente a Listo/Suspendido para liberar memoria.
- **Cualquier estado → Saliente**: Un proceso puede finalizar desde cualquier estado debido a completar su ejecución, fallos o finalización por su proceso padre.

💡 **Otros usos para la suspensión de procesos.**

1. El proceso no está inmediatamente disponible para su ejecución.
2. El proceso puede estar o no a la espera de un evento, si es así, la condicioón de bloqueo es independiente de la condición estar suspendido, y si sucede el evento que lo bloquea, eso no habiita al proceso para su ejecución inmediata.
3. El proceso fue puesto en estado suspendido por un agente: bien el proceso mismo, el proceso padre o el sistema operativo, con el propósito de prevenir su ejecución.
4. El proceso no puede ser recuperado de este estado hasta qu eek agente explícitamente así lo indique.

⚠️ **Razones para la suspensión de un proceso.**

Swapping	El sistema operativo necesita liberar suficiente memoria principal para traer un proceso en estado Listo de ejecución.
Otras razones del sistema operativo	El sistema operativo puede suspender un proceso en segundo plano o de utilidad o un proceso que se sospecha puede causar algún problema.
Solicitud interactiva del usuario	Un usuario puede desear suspender la ejecución de un programa con motivo de su depuración o porque está utilizando un recurso.
Temporización	Un proceso puede ejecutarse periódicamente (por ejemplo, un proceso monitor de estadísticas sobre el sistema) y puede suspenderse mientras espera el siguiente intervalo de ejecución.

Solicitud del proceso padre	Un proceso padre puede querer suspender la ejecución de un descendiente para examinar o modificar dicho proceso suspendido, o para coordinar la actividad de varios procesos descendientes.

Cambio de contexto

Cuando ocurre una interrupción, el sistema realiza un cambio de contexto que implica guardar el estado del procesador en el BCP y ejecutar la rutina de tratamiento correspondiente. Este cambio puede o no modificar el estado de los procesos - por ejemplo, una lectura de disco completada cambiaría el estado del proceso a "listo", mientras que una interrupción de teclado en un proceso incompleto no alteraría su estado.

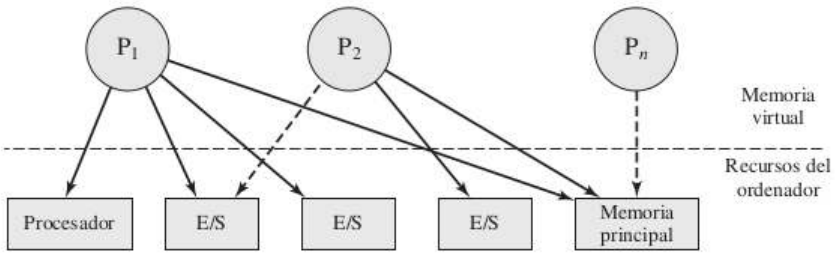
Es fundamental que el sistema operativo guarde correctamente el estado de los registros de la máquina antes de cualquier cambio en la ejecución, ya que estos contienen información vital de los procesos en curso y no deben sobrescribirse durante el tratamiento de la interrupción.

Descripción de procesos

El sistema operativo (SO) administra los recursos de la computadora y coordina los procesos para su ejecución, asignándoles recursos y respondiendo a sus solicitudes básicas.

En un entorno multiprogramado, varios procesos (como P1, P2, Pn) comparten la memoria virtual y requieren recursos como el procesador y dispositivos de entrada/salida. Algunos pueden estar en ejecución, otros bloqueados esperando recursos, y otros suspendidos en disco.

La clave aquí es entender qué información necesita el SO para controlar estos procesos y gestionar sus recursos.



Estructuras de control del sistema operativo

Si el sistema operativo se encarga de la gestión de procesos y recursos, debe disponer de información sobre el estado actual de cada proceso y cada recurso. El mecanismo universal para proporcionar esta información es el siguiente: el sistema operativo construye y mantiene tablas de información sobre cada entidad que gestiona.

A pesar de que los detalles difieren de un sistema operativo a otro, fundamentalmente, todos los sistemas operativos mantienen información de estas

cuatro categorías: memoria, E/S, ficheros y procesos.

Las **tablas de memoria** son estructuras que llevan el control tanto de la memoria principal como de la secundaria. Reservan una parte de la memoria principal para el sistema operativo y el resto lo dejan disponible para los procesos. Estos procesos, a su vez, se mantienen en memoria secundaria usando técnicas de memoria virtual o swapping.

Las tablas de memoria incluyen

1. Reservas de memoria principal de los procesos.
2. Reservas de memoria secundaria de los procesos.
3. Atributos de protección que limitan el acceso a la memoria, permitiendo áreas compartidas cuando sea necesario.
4. Datos necesarios para manejar la memoria virtual.

El sistema operativo utiliza diferentes tipos de tablas para gestionar sus recursos:

1. Las **tablas de E/S** supervisan los dispositivos de entrada/salida, monitoreando si están disponibles o asignados a algún proceso, su estado actual y las direcciones de memoria utilizadas para transferir datos.
2. Las **tablas de ficheros** almacenan información sobre la existencia, ubicación y estado de los archivos. Esta gestión puede ser manejada directamente por el sistema operativo o delegada al sistema de ficheros.
3. Las **tablas de procesos** controlan los procesos en ejecución. Todas estas tablas están interconectadas, ya que los procesos necesitan acceder a los recursos que las otras tablas gestionan.

Para **crear** estas **tablas** inicialmente, el sistema operativo requiere información básica de configuración (como cantidad de memoria física y dispositivos disponibles), que se establece durante el inicio del sistema, ya sea manualmente o mediante autoconfiguración.

Estructuras de control de proceso

Se considerará qué información debe conocer el sistema operativo si quiere manejar y controlar los procesos. Primero, debe conocer dónde están localizados los procesos, y segundo, debe conocer los atributos de los procesos que quiere gestionar.

Localización de los procesos

Esto implica entender su representación física. Un proceso debe incluir al menos un programa o conjunto de programas que se ejecutan, junto con espacio en memoria para datos de variables locales y globales, así como constantes definidas. Esto significa que un proceso necesita suficiente memoria para almacenar tanto el programa como sus datos.

Además, la ejecución de un programa típicamente requiere una pila para registrar las llamadas a procedimientos y los parámetros pasados entre ellos. Cada proceso también tiene atributos que el sistema operativo utiliza para controlarlo, conocidos como el bloque de control del proceso (BCP). En

conjunto, el programa, los datos, la pila y los atributos forman lo que se denomina la imagen del proceso.

> 💡 Imagen del proceso

Datos del usuario	La parte modificable del espacio de usuario. Puede incluir datos de programa, área de pila de usuario, y programas que puedan ser modificados.
Programa de usuario	El programa a ejecutar
Pila de sistema	Cada proceso tiene una o más pilas de sistema (LIFO) asociadas a él. Una pila se utiliza para almacenar los parámetros y las direcciones de retorno de los procedimientos y llamadas a sistema.
Bloque de control de proceso	Datos necesarios para que el sistema operativo pueda controlar los procesos

La **imagen del proceso** puede almacenarse de diferentes formas según el sistema de gestión de memoria. En su versión más básica, se guarda como un **bloque contiguo en memoria secundaria** (disco). Para su gestión, el sistema operativo requiere que al menos una parte de la imagen esté en **memoria principal**, y para su ejecución, necesita la **imagen completa** ya sea en memoria principal o virtual.

El sistema operativo debe mantener un registro de la **ubicación exacta de cada proceso**, tanto en disco como en memoria principal. En sistemas modernos que utilizan **hardware de paginación**, es posible tener la imagen del proceso distribuida en **memoria no contigua** y permitir que los procesos residan **parcialmente en memoria principal**.

Para gestionar esta información, el sistema operativo utiliza **tablas** que incluyen:

- Una **tabla primaria de procesos** con entradas para cada proceso
- **Referencias cruzadas** entre tablas de memoria
- Información sobre la **ubicación de cada página** de la imagen del proceso

Aunque esta es una estructura general, cada sistema operativo puede implementar su propia organización de esta información.

Atributos de proceso.

Todo sistema operativo multiprogramado tiene que gestionar una cantidad significativa de información para cada proceso, y esta se almacena en el BCP (Bloque de Control del Proceso). Aunque cada sistema organiza esta información de manera distinta, todos requieren ciertos datos clave para gestionar los procesos:

1. **Identificación del proceso**

   Identificadores.

Identificadores numericos que se pueden guardar dentro del bloque de control del proceso:

- Identificadores del proceso
- Identificador del proceso que creó a este proceso (proceso padre)
- Identificador del usuario

2. **Información de estado del proceso**

Registros visibles por el usuario.

Un registro visible por el usuario es aquel al que se puede hacer referencia por medio del lenguaje máquina que ejecuta el procesador cuando está en modo usuario. Normalmente, existen de 8 a 32 de estos registros, aunque determinadas implementaciones RISC tienen más de 100.

Registros de estado y control.

- Contador de programa: contiene la dirección de la siguiente instrucción a ejecutar.
- Códigos de condición: resultan de la operación lógica o aritmética más reciente (por ejemplo, signo, cero, acarreo, igual, desbordamiento).
- Información de estado: incluyen los flags de interrupciones habilitadas/deshabilitadas, modo ejecución.

Puntero de pila.

Cada proceso tiene una o más pilas de sistema (LIFO) asociadas a él. Una pila se utiliza para almacenar los parámetros y las direcciones de retorno de los procedimientos y llamadas a sistema.

Un puntero de pila apunta a la parte más alta de la pila.

3. **Información de control de proceso**

Información de estado y de planificación.

- Estado del proceso: Define si está listo para ejecutarse (Ejecutando, Listo, Esperando, Detenido)
- Prioridad: Campos que definen la prioridad de planificación (valores por defecto, actual y mayor disponible)
- Información de planificación: Datos según el algoritmo usado (tiempos de espera y de ejecución)
- Evento: Identifica qué evento espera el proceso para continuar

Estructuración de datos.

- Procesos pueden estar enlazados entre sí en colas, anillos u otras estructuras
- Pueden mostrar relaciones padre-hijo
- El BCP puede contener punteros a otros procesos

Comunicación entre procesos.

Se pueden asociar diferentes *flags, señales, y mensajes relativos* a la comunicación entre dos procesos independientes. Alguna o toda esta información se puede mantener en el bloque de control de proceso (BCP).

Privilegios de proceso.

Los procesos adquieren privilegios de acuerdo con la memoria a la que van a acceder y los tipos de instrucciones que se pueden ejecutar. Adicionalmente, los privilegios se pueden utilizar para usar utilidades de sistema o servicios.

Gestión de memoria.

Esta sección puede incluir punteros a tablas de segmentos y/o páginas que describen la memoria virtual asignada a este proceso.

Propia de recursos y utilización.

Se deben indicar los recursos controlados por el proceso, por ejemplo, ficheros abiertos. También se puede incluir un histórico de utilización del procesador o de otros recursos; esta información puede ser necesaria para el planificador.

Los sistemas operativos asignan a cada proceso un **identificador de proceso** único y numérico, que puede funcionar como índice en la tabla de procesos o como número de referencia para localizar tablas específicas. Este identificador es fundamental para varias funciones: permite referenciar procesos en diferentes tablas (memoria, E/S, archivos), facilita la comunicación entre procesos y establece jerarquías entre procesos padres e hijos. También se puede asignar un identificador de usuario para determinar quién es responsable del proceso.

La **información de estado del proceso** es crucial y comprende el contenido de los registros del procesador. Durante la ejecución, esta información reside en los registros, pero debe guardarse cuando el proceso se interrumpe para poder recuperarla posteriormente. Los tipos y cantidad de registros varían según el procesador, incluyendo registros visibles para usuarios, de control y estado, y punteros de pila. Es especialmente importante la palabra de estado de programa (PSW), presente en todos los procesadores, que contiene información de estado como códigos de condición.

El sistema operativo requiere también **información de control de proceso** para gestionar y coordinar los procesos activos. Las imágenes de los procesos en memoria virtual incluyen varios componentes: el bloque de control del proceso, una pila de usuario, un espacio de direcciones privadas y espacios compartidos con otros procesos. Aunque tradicionalmente se representan como rangos de direcciones contiguos, su organización real puede variar según el sistema de gestión de memoria y las estructuras de control del sistema operativo.

Bits de control	
AC (*Alignment check*)	Fija si una palabra (o una doble-palabra) se encuentra direccionada en la frontera entre palabras (o dobles-palabras).
ID (*Identification flag*)	Si este bit puede ser puesto a 1 y a 0, este procesador soporta la instrucción CPUID. Esta instrucción proporciona información sobre el vendedor, familia, y modelo.
RF (*Resume flag*)	Permite al programador desactivar las extensiones de depuración de forma que la instrucción pueda volverse a ejecutar después de una excepción de depuración sin causar inmediatamente después otra excepción de depuración.
IOPL (*I/O privilege level*)	Cuando está activado, hace que el procesador genere una excepción para todos los accesos a dispositivo de E/S durante una operación en modo protegido.
DF (*Direction flag*)	Determina cuando una instrucción de procesamiento de cadenas de caracteres incrementa o decrementa los semi-registros de 16 bits SE y DI (para operaciones de 16 bits) o los registros de 32 bits ESE y EDI (para operaciones de 32 bits).
IF (*Interruption enable flag*)	Cuando está puesto a 1, el procesador reconocerá interrupciones externas.
TF (*Trap flag*)	Cuando está puesto a 1, causa una interrupción después de la ejecución de cada instrucción. Su uso está dirigido a la depuración de código.
Bits de modos de operación	
NT (*Nested task flag*)	Indica que la tarea actual está anidada dentro de otra tarea en modo de operación protegido.
VM (*Virtual 8086 mode*)	Permite al programador activar o desactivar el modo virtual 8086, que determina si el procesador funciona como si fuese una máquina 8086.
VIP (*Virtual interrupt pending*)	Usado en el modo virtual 8086 para indicar que una o más interrupciones están a la espera de servicio.
VIF (*Virtual interruption flag*)	Usado en modo virtual 8086 en lugar de IF.
Códigos de condición	
AF (*Auxiliar carry flag*)	Representa el acarreo o resto entre medios bytes de una operación aritmética o lógica de 8 bits usando el registro AL.
CF (*Carry flag*)	Indica el acarreo o el resto por medio del bit situado más a la izquierda después de una operación aritmética. También se modificaron por algunas operaciones de desplazamiento o rotación.
OF (*Overflow flag*)	Indica un desbordamiento (*overflow*) aritmético después de una suma o resta.
PF (*Parity flag*)	Paridad del resultado de una operación aritmética o lógica. 1 indica paridad par; 0 indica paridad impar.
SF (*Sign flag*)	Indica el signo del resultado de una operación aritmética o lógica.
ZF (*Zero flag*)	Indica que el resultado de una operación aritmética o lógica es 0.

El **bloque de control de proceso (BCP)** representa la estructura de datos más crucial en un sistema operativo, pues almacena toda la información esencial sobre los procesos. Su estructura puede incluir punteros que permiten conectar diferentes BCPs entre sí, facilitando la implementación de colas mediante listas enlazadas. Prácticamente todos los módulos del sistema operativo, desde la planificación hasta el análisis de rendimiento, interactúan con los BCPs, haciendo que estos definan colectivamente el estado del sistema.

El sistema está diseñado para permitir un acceso eficiente a los BCPs mediante identificadores únicos que funcionan como índices en una tabla de punteros. Sin embargo, esta accesibilidad presenta **desafíos de seguridad** importantes:

- Un error en una rutina simple podría comprometer la gestión de los procesos afectados
- Las modificaciones en la estructura de los BCPs pueden impactar múltiples módulos del sistema operativo

Para proteger la integridad de los BCPs, se puede implementar un **manejador dedicado** que actúe como único punto de acceso autorizado para operaciones de lectura y escritura. Esta solución requiere un balance cuidadoso entre el rendimiento del sistema y la verificabilidad del software para garantizar su correcto funcionamiento.

Control de procesos

Modos de ejecución

El sistema operativo maneja los procesos en dos modos principales: **modo usuario** y **modo sistema** (o **modo núcleo**).

- **Modo usuario:** Aquí se ejecutan tus programas, con **privilegios limitados**. No pueden acceder a ciertas instrucciones o áreas de memoria para evitar errores o que un programa malicioso dañe el sistema.
- **Modo sistema:** Este es el modo donde el sistema operativo tiene **privilegios completos**. Aquí se ejecutan las instrucciones críticas como controlar los registros, gestionar la memoria y las operaciones de entrada/salida (E/S). Es fundamental para la seguridad y estabilidad del sistema.

Funciones tipicas de un nucleo del sistema operativo.

Gestión de procesos	Creación y terminación de procesos Planificación y activación de procesos Intercambio de procesos Sincronización de procesos y soporte para comunicación entre procesos Gestión de los bloques de control de proceso
Gestión de memoria	Reserva de espacio direcciones para los procesos Swapping Gestión de páginas y segmentos
Gestión E/S	Gestión de buffers Reserva de canales de E/S y de dispositivos para los procesos
Funciones de soporte	Gestión de interrupciones Auditoría Monitorización

El modo núcleo en los procesadores permite al sistema operativo tener control total del hardware, protegiendo áreas clave como los bloques de control de procesos de interferencias de los programas de usuario. El procesador utiliza un bit en su palabra de estado (PSW) para distinguir entre modo usuario y modo núcleo. Este bit cambia cuando se invoca una llamada al sistema o llega una interrupción, pasando al modo núcleo para permitir el control seguro. Al finalizar el servicio, el procesador vuelve al modo usuario. En procesadores como el Intel Itanium, el nivel de privilegio se controla mediante el registro de estado `psr`, donde el campo `cpl` (nivel de privilegio actual) define niveles del 0 al 3. Linux y otros sistemas operativos suelen usar el nivel 0 para el núcleo y un nivel inferior para el usuario.

Creación de procesos

1. **Asignación de identificador**: Se le asigna un ID único al proceso y se crea una entrada en la tabla principal de procesos.
2. **Reserva de memoria**: Se aparta el espacio necesario para el proceso, que incluye el área de direcciones (programa, datos y pila) y el bloque de control de proceso (BCP). Si el proceso es creado por otro, puede recibir parámetros del proceso padre, y se configuran los enlaces para espacios compartidos.
3. **Inicialización del BCP**: El BCP se inicializa con el ID del proceso y otros datos. El estado se fija en "Listo" o "Listo/Suspendido", y la prioridad se ajusta según las solicitudes o valores por defecto.
4. **Enlaces en el planificador**: Se colocan los enlaces para que el proceso entre en la cola de Listos o Listos/Suspendidos del sistema operativo.
5. **Creación de registros adicionales**: Se generan registros adicionales como auditorías de rendimiento o facturación, según lo necesite el sistema operativo.

Cambio de proceso

El cambio de proceso ocurre cuando el sistema operativo toma control del proceso en ejecución, lo cual sucede en momentos específicos, como interrupciones de sistema y traps:

1. **Interrupciones ordinarias**: Son eventos externos que requieren atención del sistema:
   - **Interrupción de reloj**: Si un proceso excede su rodaja de tiempo (el tiempo máximo de ejecución continua), el SO lo pasa a Listo y activa otro proceso.
   - **Interrupción de E/S**: Cuando finaliza una operación de E/S esperada, los procesos en espera pasan a Listo (o a Listo/Suspendido si estaban en Bloqueado/Suspendido). El SO puede reemplazar el proceso en ejecución si hay uno de mayor prioridad.
   - **Fallo de memoria**: Ocurre si el procesador intenta acceder a una dirección virtual cuya página no está en memoria principal. El SO debe cargar el bloque desde la memoria secundaria. En este caso, el proceso se pone en estado de Bloqueado, y se realiza un cambio de proceso. Una vez que el bloque esté disponible en memoria, el proceso vuelve a Listo.
2. **Traps**: Son interrupciones causadas por errores dentro del proceso. El SO evalúa si el error es crítico. Si es irreversible, el proceso pasa a Saliente y el SO realiza un cambio de proceso. En caso contrario, el SO puede intentar una recuperación o notificar al usuario, lo que podría implicar un cambio o la continuación del proceso actual.
3. **Llamada al sistema**: Un proceso puede solicitar una operación del SO, como una operación de E/S. Esto implica un salto a una rutina del SO y, en

algunos casos, puede hacer que el proceso pase al estado de Bloqueado mientras espera el resultado de la operación.

Cambio de modo

Cuando llega una **interrupción**, el procesador primero verifica si hay una señal. Si no hay ninguna, continúa con el proceso actual. Si hay una interrupción pendiente, el procesador:

1. **Cambia el contador de programa** para ejecutar la **rutina del manejador de la interrupción**.
2. **Cambia al modo núcleo** para que la rutina tenga acceso a instrucciones privilegiadas.

Luego, se guarda el **contexto** del proceso interrumpido (como el contador de programa, registros y datos de la pila) en su **bloque de control (BCP)**. Esto es como tomar una foto del estado del proceso para poder volver a él más tarde.

El **manejador de interrupción** realiza tareas como:

- **Limpiar el indicador de interrupción**.
- **Confirmar la interrupción** a la entidad que la causó (por ejemplo, un módulo de E/S).
- **Verificar si hubo errores**.
- **Notificar al proceso** si hubo un fallo en una operación de E/S.
- **Transferir el control al planificador** si es una interrupción de reloj y la rodaja de tiempo ha expirado.

No siempre se cambia de proceso tras una interrupción. A veces, el mismo proceso retoma la ejecución después de la rutina de interrupción. En estos casos, solo se necesita guardar y restaurar el estado del procesador, lo que suele hacerse automáticamente por el hardware.

Cambio del estado del proceso

El cambio de modo y el cambio de proceso son distintos: un cambio de modo no afecta el estado del proceso en ejecución y solo requiere una ligera sobrecarga para salvaguardar y restaurar el estado. Sin embargo, si el proceso debe cambiar de estado (por ejemplo, de Ejecutando a Listo o Bloqueado), el sistema operativo realiza un cambio de proceso completo. Este proceso incluye:

1. Guardar el estado del procesador (contador de programa, registros).
2. Actualizar el bloque de control del proceso saliente (nuevo estado, motivo de salida, auditoría).
3. Mover el bloque a la cola correspondiente (Listo, Bloqueado, etc.).
4. Elegir el nuevo proceso a ejecutar.
5. Actualizar el bloque de control del proceso entrante al estado Ejecutando.
6. Modificar las estructuras de datos de memoria si es necesario.
7. Restaurar el estado del procesador al del proceso entrante.

1/6

Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

ING BANK NV se encuentra adherido al Sistema de Garantía de Depósitos Holandés con una garantía de hasta 100.000 euros por depositante. Consulta más información en ing.es

Que te den **10 € para gastar** es una fantasía. ING lo hace realidad.

Abre la **Cuenta NoCuenta** con el código **WUOLAH10**, haz tu primer pago y llévate 10 €.

Quiero el cash

Consulta condiciones **aquí**

do your thing

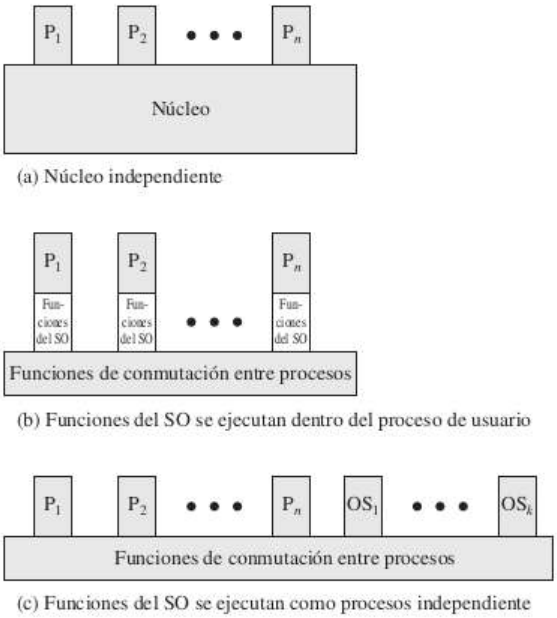
Este proceso de cambio de estado es más complejo y demanda más esfuerzo que un simple cambio de modo.

Ejecución del sistema operativo

Anteriormente se mencionan dos aspectos fundamentales del sistema operativo:

1. Actúa como cualquier software, ya que es un conjunto de programas que el procesador ejecuta.
2. El sistema operativo a menudo cede el control al procesador y depende de él para recuperar dicho control.

Esto plantea la pregunta de si el sistema operativo es un proceso del sistema y cómo se controla. Hay varias perspectivas al respecto:



Núcleo sin procesos (a)

Esta es una visión tradicional en muchos sistemas operativos antiguos, donde el sistema operativo se ejecuta fuera de todo proceso. En esta perspectiva, cuando se interrumpe un proceso o se hace una llamada al sistema, se guarda el contexto y el control pasa al núcleo del sistema operativo. Este tiene su propia memoria y pila para gestionar las llamadas y sus retornos. Puede realizar las funciones necesarias y restaurar el contexto del proceso interrumpido, permitiendo así continuar la ejecución del programa de usuario afectado. También puede activar otro proceso, dependiendo de la causa de la interrupción y las circunstancias en ese momento.

En resumen, en esta visión, el concepto de proceso se aplica únicamente a los programas de usuario, mientras que el código del sistema operativo se maneja de manera separada.

Ejecución dentro de los procesos de usuario (b)

En sistemas operativos para PCs, el sistema operativo se ejecuta como parte de los procesos de usuario. Esto significa que el sistema operativo es un conjunto de herramientas que los usuarios pueden usar. Cuando ocurre una interrupción o una llamada al sistema, el control pasa al sistema operativo, pero la ejecución sigue dentro del mismo proceso de usuario. Si se necesita cambiar de proceso, el control pasa a una rutina especial. Este enfoque garantiza que el sistema operativo tenga control en momentos críticos y que los usuarios no puedan interferir. Además, permite que un proceso ejecute tanto programas de usuario como del sistema operativo.

Sistemas operativos basados en procesos (c)

En este modelo, el sistema operativo se organiza como un conjunto de procesos independientes que se ejecutan en modo núcleo. Aunque el núcleo incluye una pequeña porción de código para el intercambio de procesos que opera fuera de estos procesos, la mayor parte de las funciones del sistema operativo se gestionan de forma modular.

Ventajas de este enfoque:

1. **Diseño modular:** Promueve un diseño disciplinado con interfaces mínimas y claras entre módulos, facilitando la gestión y mantenimiento del sistema.
2. **Separación de funciones no críticas:** Permite que funciones como la monitorización del uso de recursos (procesador, memoria, etc.) se ejecuten como procesos separados, lo que mejora la organización y la eficiencia del sistema.
3. **Optimización en entornos multiprocesadores:** En sistemas de multiprocesadores y multicomputadores, ciertos servicios del sistema operativo pueden asignarse a procesadores dedicados, lo que potencia el rendimiento del sistema.

Este enfoque busca mejorar la eficiencia y la flexibilidad del sistema operativo al implementar una estructura modular que puede adaptarse mejor a las necesidades de los entornos actuales.

Unix SVR4 process management

UNIX System V utiliza una gestión de procesos sencilla pero efectiva, basada en un modelo donde la mayoría del sistema operativo opera dentro del entorno del proceso. Esto implica la existencia de dos modos: usuario y núcleo. Hay dos categorías de procesos:

1. **Procesos del sistema:** Funcionan en modo núcleo y manejan tareas administrativas y funciones internas, como la reserva de memoria o el intercambio de procesos (swapping).

2. **Procesos de usuario:** Operan en modo usuario para ejecutar programas y utilidades, pero también pueden entrar en modo núcleo para ejecutar instrucciones del sistema operativo, lo cual sucede a través de llamadas al sistema, excepciones o interrupciones.

Esta estructura permite una interacción fluida y eficaz entre los procesos del usuario y del sistema.

Estados de los procesos

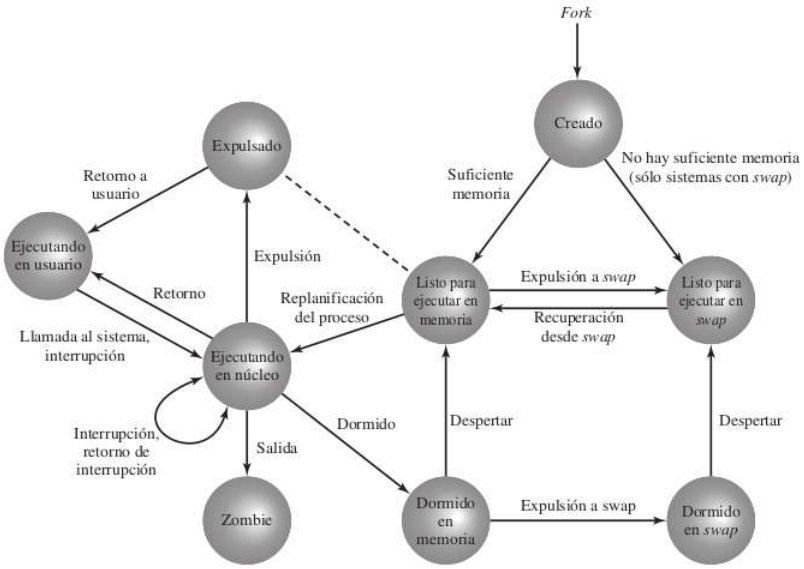


Figura 3.17. Diagrama de transiciones entre estados de procesos UNIX.

- **Ejecutando Usuario:** ejecutando en modo usuario.
- **Ejecutando Núcleo:** ejecutando en modo núcleo.
- **Listo para Ejecutar, en memoria:** listo para ejecutar tan pronto como el núcleo lo planifique.
- **Listo para Ejecutar, en Swap:** El proceso está listo para preguntar, pero el swapper debe cargar el proceso en memoria principal antes de que el núcleo pueda planificarlo para su ejecución.
- **Durmiendo, en Swap:** el proceso está esperando un evento y ha sido expulsado al almacenamiento secundario (estado de bloqueo).
- **Expulsado**: el proceso ha regresado de modo núcleo a modo usuario, pero el núcleo lo ha expulsado y ha realizado la activación de otro proceso.
- **Creado**: el proceso ha sido creado recientemente y aún no está listo para ejecutar.
- **Zombie**: El proceso ya no existe, pero deja un registro para que lo recoja su proceso padre.

Descripción de procesos

Un proceso en UNIX se compone de estructuras de datos complejas que permiten al sistema operativo gestionar y activar los procesos de manera efectiva. Estas estructuras se organizan en tres partes:

Contexto a Nivel de Usuario

- **Texto:** Instrucciones máquina ejecutables del programa.
- **Datos:** Datos accesibles por el programa asociado a dicho proceso.
- **Pila de Usuario:** Contiene los argumentos, las variables locales y los punteros a funciones que se ejecutan en modo usuario.
- **Memoria Compartida:** Espacio de memoria compartido con otros procesos, utilizado para la comunicación entre procesos.

Contexto de Registros

- **Contador de Programa:** Dirección de la siguiente instrucción a ejecutar, que puede corresponder al espacio de memoria del núcleo o al de usuario del proceso.
- **Registro de Estado del Procesador:** Contiene el estado del hardware del procesador en el momento de la expulsión; su contenido y formato dependen del hardware específico.
- **Puntero de Pila:** Apunta a la cima de la pila, ya sea del núcleo o del usuario, dependiendo del modo de operación al momento de la expulsión.
- **Registros de Propósito General:** Su contenido depende del hardware.

Contexto a Nivel de Sistema

- **Entrada en la Tabla de Procesos:** Define el estado del proceso; esta información siempre es accesible para el sistema operativo.
- **Área U (de Usuario):** Contiene información de control del proceso que solo es necesaria en el contexto del propio proceso.
- **Tabla de Regiones por Proceso:** Define la traducción entre direcciones virtuales y físicas, e incluye información sobre los permisos de acceso permitidos (solo-lectura, lectura-escritura o lectura-ejecución).
- **Pila del Núcleo:** Contiene el marco de pila de los procedimientos del núcleo cuando el proceso se ejecuta en modo núcleo.
- **Estado del Proceso:** Indica el estado actual del proceso.
- **Punteros:** Referencias al Área U y a la memoria del proceso (texto, datos, pila).
- **Tamaño del Proceso:** Permite al sistema operativo conocer cuánto espacio está reservado para este proceso.
- **Identificadores de Usuario:**
  - **ID de Usuario Real:** Identifica al usuario responsable de la ejecución del proceso.

    - **ID de Usuario Efectivo:** Permite que el proceso obtenga temporalmente los privilegios de un programa específico durante su ejecución.
- **Identificadores de Proceso:** Incluyen el identificador del proceso actual y el identificador del proceso padre, fijados al entrar en el estado de Creado tras la llamada al sistema `fork`.
- **Descriptor de Evento:** Válido cuando el proceso está en un estado dormido; cuando ocurre un evento, el proceso cambia al estado Listo para Ejecutar.
- **Prioridad:** Utilizada en la planificación del proceso.
- **Señal:** Enumera las señales enviadas a este proceso que aún no han sido manejadas.
- **Temporizadores:** Incluye el tiempo de ejecución del proceso, el uso de recursos del núcleo y el temporizador fijado por el usuario para enviar una señal de alarma al proceso.
- **P_link:** Puntero al siguiente enlace en la cola de Listos (válido si el proceso está Listo para Ejecutar).
- **Estado de Memoria:** Indica si la imagen del proceso se encuentra en memoria principal o secundaria; si está en memoria principal, se indica si puede ser expulsada a swap o si está temporalmente fijada en memoria principal.

Control de procesos

La creación de procesos en UNIX se lleva a cabo mediante la llamada al sistema `fork()`. Al solicitar esta llamada, el sistema operativo realiza las siguientes funciones:

1. **Entrada en la Tabla de Procesos:** Solicita una entrada para el nuevo proceso en la tabla de procesos.
2. **Identificador Único:** Asigna un identificador de proceso único al proceso hijo.
3. **Copia de la Imagen del Proceso:** Realiza una copia de la imagen del proceso padre, exceptuando las regiones de memoria compartidas.
4. **Incremento del Contador de Archivos:** Aumenta el contador de cualquier archivo que posea el padre para reflejar el proceso adicional que también tiene acceso a esos archivos.
5. **Estado del Proceso Hijo:** Asigna al proceso hijo el estado "Listo para Ejecutar".
6. **Retorno de Identificadores:** Devuelve el identificador del proceso hijo al proceso padre y un valor 0 al proceso hijo.

Este trabajo se ejecuta en modo núcleo dentro del proceso padre. Una vez completadas estas funciones, el núcleo puede realizar cualquiera de las siguientes acciones como parte de la rutina del activador:

1. **Continuar con el Proceso Padre:** El control regresa al modo usuario en el punto donde se realizó la llamada a `fork` por parte del padre.

2. **Transferir Control al Proceso Hijo:** El proceso hijo comienza a ejecutar desde el mismo punto del código del padre, es decir, en el punto de retorno de la llamada a `fork`.
3. **Transferir Control a Otro Proceso:** Ambos procesos, padre e hijo, permanecen en el estado "Listos para Ejecutar".

Puede ser un poco complicado visualizar este método de creación de procesos, ya que ambos, padre e hijo, están ejecutando el mismo segmento de código. La diferencia clave es que, al retornar de la función `fork`, se verifica el parámetro de retorno. Si el valor es 0, se está en el proceso hijo, permitiendo bifurcar la ejecución del programa hacia el hijo. Si el valor no es 0, se está en el proceso padre, que continúa con la ejecución principal.servidores

Señales y excepciones

Cuando un sistema operativo desea notificar a un proceso la ocurrencia de un determinado evento, o error, recurre a dos tipos de mecanismos: señales y excepciones.
Las primeras se utilizan en POS IX y las segundas en Windows NT.

Señales

Las señales funcionan de manera similar a las interrupciones, pero para los procesos. Cuando un proceso recibe una señal:

1. **Detiene su ejecución** en la instrucción que está ejecutando.
2. **Ejecuta una rutina de tratamiento** de la señal, cuyo código debe estar dentro del mismo proceso.
3. Tras ejecutar la rutina, **continúa la ejecución** en la instrucción donde fue interrumpido.

Señal → proceso: un proceso puede enviar una señal a otro proceso o a un grupo de procesos siempre que compartan el mismo identificador de usuario (uid).

Señal sistema operativo → proceso: el sistema operativo también toma la decisión de enviar señales a los procesos cuando ocurren determinadas condiciones. Por ejemplo, las excepciones de ejecución programa las convierte el sistema operativo en señales al proceso que ha causado la excepción.

Tipos de señales:

- Excepciones hardware
- Comunicación
- *FIS* asincrona

El **efecto de una señal** es ejecutar una rutina de tratamiento cuando el proceso la recibe, pero para que esto ocurra, el proceso debe **armar** esa señal, es decir, debe estar preparado para manejarla.

Armar una señal significa indicar al sistema operativo qué rutina debe ejecutar cuando el proceso reciba esa señal, lo que se configura usando la función `sigaction` en POSIX.

Algunas señales pueden ser **ignoradas**, lo que el proceso debe indicarle al sistema operativo. En este caso, el sistema operativo simplemente descarta esas señales.

Además, un proceso puede **enmascarar señales**, bloqueándolas temporalmente. Las señales enmascaradas no se desechan, sino que se mantienen pendientes hasta que el proceso decida desenmascararlas.

Si un proceso recibe una señal sin haberla armado o enmascarado, el sistema ejecuta una **acción por defecto**, que generalmente es terminar el proceso (matarlo).

Excepciones

Una **excepción** es un evento que ocurre durante la ejecución de un programa y que interrumpe el flujo normal de ejecución, requiriendo que se ejecute un fragmento de código especial para manejarla.

Las excepciones pueden ser **generadas por el hardware** (como la división por cero o la ejecución de instrucciones ilegales) o por el **software** (como aquellas detectadas y notificadas por el sistema operativo o el propio proceso).

Cuando ocurre una excepción, el **control** se transfiere al sistema operativo, que ejecuta una rutina de tratamiento para esa excepción. Esta rutina crea un **registro de excepción** con información relevante sobre lo ocurrido. Si existe un **manejador** para esa excepción, el sistema operativo transfiere el control a él. Si no, el sistema operativo aborta la ejecución del proceso.

> 📌 En resumen, **las excepciones** suelen estar asociadas con **errores de ejecución** que requieren atención inmediata, mientras que **las señales** son una forma de **comunicación** o **gestión de eventos** que pueden ser controladas por los procesos y no siempre están relacionadas con errores.

2. Hilos, SMP y micronúcleos

Procesos e hilos

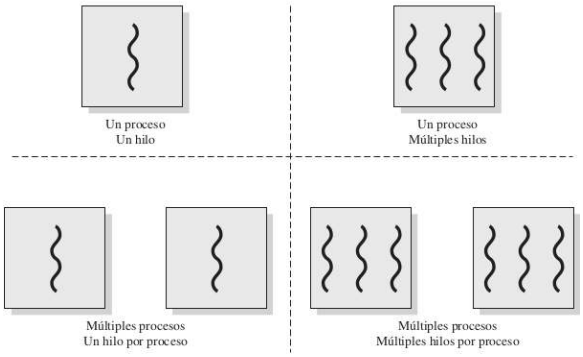
El concepto de proceso tiene dos aspectos clave:

1. **Propiedad de recursos**: Un proceso posee un espacio de direcciones virtuales que contiene su imagen (programa, datos, pila, y atributos) y puede controlar recursos como memoria, dispositivos de E/S, y archivos. El sistema operativo protege estos recursos para evitar interferencias entre procesos.
2. **Planificación y ejecución**: Un proceso sigue una ruta de ejecución y tiene estados (Ejecutando, Listo, etc.), además de una prioridad que define cuándo se ejecuta.

En muchos sistemas operativos, estos dos aspectos funcionan juntos, pero pueden tratarse por separado. En ese caso, la unidad de ejecución se denomina *hilo* o *thread*, mientras que la unidad que controla los recursos sigue siendo el *proceso*.

Multihilo

Multihilo se refiere a la capacidad de un sistema operativo de dar soporte a múltiples hilos de ejecución en un solo proceso. Sin embargo el enfocque tradicional es de un solo hilo, llamado tambien monohilo.



Tipos de sistemas operativos:

- **Monoproceso:** Un solo proceso de usuario y un solo hilo (ejemplo: MS-DOS).
- **Multiproceso:** Múltiples procesos de usuario, pero un solo hilo por proceso (ejemplo: algunas versiones de UNIX).
- **Multihilo:** Un solo proceso con múltiples hilos (ejemplo: entorno de ejecución de Java).

Procesos:

- Unidad de asignación de recursos y protección.
- Tienen un espacio de direcciones virtuales propio, acceso a procesadores, otros procesos, archivos y recursos de E/S.

Hilos:

- Unidad de ejecución dentro de un proceso.
- Tienen su propio estado de ejecución, contexto, pila y variables locales.
- Comparten la memoria y los recursos del proceso con otros hilos del mismo proceso.

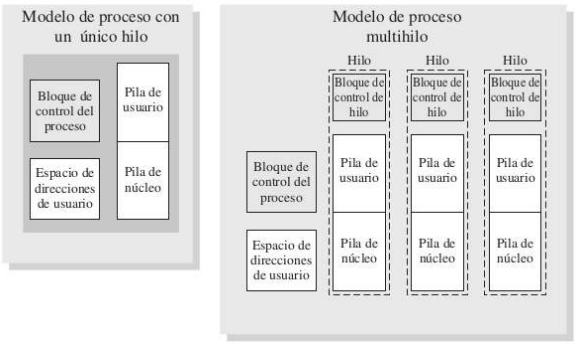
Modelos de proceso:

- **Monohilo:** Cada proceso tiene su propio bloque de control, espacio de direcciones de usuario, pilas y registros del procesador.
- **Multihilo:** Cada proceso tiene varios hilos que comparten el mismo bloque de control y espacio de direcciones de usuario, pero cada hilo tiene su propia pila y un bloque de control específico.

📌 Los sistemas operativos pueden manejar procesos de diferentes maneras. Los sistemas monoproceso solo permiten un proceso a la vez, mientras que los multiproceso permiten varios procesos. Los sistemas multihilo permiten que un solo proceso tenga varios hilos que pueden ejecutarse de forma independiente, compartiendo los recursos del proceso.

Ventajas de los hilos en rendimiento:

1. Crear un hilo dentro de un proceso es mucho más rápido que crear un proceso nuevo (hasta 10 veces más rápido, según estudios en el SO Mach).
2. Finalizar un hilo también es más rápido que finalizar un proceso.
3. Cambiar entre hilos en el mismo proceso toma menos tiempo que cambiar entre procesos.
4. Los hilos mejoran la comunicación entre programas, ya que comparten memoria y archivos, evitando la intervención del núcleo para comunicarse, como sucede entre procesos separados.



Los **hilos** resultan más eficientes que los procesos independientes para ejecutar aplicaciones con múltiples tareas relacionadas. Un ejemplo claro es un servidor de archivos: en lugar de crear un proceso para cada petición, puede iniciar un hilo por cada solicitud. En sistemas multiprocesadores, estos hilos pueden operar en paralelo en distintos procesadores, facilitando la gestión de archivos compartidos y optimizando la coordinación de tareas.

Además, el uso de hilos es beneficioso en aplicaciones que necesitan ejecutar varias funciones en paralelo, incluso en sistemas de un solo procesador. Algunos usos comunes de los hilos en un sistema de multiprocesamiento incluyen:

1. **Trabajo en primer y segundo plano**: permite que una aplicación, como una hoja de cálculo, procese comandos y despliegue opciones simultáneamente, mejorando la percepción de velocidad.
2. **Procesamiento asíncrono**: hilos dedicados pueden realizar tareas de fondo, como una copia de seguridad periódica en un procesador de textos, sin que el programa principal controle el tiempo o la E/S.

3. **Aumento de la velocidad de ejecución**: en sistemas multiprocesadores, un hilo puede realizar operaciones mientras otro realiza E/S, lo que permite que una aplicación no quede bloqueada.
4. **Estructura modular**: facilita el diseño y ejecución de programas complejos que requieren gestionar múltiples tareas o distintas fuentes de entrada y salida.

En un sistema operativo que soporte hilos, la planificación y la activación se realizan a nivel de hilo; de aquí que la mayor parte de la información de estado relativa a la ejecución se mantenga en estructuras de datos a nivel de hilo. Existen, sin embargo, diversas acciones que afectan a todos los hilos de un proceso y que el sistema operativo debe gestionar a nivel de proceso. Suspender un proceso implica expulsar el espacio de direcciones de un proceso de memoria principal 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 los hilos se suspenden al mismo tiempo. De forma similar, la finalización de un proceso finaliza todos los hilos de ese proceso.

Funcionalidades de los hilos

Los hilos, al igual que los procesos, tienen estados de ejecución y se pueden sincronizar entre ellos. A continuación se analizan estos dos aspectos de las funcionalidades de los hilos.

Estado de los hilos

Igual que con los procesos, los principales estados de los hilos son: Ejecutando, Listo y Bloqueado. Generalmente, no tiene sentido aplicar estados de suspensión a un hilo, ya que dichos estados son conceptos de nivel de proceso. En particular, si se expulsa un proceso, todos sus hilos se deben expulsar porque comparten el espacio de direcciones del proceso.

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

1. **Creación**: Al crear un proceso, se genera automáticamente un hilo. Un hilo existente puede crear otro dentro del mismo proceso, asignándole su propio contexto y espacio de pila para colocarlo en la cola de Listos.
2. **Bloqueo**: Si un hilo necesita esperar un evento, se bloquea, almacenando su contexto. Durante este tiempo, el procesador puede ejecutar otros hilos en estado de Listo, ya sea del mismo proceso o de otro.
3. **Desbloqueo**: Al ocurrir el evento esperado, el hilo bloqueado se mueve nuevamente a la cola de Listos.
4. **Finalización**: Cuando un hilo termina, libera su contexto y pila asignados.

Un punto crucial es si el bloqueo de un hilo afecta al proceso completo. Para maximizar la eficiencia, se espera que un hilo bloqueado no detenga la ejecución de otros hilos dentro del mismo proceso, permitiendo así operaciones en paralelo.

Como ejemplo, si un programa necesita realizar dos llamadas a procedimiento remoto (RPC) a servidores distintos, crear un hilo para cada solicitud acelera

el tiempo de espera. Aunque en un sistema uniprocesador las llamadas se ejecutan secuencialmente, el programa aún puede esperar ambas respuestas al mismo tiempo, aprovechando la concurrencia en la espera.

En un uniprocesador, la multiprogramación permite el intercalado de múltiples hilos con múltiples procesos.

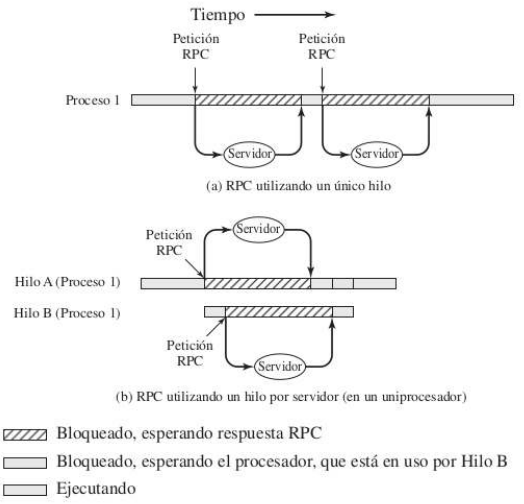


Figura 4.3. Llamadas a Procedimiento Remoto (RPC) utilizando hilos.

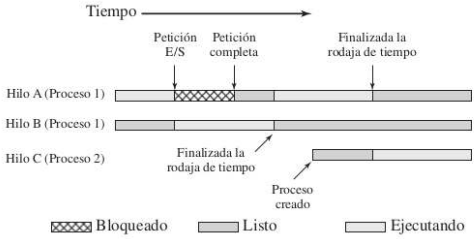


Figura 4.4. Ejemplo multihilo en un uniprocesador.

Sincronización de hilos

Todos los hilos de un proceso comparten el mismo espacio de direcciones y otros recursos, como por ejemplo, los archivos abiertos. Cualquier alteración de un recurso por cualquiera de los hilos, afecta al entorno del resto de los hilos del mismo proceso. Por tanto, es necesario sincronizar las actividades de los hilos para que no interfieran entre ellos o corrompan estructuras de datos. Por ejemplo, si dos hilos de modo simultáneo intentan añadir un elemento a una lista doblemente enlazada, se podría perder un elemento o la lista podría acabar malformada.
Los asuntos que surgen y las técnicas que se utilizan en la sincronización de los hilos son, en general, los mismos que en la sincronización de procesos.

Hilos de nivel de usuario y de nivel de núcleo

Existen dos amplias categorías de implementación de hilos: hilos de nivel de usuario (*user-level threads, ULT*) e hilos de nivel de núcleo (*kernel-level threads, KLT*). Los últimos son también conocidos en la literatura como hilos soportados por el núcleo (*kernel-supported threads*) o procesos ligeros (*lightweight processes*).

Hilos de nivel de usuario.

En un entorno ULT puro, la aplicación gestiona todo el trabajo de los hilos y el núcleo no es consciente de la existencia de los mismos.

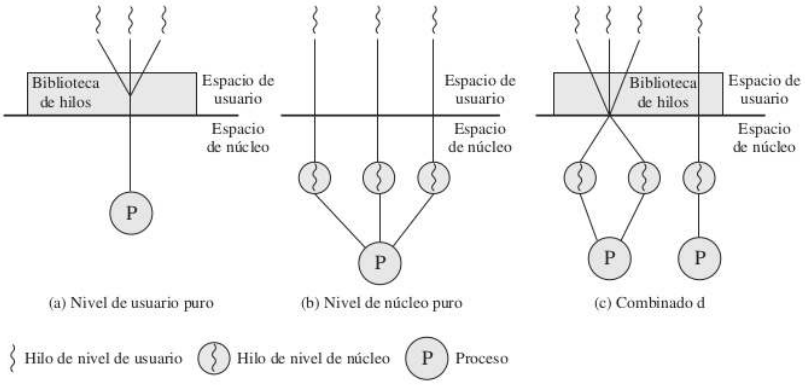


Figura 4.6. Hilos de nivel de usuario y de nivel de núcleo.

Cualquier aplicación puede programarse para ser multihilo a través del uso de una biblioteca de hilos, que es un paquete de rutinas para la gestión de ULT. La biblioteca de hilos contiene código para la creación y destrucción de hilos, para paso de mensajes y datos entre los hilos, para planificar la ejecución de los hilos, y para guardar y restaurar el contexto de los hilos.

En resumen, cualquier aplicación puede convertirse en una aplicación multihilo usando una biblioteca de hilos, la cual proporciona las herramientas necesarias para crear, destruir, y gestionar hilos (ULT, "User-Level Threads") dentro de un proceso.

- **Inicio y Creación de Hilos**: la aplicación comienza con un solo hilo en ejecución dentro de un proceso. Durante la ejecución, el programa puede crear nuevos hilos llamando a una función de la biblioteca de hilos. Esta biblioteca se encarga de crear el hilo y cambiar entre hilos usando un planificador interno.
- **Gestión del Contexto**: cada vez que se cambia de un hilo a otro, la biblioteca guarda el contexto del hilo actual (como registros y punteros de pila) y recupera el contexto del siguiente hilo.
- **Planificación de Procesos e Hilos**: todo este cambio de contexto y la creación de hilos ocurre en el espacio de usuario, sin que el núcleo (SO) intervenga directamente en la planificación de los hilos. El núcleo sigue gestionando el proceso completo, asignándole un único estado y recursos generales.

Este modelo permite que el sistema planifique el proceso como una sola unidad, mientras la biblioteca de hilos se encarga de los cambios entre hilos dentro del proceso sin intervención directa del núcleo.

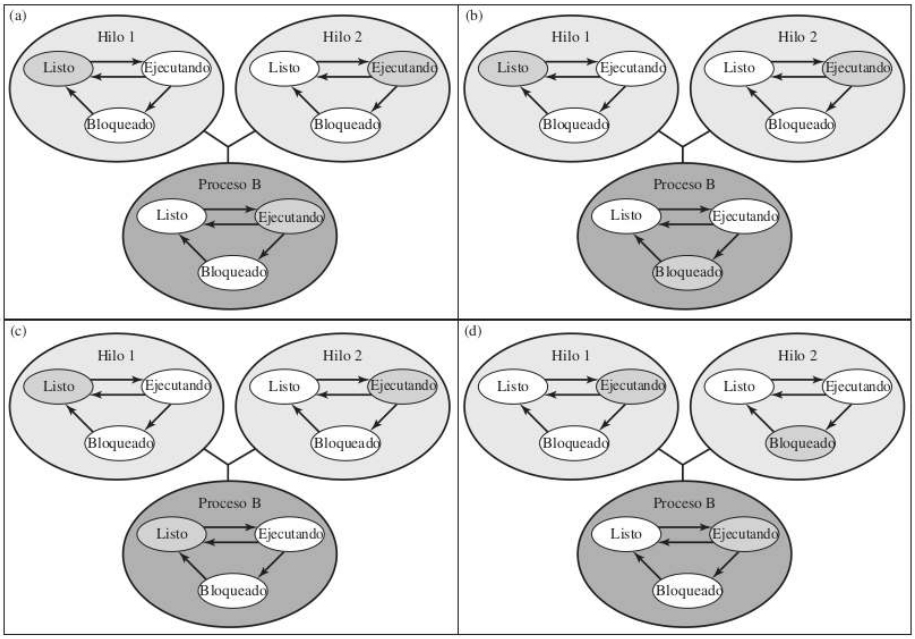


Figura 4.7. Ejemplos de relaciones entre los estados de los hilos de nivel de usuario y los estados de proceso.

El uso de hilos a nivel de usuario (ULT) en vez de hilos a nivel de núcleo (KLT) tiene estas ventajas:

1. **Eficiencia**: Cambiar de hilo no requiere cambiar a modo núcleo, lo cual ahorra tiempo al evitar el cambio de contexto al sistema operativo.
2. **Flexibilidad en la planificación**: Las aplicaciones pueden personalizar su algoritmo de planificación de hilos sin afectar al sistema operativo.
3. **Compatibilidad**: Los ULT funcionan en cualquier sistema operativo sin modificaciones en el núcleo.

Sin embargo, presentan desventajas:

1. **Bloqueo global**: Si un ULT realiza una llamada bloqueante, todos los hilos del proceso se bloquean.
2. **Limitación en multiprocesadores**: En sistemas con múltiples procesadores, los ULT no pueden ejecutarse concurrentemente en diferentes procesadores, lo que limita su rendimiento en aplicaciones que podrían aprovechar la concurrencia.

Hay varias formas de afrontar estos dos problemas. Por ejemplo:

- Ambos problemas pueden superarse escribiendo una aplicación de múltiples procesos en lugar de múltiples hilos. Pero este enfoque elimina la principal ventaja de los hilos: cada cambio es un cambio de proceso en lugar de un cambio de hilo, lo que genera una gran sobrecarga.
- Otra forma de solucionar el problema de hilos que se bloquean es una técnica denominada *jacketing* (revestimiento). El objetivo de esta técnica es convertir una llamada al sistema bloqueante en una llamada al sistema no bloqueante. Por ejemplo, en lugar de llamar directamente a una rutina del sistema de E/S, un hilo puede llamar a una rutina *jacket* de E/S a nivel de

aplicación. Con esta rutina *jacket*, el código verifica si el dispositivo de E/S está ocupado. Si lo está, el hilo entra en estado Bloqueado y pasa el control (a través de la biblioteca de hilos) a otro hilo. Cuando este hilo recupera denuevo el control, chequea de nuevo el dispositivo de E/S.

Hilos a nivel de núcleo

En un sistema de hilos a nivel de núcleo (KLT), el núcleo gestiona directamente los hilos, y la aplicación solo usa una API para trabajar con ellos. Windows es un ejemplo de este modelo. En este enfoque:

1. **Ventajas**:
   - **Ejecución en paralelo**: El núcleo puede planificar varios hilos de un proceso en distintos procesadores, permitiendo una verdadera concurrencia en sistemas multiprocesador.
   - **Mejor manejo de bloqueos**: Si un hilo se bloquea, el núcleo puede activar otro hilo del mismo proceso.
   - **Soporte en el núcleo**: Incluso las funciones del núcleo pueden ser multihilo.
2. **Desventaja**:
   - **Cambio de contexto**: Cambiar de un hilo a otro dentro del mismo proceso implica un cambio de modo a núcleo, lo cual puede reducir la eficiencia.

En general, los KLT mejoran la ejecución concurrente y el manejo de bloqueos comparado con los ULT, aunque con una ligera penalización en la velocidad de cambio entre hilos del mismo proceso.

En resumen, tanto los ULT como los KLT ofrecen ventajas sobre el uso de procesos de un solo hilo, con los ULT siendo más eficientes al no requerir acceso al modo núcleo para cambiar entre hilos. Sin embargo, la elección entre ULT y KLT depende del tipo de aplicación: si una aplicación necesita cambios de hilo que involucren el modo núcleo frecuentemente, los KLT se vuelven una opción más adecuada, ya que los ULT pierden parte de su eficiencia en esos casos.

Enfoques combinados

Algunos sistemas operativos, como Solaris, utilizan un enfoque híbrido que combina ULT y KLT. En este modelo, los ULT se crean y gestionan en el espacio de usuario, mientras que varios ULT se asignan a un número ajustable de KLT, los cuales se encargan de la ejecución en el núcleo. Esto permite que los hilos de una aplicación se ejecuten en paralelo en varios procesadores y evita que una llamada bloqueante detenga todo el proceso. Así, el modelo híbrido busca aprovechar las ventajas de ULT y KLT, minimizando sus desventajas.

Otras configuraciones

Existen otras configuraciones para gestionar hilos y procesos más allá del modelo clásico de relación uno-a-uno. Aquí se exploran las relaciones muchos-a-muchos y uno-a-muchos:

1. **Muchos-a-muchos**: Permite que varios hilos compartan múltiples dominios de ejecución (espacios de memoria separados). En el sistema TRIX, por ejemplo, un solo hilo puede moverse entre dominios, permitiendo que un programa principal y un subprograma de E/S compartan memoria sin la necesidad de crear procesos adicionales. Esto reduce la sobrecarga de gestión y permite una eficiencia superior.
2. **Uno-a-muchos**: Popular en sistemas operativos distribuidos como Clouds y Emerald, permite mover un hilo entre diferentes espacios de direcciones e incluso entre distintas máquinas. Este enfoque es útil en entornos distribuidos, donde el sistema operativo puede gestionar el movimiento del hilo para optimizar el acceso a recursos y balancear la carga.

Gestión de hilos y SMP en Solaris

Solaris implementa un soporte de hilo multinivel poco habitual, diseñado para proporcionar considerable flexibilidad para sacar provecho de los recursos del procesador.

Arquitectura multihilo

Solaris utiliza cuatro conceptos relacionados con los hilos:

- **Procesos:** es un proceso normal UNIX e incluye el espacio de direcciones del usuario, la pila y el bloque de control del proceso.
- **Hilos de nivel de usuario:** implementados a través de una biblioteca de hilos en el espacio de direcciones de un proceso, son invisibles al sistema operativo. Los hilos de nivel de usuario (*user-level threads*, ULT) son la interfaz para las aplicaciones paralelas.
- **Procesos ligeros:** un proceso ligero (*lightweight process*, LWP) puede ser visto como una asociación entre ULT e hilos de núcleo. Cada LWP soporta uno o más ULT y se asocia con un hilo de núcleo. Los LWP se planifican de forma independiente por el núcleo y pueden ejecutar en paralelo en múltiples procesadores.
- **Hilos de núcleo:** son entidades fundamentales que se pueden planificar para ejecutar en cualquier procesador del sistema.

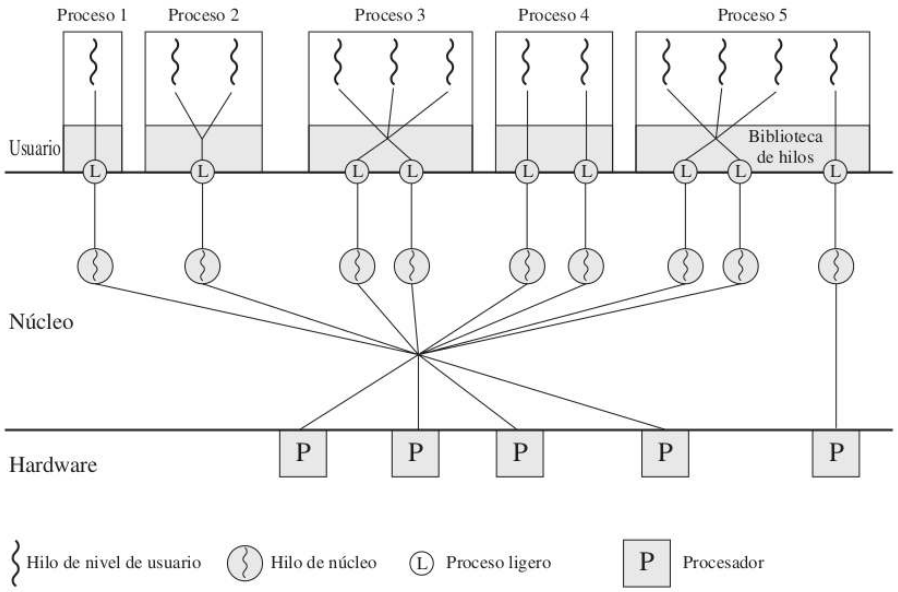


Figura 4.15. Ejemplo de arquitectura multihilo de Solaris.

Existen distintas configuraciones para manejar ULT (hilos de nivel de usuario) y LWP (hilos ligeros de núcleo):

1. **Proceso 1**: Consiste en un único ULT vinculado a un único LWP, como en un proceso UNIX tradicional. Es útil cuando no se necesita concurrencia.
2. **Proceso 2**: Usa una estrategia ULT pura, donde todos los ULT se apoyan en un solo LWP. Esto permite expresar concurrencia sin ejecutar hilos en paralelo.
3. **Proceso 3**: Multiplexa varios ULT en un menor número de LWP, lo que permite ajustar el paralelismo a nivel de núcleo, útil para procesos que no requieren uso intensivo de CPU.
4. **Proceso 4**: Cada ULT tiene un LWP exclusivo, exponiendo completamente el paralelismo al núcleo, ideal para hilos que se bloquean frecuentemente.
5. **Proceso 5**: Combina la multiplexación y el enlace exclusivo de ULT con LWP, donde incluso algunos LWP están asociados directamente a un procesador, optimizando aún más el rendimiento en entornos paralelos.

Hilos de núcleo: Estos hilos, creados y gestionados por el núcleo, manejan tareas específicas del sistema, permitiendo una menor sobrecarga de cambios de contexto comparado con el uso de procesos de núcleo.

Estructura de los procesos

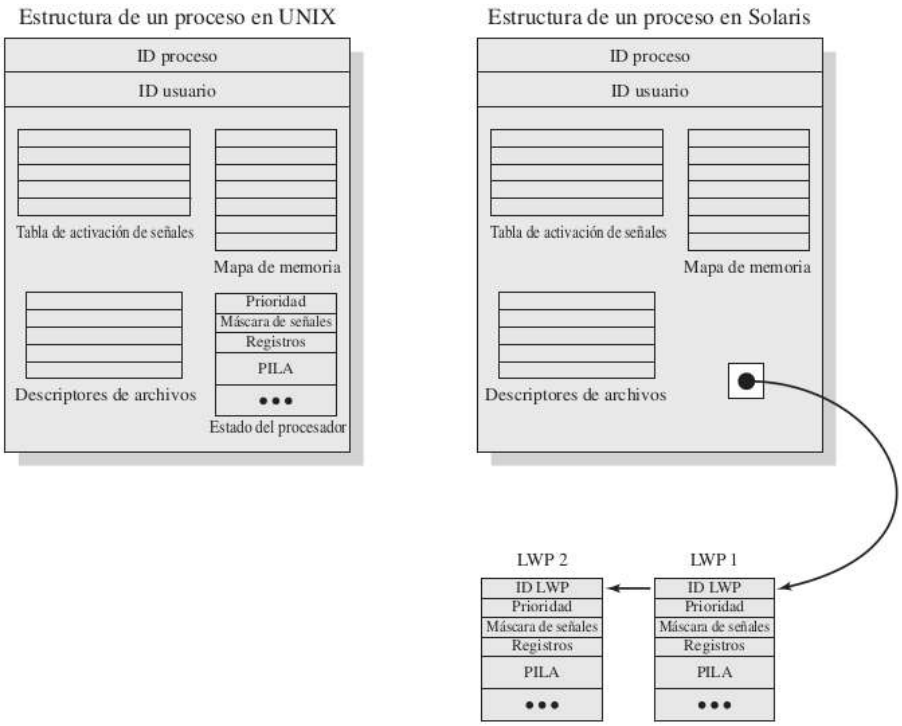


Figura 4.16. Estructura de procesos en UNIX tradicional y Solaris.

Ejecución de hilos

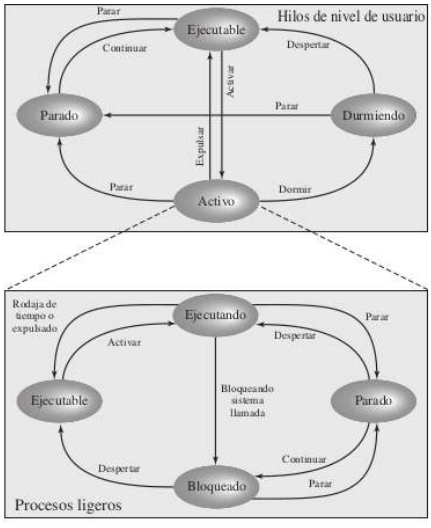


Figura 4.17. Estados de los hilos de nivel de usuarios y LWP en Solaris.

La gestión de hilos a nivel de usuario es esponsabilidad de la biblioteca de hilos y se basa en el estado de los hilos no vinculados (hilos que comparten varios LWP). Los estados posibles de estos hilos son:

- **Ejecutable**: Listo para ejecutar, pero sin LWP asignado.
- **Activo**: Ejecutando en un LWP.
- **Durmiendo**: Suspendido temporalmente, en espera de que se cumpla una condición.
- **Detenido**: Suspendido hasta recibir una señal de continuación.

Eventos que pueden afectar a un hilo activo (por ejemplo, T1):

- **Sincronización**: T1 se duerme para coordinarse con otros hilos y pasa a ejecutable cuando se cumple la condición.
- **Suspensión**: T1 es detenido por petición de otro hilo y no reanuda hasta recibir una señal.
- **Expulsión**: Si un hilo T2 de mayor prioridad se vuelve ejecutable, T1 cede su lugar.
- **Ceder paso**: Si T1 ejecuta `thr_yield()`, cede el LWP a otro hilo de la misma prioridad (T2) si está disponible.

Cada vez que un hilo sale de estado activo, la biblioteca asigna el LWP disponible a otro hilo ejecutable.

Interrupciones como hilos

En Solaris, se unifican procesos e interrupciones en un solo modelo al convertir las interrupciones en hilos de núcleo, lo que permite un manejo más eficiente y controlado. Este enfoque ayuda a reducir la sobrecarga y simplificar la sincronización, especialmente en sistemas multiprocesador, donde bloquear interrupciones en todos los núcleos es costoso.

La implementación de Solaris incluye:

1. **Hilos de núcleo para interrupciones**: Cada hilo de interrupción tiene su propio contexto y prioridad.
2. **Sincronización con exclusión mutua**: Usa las mismas primitivas que otros hilos de núcleo para proteger los datos compartidos.
3. **Alta prioridad**: Los hilos de interrupción reciben la prioridad más alta para minimizar tiempos de espera.

Cuando ocurre una interrupción, el hilo en ejecución se suspende temporalmente (queda "inmovilizado") y se ejecuta un hilo de interrupción preexistente para procesar la señal. Este hilo de interrupción solo espera si necesita acceder a datos bloqueados por otro hilo, y solo puede ser expulsado por una interrupción de mayor prioridad. En general, esta estrategia mejora el rendimiento sobre los enfoques tradicionales de manejo de interrupciones.

Gestión de procesos e hilos en Linux

Tareas Linux

Un proceso, o tarea, en Linux se representa por una estructura de datos `task_struct`, que contiene información de diversas categorías:

- **Estado**: Un proceso puede estar en varios estados, como "Ejecutando" (en ejecución o listo para ejecutarse), "Interrumpible" (esperando un evento), "Ininterrumpible" (esperando directamente al hardware sin atender señales), "Detenido" (pausado hasta que otro proceso lo reanude), y "Zombie" (proceso terminado que sigue en la tabla de procesos).
- **Planificación**: Incluye datos como si es de tiempo real o normal, su prioridad, y el tiempo acumulado en ejecución.

- **Identificadores**: Cada proceso tiene un ID único, además de identificadores de usuario y grupo para definir permisos.
- **IPC**: Soporta comunicación entre procesos, similar a UNIX SVR4.
- **Enlaces**: Cada proceso tiene enlaces a su proceso padre, hermanos (otros procesos del mismo padre), y sus hijos.
- **Tiempos y temporizadores**: Registra su tiempo de creación y uso de CPU, y puede tener temporizadores que desencadenan señales al agotarse.
- **Sistema de archivos**: Mantiene punteros a archivos abiertos, así como a los directorios actual y raíz.
- **Espacio de direcciones**: Define su espacio de direcciones virtual.
- **Contexto específico del procesador**: Incluye los registros y la pila que corresponden al contexto del proceso.

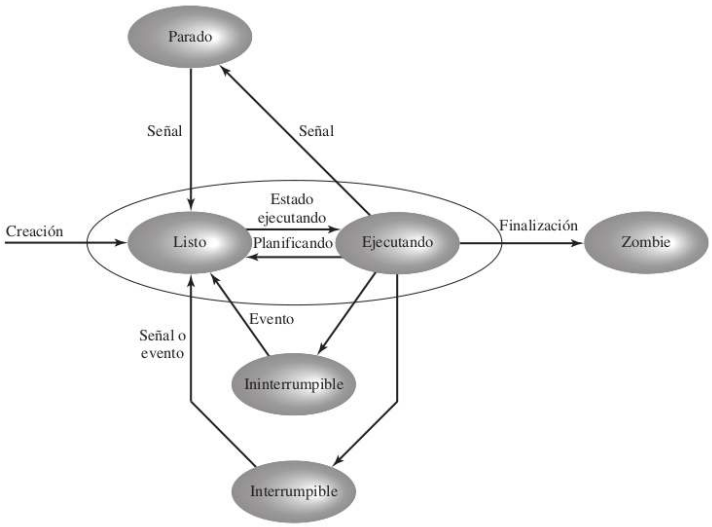


Figura 4.18. Modelo de procesos e hilos en Linux.

Hilos Linux

Los sistemas UNIX antiguos solo permitían un hilo por proceso. Hoy en día, los sistemas modernos, incluido Linux, soportan varios hilos a nivel de núcleo dentro de un mismo proceso. Antes, Linux no tenía soporte nativo para multihilos y usaba bibliotecas como `pthreads` para simularlos.

Ahora, Linux usa un esquema de "procesos ligeros" similar a Solaris, donde no diferencia entre procesos e hilos; usa el comando `clone()` en vez de `fork()`. Este permite que múltiples hilos compartan recursos como la memoria y archivos, aunque cada hilo tiene su propia pila.

3. Planificación de procesos e hilos

Planificación

La planificación de procesos tiene como objetivo repartir el tiempo del procesador entre los procesos listos para ejecutar. El planificador selecciona el proceso que pasará a ejecución, mientras que el activador lo pone en marcha. Los sistemas pueden tener varios niveles de planificación:

1. **Planificación a largo plazo**: en sistemas decide qué programas se procesarán, controlando el nivel de multiprogramación. Cuando se aprueba un trabajo, se convierte en proceso y entra en la cola del planificador a corto plazo, o puede ir a la zona de intercambio para el planificador a medio plazo.

   En sistemas por lotes, los trabajos nuevos esperan en el disco hasta que el planificador los convierte en procesos según la prioridad, el tiempo de ejecución estimado o necesidades de E/S, optimizando el uso de CPU y recursos. En sistemas de tiempo compartido, los usuarios interactivos se aceptan hasta el límite; si está saturado, los nuevos usuarios deben esperar.

2. **Planificación a medio plazo**: Gestiona la suspensión de procesos, decidiendo cuáles pasan a estado suspendido o se reactivan, controlando el grado de multiprogramación.
3. **Planificación a corto plazo**: Selecciona el proceso listo que se asignará al procesador para su ejecución.

   El planificador a corto plazo se invoca siempre que ocurre un evento que puede conllevar el bloqueo del proceso actual y que puede proporcionar la oportunidad de expulsar al proceso actualmente en ejecución en favor de otro.

   Algunos ejemplos de estos eventos son:

   - Interrupciones de reloj.
   - Interrupciones de E/S.
   - Llamadas al sistema.
   - Señales (por ejemplo, semáforos).
4. **Planificación de entrada/salida** organiza el orden en que se ejecutan las operaciones de E/S encoladas para cada periférico.

Expulsión

La planificación puede ser con o sin expulsión:

- **Sin expulsión**: Un proceso mantiene el procesador mientras no solicite un servicio que lo bloquee. Minimiza el tiempo de planificación y activación, pero puede causar que un proceso monopolice el procesador, especialmente si entra en un bucle infinito.

- **Con expulsión**: El sistema operativo puede quitar un proceso del estado de ejecución, incluso si no lo solicita. Permite controlar el tiempo de ejecución de los procesos, pero requiere que el sistema operativo intervenga periódicamente, gracias a interrupciones del reloj, para verificar si es necesario cambiar el proceso en ejecución.

Objetivos de la planificación

El objetivo de la planificación es optimizar el comportamiento del sistema. Ahora bien, el comportamiento de un sistema informático es muy complejo, por tanto, el objetivo de la planificación se deberá centrar en la faceta del comportamiento en el que se esté interesado. Entre los objetivos que se suelen perseguir están los siguientes:

- Reparto equitativo del procesador.
- Eficiencia (optimizar el uso del procesador).
- Menor tiempo de respuesta en uso interactivo.
- Menor tiempo de espera en lotes (*batch*).
- Mayor número de trabajos por unidad de tiempo (*batch*).
- Cumplir los plazos de ejecución de un sistema de tiempo real.

La mayoría de estos objetivos son incompatibles entre sí, por lo que hay que centrar la atención en aquel que sea de mayor interés.

Algoritmos de planificación

📌 Esto esta explicado en folios a parte