juanfrandm98
www.wuolah.com/student/juanfrandm98
2009

Tema-2.pdf

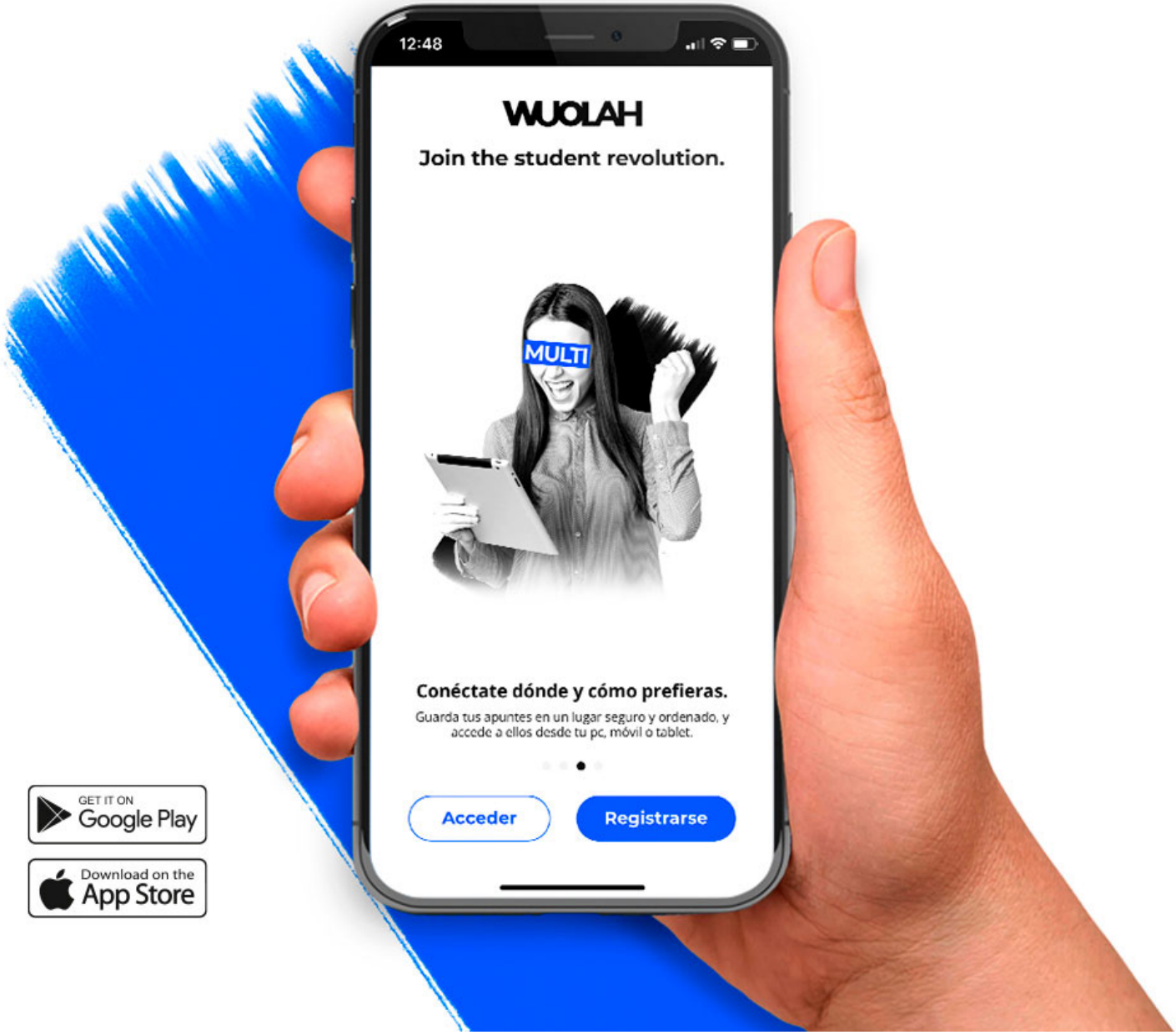
Apuntes de los Libros

2º Sistemas Operativos

Grado en Ingeniería Informática

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

Exámenes, preguntas, apuntes.



GET IT ON Google Play

Download on the App Store

UNIVERSIDAD DE GRANADA

Sistemas Operativos

Tema 2. Procesos y hebras

Juan Francisco Díaz Moreno

Curso 20/21

¿Qué es un proceso?

Procesos y Bloques de Control de Procesos

Un **proceso** es una instancia de un programa ejecutado en un computador asignada a un procesador. Tiene dos elementos esenciales: el código del programa y el conjunto de datos asociados.

Durante cualquier instante puntual de la ejecución de un proceso, este se puede caracterizar por una serie de elementos:

- **Identificador**. Es único, para distinguirlo del resto de procesos.
- **Estado**. Si está actualmente corriendo, está en estado de ejecución.
- **Prioridad**. Nivel relativo al resto de procesos.
- **Contador de programa**. Dirección de la siguiente instrucción del programa que se ejecutará.
- **Punteros a memoria**. Al código del programa y a los datos asociados.
- **Datos de contexto**. Datos presentes en los registros del procesador.
- **Información de estado de E/S**. Peticiones de E/S pendientes, dispositivos de E/S asignados, ficheros en uso…
- **Información de auditoría**. Cantidad de tiempo de procesador y tiempo de reloj utilizados, límites de tiempo, registros contables…

Todos estos elementos se almacenan en una estructura de datos conocida como **bloque de control de proceso** (PCB), creada y gestionada por el SO. Este contiene la información necesaria para interrumpir el proceso y restaurarlo posteriormente como si no hubiera habido ninguna interrupción, permitiendo al SO la multiprogramación.

Cuando un proceso se interrumpe, los valores actuales del contador de programa y los registros del procesador se guardan en los campos correspondientes del PCB y el estado del proceso se cambia a *bloqueado* o *listo*. El SO puede entonces poner otro proceso en ejecución recuperando sus datos de su respectivo PCB.

Estados de los procesos

La **traza** de un proceso es la secuencia de instrucciones que se ejecutan por él. Se puede caracterizar el comportamiento de un procesador mostrando cómo se entrelazan las trazas de distintos procesos.

El activador (***dispatcher***) es el programa que se encarga de intercambiar el procesador de un proceso a otro. Este proceso tiene distintas formas de actuación. Un ejemplo sería limitar las instrucciones que puede realizar un proceso cada vez, tras las cual se produciría una alarma de temporización (*time-out*) y el *dispatcher* realizaría el cambio de contexto. Este también actuaría en caso de que el proceso en ejecución realizase una petición de E/S.

Un modelo de proceso de dos estados

La misión principal del SO es controlar la ejecución de los procesos, determinando cómo se entrelazan sus ejecuciones y qué recursos se asignan a cada uno.

El modelo de comportamiento más simple para los procesos es considerar si están siendo ejecutados por el procesador o no (*Ejecutando* o *No ejecutando*).

Cuando el SO crea un proceso, crea su PCB y le asigna el estado *No ejecutando*, de forma que espera su oportunidad para ejecutar hasta que el *dispatcher* determine que termina la ejecución de otro programa y seleccione a otro par ejecutar. El proceso saliente pasará al estado *No ejecutando* y el nuevo proceso pasará a *Ejecutando*.

Para que esta primera aproximación funcione, el SO debe conocer en todo momento a todos los procesos, y lo consigue gracias a sus PCB. Los procesos que no se estén ejecutando se van incluyendo en una cola, cuyas entradas son punteros a los respectivos PCB.

El *dispatcher* se encarga, cuando se interrumpe la ejecución de un proceso, de colocarlo al final de la cola de procesos (si aún no ha terminado su ejecución) o de descartarlo (si ha terminado); y de seleccionar al siguiente proceso de la cola.

Creación de procesos

Cuando se va a añadir un nuevo proceso, el SO construye su PCB y reserva el espacio de direcciones en memoria principal que necesite.

Existen cuatro eventos comunes que llevan a la creación de procesos:

- Nuevo proceso de lotes: un proceso se crea como respuesta a una solicitud de trabajo.
- Sesión interactiva: un proceso se crea cuando un nuevo usuario entra en el sistema.
- Creado por el SO para proporcionar un servicio.
- Creado por un proceso existente, por motivos de modularidad o para explotar el paralelismo.

Tradicionalmente, el SO creaba todos los procesos de forma que era transparente para usuarios y programas. Esto aún es común, pero puede ser muy útil permitir a un proceso crear otro.

Cuando un proceso lanza otro, al primero se le denomina **proceso padre** y al segundo, **proceso hijo**. Normalmente es necesaria la comunicación y cooperación entre ambos.

Terminación de procesos

Todo sistema debe proporcionar los mecanismos mediante los cuales un proceso indica su finalización. Un proceso por lotes debe incluir una instrucción *HALT* o una llamada a un servicio del SO específica para su terminación. La instrucción *HALT* generará una interrupción para indicar al Soque dicho proceso ha finalizado.

Para una aplicación interactiva, serán las acciones del usuario las que indiquen cuando el proceso ha finalizado.

También puede finalizar un proceso debido a un error o a una condición de fallo.

Otro ejemplo de terminación es cuando el proceso padre decide su finalización o él mismo finaliza.

Modelo de proceso de cinco estados

Con un modelo de dos estados, puede ocurrir que un proceso que esté en el estado de *No ejecutando* se encuentre bloqueado debido a una operación de E/S. El *dispatcher*, que utiliza una estrategia cíclica (*round-robin*) sobre una cola de procesos FIFO, no puede solucionar esta situación, pues debería recorrer la cola buscando el primer proceso que no esté bloqueado.

Un modelo más natural consistiría en la utilización de los siguientes cinco estados:

- **Ejecutando**. El proceso está en ejecución (vamos a asumir que sólo se utiliza un único procesador, por lo que sólo un proceso puede estar en este estado en cualquier instante de tiempo).
- **Listo**. Un proceso preparado para ejecutar cuando tenga la oportunidad.
- **Bloqueado**. Un proceso que no puede ejecutar hasta que se cumpla un evento determinado, como una operación de E/S.
- **Nuevo**. Un proceso recién creado que aún no ha sido admitido en la cola de procesos. Todavía no ha sido cargado en memoria principal, aunque tenga su PCB creado.
- **Saliente**. Un proceso que ha sido expulsado de la lista de procesos ejecutables, ya sea porque ha terminado su ejecución o porque ha sido abortado. En este punto, el proceso aún tiene memoria reservada.

En este modelo, las posibles transiciones de estados son las siguientes:

- **Null -> Nuevo.** Se crea un nuevo proceso para ejecutar un programa.
- **Nuevo -> Listo.** El SO realiza este cambio cuando esté preparado para ejecutar. Algunos SO tienen un límite en el número de procesos existentes o en la cantidad de memoria virtual utilizada por procesos existentes para evitar que se degrade el rendimiento del sistema.
- **Listo -> Ejecutando.** Cuando el procesador está libre, el SO selecciona uno de los procesos en estado *Listo* a través del planificador.
- **Ejecutando -> Saliente.** El SO finaliza su ejecución si el proceso indica que se ha completado o si es abortado.
- **Ejecutando -> Listo.** La razón principal es que el proceso ha superado la restricción de tiempo establecida (*quantum*). También puede ocurrir que un proceso de prioridad mayor pase a estado *Listo* y el SO, interrumpa el proceso actual para ejecutar el nuevo; o que el proceso en ejecución deje voluntariamente de utilizar el procesador, por ejemplo, cuando realiza alguna función de mantenimiento de forma periódica.
- **Ejecutando -> Bloqueado.** El proceso en ejecución realiza una llamada al sistema cuando solicita un recurso o servicio que no está disponible; o cuando necesita esperar a que un evento termine, como una operación de E/S o de comunicación.

- **Bloqueado -> Listo.** Esto ocurre cuando sucede el evento por el cual un proceso estaba esperando.
- **Listo -> Saliente.** Un padre puede terminar la ejecución de un hijo en cualquier momento, o siempre que termina su propia ejecución.
- **Bloqueado -> Saliente.** Como en el caso anterior.

Este modelo se puede implementar con dos colas, una para procesos listos y otra para procesos bloqueados. Cuando el procesador termina la ejecución de un proceso, se coloca en la cola adecuada.

Cada vez que un evento tiene lugar, el SO debe recorrer toda la cola de bloqueados buscando los procesos que esperasen a dicho evento. Esta cola puede ser enorme, por lo que sería mucho más eficiente tener una cola para cada evento, de tal forma que cuando un evento ocurra, toda su cola se movería a la cola de listos.

También se podría crear una cola de listos por cada nivel de prioridad, de tal forma que el proceso buscaría primero en la cola de mayor prioridad.

Procesos suspendidos

Muchos SO se implementan en torno a estos tres estados principales (Listo, Ejecutando y Bloqueado). Sin embargo, pueden añadirse otros que beneficiarían en especial a sistemas que no utilicen memoria virtual, en los que cada proceso que se ejecute debe cargarse completamente en memoria principal.

La necesidad de estos procesos reside en que las operaciones de E/S son mucho más lentas que los procesos de cómputo, por lo que el procesador de un sistema monoprogramado estaría ocioso la mayor parte del tiempo. La diferencia de velocidad entre el procesador y la E/S provocaría que fuese muy habitual que todos los procesos en memoria se encontrasen esperando estas operaciones.

Para solucionarlo, en primer lugar, la memoria principal puede expandirse para acomodar más procesos. Esto provocaría dos fallos: el aumento del coste y la creciente necesidad de memoria por parte de los programas.

Otra solución es el ***swapping*** (memoria de intercambio), que implica mover parte o todo el proceso de memoria principal al disco. Cuando ninguno de los procesos en memoria principal se encuentra en estado *Listo*, el SO intercambia procesos bloqueados a disco, el la cola de Suspendidos. Esta es una lista de procesos existentes que han sido temporalmente expulsados de la memoria principal, o suspendidos. El SO trae otro proceso de esta cola o responde a solicitudes de nuevos procesos.

El *swapping* es una operación de E/S, por lo que podría hacer que el problema empeore. Pero debido a que la E/S en disco es habitualmente más rápida que la E/S sobre otros sistemas, el swapping habitualmente mejora el rendimiento del sistema.

De esta forma, el swapping añadiría un nuevo estado al modelo: Suspendido. El hecho de que el SO libere memoria de presión gracias a este estado permite que pueda ser utilizada por otros procesos.

Cuando el SO ha realizado una operación de *swap*, puede traer a memoria principal un nuevo proceso creado o un proceso en estado *Suspendido*, lo que a priori podría parecer mejor para evitar incrementar la carga total del sistema. Sin embargo, estos procesos se encontraban previamente en estado *Bloqueado*, y es posible que aún no haya ocurrido el evento al que estaban esperando.

Para evitar esto, se podrían utilizar 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 esta disponible para su ejecución en cuanto se cargue en memoria principal.

Además, con un esquema de memoria virtual, es posible ejecutar un proceso que está solo parcialmente en memoria principal. Si se hace referencia a una dirección de proceso que no se encuentra en memoria principal, la porción correspondiente se trae a ella. El uso de la memoria principal podría parecer que elimina la necesidad del *swapping* explícito, porque cualquier dirección necesitada por un proceso puede traerse a memoria por el propio hardware de gestión de memoria del procesador. Sin embargo, el rendimiento de los sistemas de memoria virtual puede colapsarse si hay un numero suficientemente grande de procesos activos, todos parcialmente en memoria principal. Así, incluso con un sistema de memoria virtual, el SO necesitaría hacer *swapping* con la intención de mejorar el rendimiento.

Con el *swapping*, las nuevas transiciones más importantes serían las siguientes:

- **Bloqueado -> Bloqueado/Suspendido.** Si no hay procesos listos, alguno de los procesos bloqueados se transfiere a disco para hacer espacio para otro proceso que no se encuentre bloqueado. Esto ocurre también si los procesos listos requieren más memoria principal.
- **Bloqueado/Suspendido -> Listo/Suspendido.** Ocurre cuando sucede el evento por el cual un proceso estaba esperando en memoria secundaria. Esto requiere que la información de estos procesos sea accesible para el SO.
- **Listo/Suspendido -> Listo.** Esto ocurre cuando no hay procesos listos en memoria principal o cuando un proceso en estado *Listo/Suspendido* tiene mayor prioridad que cualquiera de los procesos listos.
- **Listo -> Listo/Suspendido.** A pesar de que el SO preferirá suspender procesos bloqueados, puede suspender también procesos listos si con ello consigue liberar una buena cantidad de memoria. También podría hacerlo si determina que un proceso bloqueado de mayor prioridad pronto estará listo.

Otras transiciones interesantes a considerar son las siguientes:

- **Nuevo -> Listo/Suspendido y Nuevo -> Listo.** Cuando se crea un proceso nuevo, puede añadirse a la cola de Listos o la cola de Listos/Suspendidos. En cualquiera de los casos, el SO crearía su PCB y reservaría el espacio de direcciones del proceso. Si no hay suficiente espacio en memoria principal para el nuevo proceso, se enviaría a la cola de Listos/Suspendidos.
- **Bloqueado/Suspendido -> Bloqueado.** Esto se podría dar cuando la prioridad de un proceso en la cola de Bloqueados/Suspendidos es mucho mayor que la de los procesos

de la cola de Listos/Bloqueados y el SO predice que el evento al que espera está cerca de finalizar.
- **Ejecutando -> Listo/Suspendido.** El SO puede expulsar un proceso en ejecución cuando un proceso de mayor prioridad de la cola de Bloqueados/Suspendidos acaba de desbloquearse, y mover directamente el proceso a la cola de Listos/Suspendidos para liberar memoria principal.
- **De cualquier estado -> Saliente.** Cuando un proceso padre termina, sus hijos también lo hacen.

Otros usos para la suspensión de procesos

Un proceso suspendido puede ser aquel que cumple las siguientes características:

- El proceso no está inmediatamente disponible para su ejecución.
- El proceso puede estar o no a la espera de un evento. Si el evento sucede, no habilita al proceso para su ejecución inmediata.
- El proceso fue puesto en estado suspendido por un agente.
- El proceso no puede ser recuperado hasta que el agente lo indique explícitamente.

Un proceso puede ser suspendido por distintas razones. La principal es para traer procesos en estado Listo/Suspendido o incrementar el espacio de memoria disponible. También puede ser una alternativa para procesos periódicos o procesos que puedan tener problemas. También puede ser el usuario quien puede dejar procesos en segundo plano o suspenderlos en caso de sospecha de *bug*. Por último, un proceso padre puede suspender a un proceso hijo.

En cualquier caso, la activación de un proceso suspendido se solicita por medio del agente que inicialmente puso al proceso en suspensión.

Descripción de procesos

Fundamentalmente, el SO es la entidad que gestiona el uso de recursos del sistema por parte de los procesos: controla eventos, planifica, activa procesos, reserva recursos… En un entorno multiprogramado, hay numerosos procesos que irán necesitando acceder a ciertos recursos, como procesador, dispositivos de E/S o memoria principal.

Estructuras de control del Sistema Operativo

El SO se encarga de la gestión de procesos y recursos, por lo que debe disponer de información sobre el estado actual de cada uno de ellos. Para ello, mantienen tablas de cada tipo de entidad que gestiona, dividiéndose en memoria, E/S, ficheros y procesos.

Las **tablas de memoria** se usan para mantener un registro tanto de la memoria principal (real) como de la secundaria (virtual). Parte de la memoria principal está reservada

para el SO, el resto está disponible para los procesos, que se mantienen en memoria secundaria utilizando técnicas de *swapping* o memoria virtual. Las tablas de memoria incluyen:

- Las reservas de memoria principal por parte de los procesos.
- Las reservas de memoria secundaria por parte de los procesos.
- Todos los atributos de protección que restringen el uso de la memoria principal y virtual, permitiendo que los procesos puedan acceder a memoria compartida.
- La información necesaria para manejar la memoria virtual.

El SO utiliza las **tablas de E/S** para gestionar los dispositivos de E/S y los canales del computador. El SO mantiene para cada dispositivo si está disponible o asignado a un proceso; y debe conocer el estado de la operación y la dirección de memoria principal usada como fuente o destino de la transferencia.

Las **tablas de ficheros** del SO proporcionan información sobre la existencia de ficheros, su posición en almacenamiento secundario, su estado y otros atributos. En algunos SO, esta información se puede gestionar por el sistema de ficheros en lugar del SO.

Por último, la gestión de procesos se realiza a través de las **tablas de procesos**. Estas también están entrelazadas y referenciadas entre sí, ya que memoria, E/S y ficheros se gestionan por parte de los procesos, además del SO.

Para la creación de las tablas iniciales, el SO debe tener información sobre el entorno básico, como la memoria física existente, los dispositivos de E/S disponible junto con sus identificadores… Para esto, el SO debe tener acceso a datos de configuración del entorno al iniciarse, que se encuentran fuera del SO.

Estructuras de control de proceso

Datos que el SO debe conocer para manejar y controlar procesos

Localización de procesos

Para localizar un proceso, se debe tener en cuenta que debe incluir un programa o conjunto de ellos a ejecuta. Asociados a estos programas existen unas posiciones de memoria para los datos de variables locales, globales y constantes. Por ello, un proceso debe consistir en una entidad suficiente de memoria para almacenar el programa y sus datos.

Adicionalmente, la ejecución de un programa incluye una **pila** que se utiliza para registrar las llamadas a procedimientos y los parámetros pasados entre estos. Finalmente, los procesos poseen un conjunto de atributos utilizados por el SO para controlarlos, el **bloque de control de procesos** (PCB).

Todo el conjunto del programa, datos, pila y atributos se conoce como **imagen del proceso**. Su posición dependerá del esquema de gestión de memoria utilizado, pero en el caso más simple se mantiene como un bloque de memoria contiguo en memoria secundaria, habitualmente en disco. Para que pueda ser gestionado por el SO, al menos una pequeña parte

debe mantenerse en memoria principal, aunque para ejecutarse, se debe cargar en memoria toda la imagen. En cualquier caso, el SO debe conocer la posición en disco o en memoria de cada proceso.

Los SO modernos suponen la existencia de un hardware de paginación que permite el uso de memoria física no contigua, por lo que las tablas que mantiene deben mostrar la localización de cada página de la imagen del proceso.

La estructura de localización se mantiene de la siguiente forma (genérica): hay una tabla primaria de procesos con una entrada por cada proceso. Cada entrada contiene, al menos, un puntero a la imagen del proceso. Si la imagen contiene múltiples bloques, esta información se mantiene directamente en la tabla principal del proceso o está disponible por referencias cruzadas entre las entradas de las tablas de memoria.

Atributos de proceso

Toda la información que el SO requiere para manejar los procesos se encuentra en sus respectivos PCB, aunque cada SO organiza esta información de forma diferente. De forma genérica, se puede agrupar en tres categorías principales: identificación del proceso, información de estado del procesador e información de control del proceso.

En cualquier SO, a cada proceso se le asocia un **identificador** único, que puede ser simplemente un índice de la tabla de procesos principal. Muchas otras tablas contienen información que referencia a determinados procesos usando sus identificadores.

También se le puede asignar a cada proceso un identificador de usuario, que indica qué usuario es responsable del trabajo.

La **información de estado de proceso** indica los contenidos de los registros del procesador. Mientras está ejecutando, se mantiene en los procesos, pero cuando se interrumpe, toda esta información debe salvaguardarse para que se pueda restaurar posteriormente su ejecución.

Habitualmente, los procesadores incluyen un registro o conjunto de ellos, conocidos como **palabras de estado de programa** (*program status Word*, PSW), que contienen información de estado.

La **información de control de proceso** es necesaria para que el SO puede controlar y coordinar varios procesos activos.

En la estructura de las imágenes de procesos en memoria virtual, cada imagen de proceso consiste en un PCB, una pila de usuario, un espacio de direcciones privadas del proceso y cualquier otro espacio de direcciones que comparta con otros procesos.

Un PCB puede contener información estructural que incluye punteros que permiten enlazar PCBs entre sí. Por ello, las listas de procesos que utiliza el SO (Ejecutando, listo, bloqueado) pueden implementarse como listas enlazadas de PCB.

El papel del Bloque de Control de Proceso

El PCB es la estructura de datos más importante del SO, pues cada uno contiene toda la información que necesita el SO de un proceso. Se podría decir que el conjunto de PCBs definen el estado del SO.

Un gran número de rutinas del SO necesitan acceder a la información de los PCBs. El acceso a la misma es fácil gracias a los identificadores, pero su protección puede tener dos problemas:

- Un fallo en una rutina, como un manejador de interrupción, puede dañar un PCB, pudiendo dejar al SO sin poder manejar los procesos afectados.
- Un cambio en la estructura o la semántica de los PCBs puede afectar a un gran número de módulos del SO.

Estos problemas pueden solucionarse obligando a que las rutinas del SO pasen a través de una rutina de manejador, cuyo trabajo es proteger a los PCBs, siendo la única que realiza el arbitraje de las operaciones de lectura y escritura sobre dichos bloques.

Control de procesos

Modos de ejecución

Muchos procesadores proporcionan al menos dos modos de ejecución, permitiendo la ejecución de ciertas instrucciones o el acceso a ciertas regiones de memoria únicamente en modos privilegiados.

El modo menos privilegiado se denomina **modo usuario**, al ser el modo en el que se ejecutan los programas de usuario. El modo privilegiado es el **modo núcleo**, ya que el núcleo del SO engloba las funciones más importantes (gestión de procesos, de memoria, de E/S, funciones de soporte…).

La utilización de dos modos permite proteger al SO y a las tablas clave del sistema. En modo núcleo, el software tiene control completo del procesador y de sus instrucciones, registros y memoria.

El procesador conoce el modo en que está ejecutando con un bit en la palabra de estado del programa (PSW), que se cambia como respuesta a determinados eventos. Cuando un usuario realiza una llamada a un servicio del SO o cuando una interrupción dispara la ejecución de una rutina del SO, el modo de ejecución se cambia al modo núcleo; y cuando termina, se fija en modo usuario.

Creación de procesos

Una vez que el SO decide crear un proceso, procederá de la siguiente manera:

1. **Asignar un identificador de proceso único al proceso.** Añade una nueva entrada a la tabla primaria de procesos.
2. **Reservar espacio para el proceso.** Incluye todos los elementos de la imagen del proceso. Para ello, el SO deberá conocer cuánta memoria se requiere para el espacio de direcciones y para la pila de usuario. Dichos valores pueden asignarse por defecto, dependiendo del tipo de proceso, o pueden fijarse en base a la solicitud de creación del usuario. Si el proceso es creado por otro proceso, el padre puede pasar como parámetros los datos necesarios. Si existe una parte del espacio de direcciones compartido por el nuevo proceso, se fijan los enlaces. Finalmente, se debe reservar el espacio para el PCB.
3. **Inicialización del bloque de control de proceso.** La información de estado de proceso suele inicializar la mayoría de entradas a 0, excepto el contador de programa (fijado en el punto de entrada) y los punteros de pila de sistema (fijados para definir los límites de la pila del proceso). La parte de información de control de procesos se inicializa con los valores por omisión, teniendo en cuenta los solicitados. Inicialmente, un proceso debe tener la prioridad más baja y ningún recurso en posesión, a no ser que se indique explícitamente o los atributos sean heredados del padre.
4. **Establecer los enlaces apropiados.** Por ejemplo, si el SO mantiene cada cola del planificador como una lista enlazada, el nuevo proceso debe situarse en la cola de Listos o de Listos/Suspendidos.
5. **Creación o expansión de otras estructuras de datos.** Por ejemplo, el SO puede mantener un registro de auditoría por cada proceso que se pueda utilizar posteriormente a efectos de facturación y/o análisis de rendimiento del sistema.

Cambio de proceso

Un cambio de proceso tiene lugar cuando se interrumpe la ejecución de un proceso para que el SO asigne a otro proceso el estado Ejecutando.

¿Cuándo se realiza un cambio de proceso?

Un cambio de proceso puede ocurrir en cualquier instante en el que el SO obtiene el control sobre el proceso actualmente en ejecución. Esto puede deberse a interrupciones, traps o llamadas al sistema.

Las **interrupciones** se producen por eventos externos al proceso en ejecución, como la finalización de una operación de E/S. El control se transfiere inicialmente al manejador de interrupción, que realiza tareas internas y salta a una rutina del SO, que se encarga de cada uno de los tipos de interrupciones en particular. Algunos ejemplos son:

- **Interrupción de reloj.** El SO determina si el proceso en ejecución ha excedido o no su ***time-slice*** (unidad máxima de tiempo de ejecución antes de que un proceso sea interrumpido). En este caso, el proceso pasa al estado Listo y se activa otro proceso.
- **Interrupción de E/S.** El SO determina qué acción de E/S ha ocurrido. Si corresponde al evento por el que uno o más procesos estaban esperando, el SO los mueve al estado

de Listos o Listos/Suspendidos. Entonces decide si reanuda la ejecución del proceso en estado Ejecutando o si lo expulsa para ejecutar un proceso de mayor prioridad.

- **Fallo de memoria.** El procesador se encuentra con una referencia a una dirección que no se encuentra en memoria principal. El SO debe traer el bloque desde memoria secundaria y, tras la solicitud de la operación de E/S, pasa al proceso a estado Bloqueado. El SO realiza un cambio de proceso.

Los **traps** están asociados a condiciones de error o excepciones generadas dentro del proceso en ejecución, como un intento de acceso no permitido a un fichero. El SO conoce si dicha condición es irreversible y, si es así, el proceso en ejecución se pone en estado Saliente y se hace un cambio de proceso. De no ser así, el SO intentará un procedimiento de recuperación o notificará al usuario, pudiendo implicar tanto un cambio de proceso como la continuación de la ejecución del proceso actual.

El SO también puede activarse por medio de una **llamada al sistema** procedente del programa en ejecución. Por ejemplo, un programa puede solicitar abrir un archivo, lo que requiere una llamada de E/S que implica un salto a una rutina que es parte del código del SO. Esta llamada puede producir en algunos casos que el proceso pase a estado Bloqueado.

Cambio de modo

Si en el ciclo de una instrucción existe una fase de interrupción, el procesador comprueba que no exista ninguna interrupción pendiente, indicada por la presencia de una señal de interrupción. Si no la hay, pasa a la fase de búsqueda de instrucción; pero si la hay, actúa de la siguiente manera:

1. Coloca el contador de programa en la dirección de comienzo de la rutina del programa manejador de la interrupción.
2. Cambio de modo usuario a modo núcleo de forma que el código de tratamiento de la interrupción pueda incluir instrucciones privilegiadas.

Entonces el procesador pasa a la fase de búsqueda de instrucción, sirviendo a la primera instrucción del programa de manejo de la interrupción. En este punto, el contexto del proceso interrumpido se guarda en su PCB (contador de programa, registros del procesador, información de la pila…).

El manejador de la interrupción realiza unas pocas tareas básicas relativas a la interrupción, como borrar el *flag* que señala la presencia de interrupciones, enviar una confirmación a la entidad que lanzó la interrupción, o realizar tareas internas relativas al evento que causó la interrupción.

En muchos Sistemas Operativos, la existencia de una interrupción no implica necesariamente un cambio de proceso. Las operaciones de salvaguarda y recuperación del PCB se realizan por hardware.

Cambio del estado del proceso

Un cambio de modo puede ocurrir sin que se cambie el estado del proceso en ejecución. Si se realizara dicho cambio, el SO actuaría de la siguiente manera:

1. Salva el estado del procesador, incluyendo el contador de programa y los registros.
2. Actualiza el PCB del programa en ejecución, cambiando al nuevo estado y actualizando campos como la razón por la cuál el proceso ha dejado de ejecutar.
3. Mover el PCB a la cola apropiada, en función de su nuevo estado.
4. Selección de un nuevo proceso a ejecutar.
5. Actualizar el PCB del proceso elegido, incluyendo su estado a Ejecutando.
6. Actualizar las estructuras de datos de gestión de memoria, si fuera necesario.
7. Restaurar el estado del procesador al que tenía en el momento en el que el proceso seleccionado salió del estado Ejecutando, leyendo los valores del contador de programa y los registros.

Con todos estos pasos, un cambio de proceso requiere mayor esfuerzo que un cambio de modo.

Ejecución del Sistema Operativo

Si el SO es un conjunto de programas que se ejecutan por el procesador como cualquier otro, ¿es el SO un proceso del sistema? ¿Cómo se controla? Hay diferentes visiones:

Núcleo sin procesos

Es una visión tradicional en la que cuando el proceso en ejecución se interrumpe o invoca una llamada al sistema, el contexto se guarda y el control pasa al núcleo. El SO tiene su propia región de memoria y su propia pila de sistema para controlar la llamada a procedimientos y sus retornos. El SO puede realizar todas las funciones que necesite y restaurar el contexto del proceso interrumpido.

De forma alternativa, el SO puede realizar la salvaguarda del contexto y la activación de otro proceso diferente, dependiendo de la causa de la interrupción y las circunstancias del momento.

En cualquier caso, el concepto de proceso se aplica únicamente a los programas de usuario. El código del SO se ejecuta como una entidad independiente que requiere un modo privilegiado de ejecución.

Ejecución dentro de los procesos de usuario

Esta visión es común en máquinas pequeñas, y consiste en ejecutar virtualmente todo el SO en el contexto de un proceso de usuario. El SO se ve como un conjunto de rutinas que el usuario invoca dentro del proceso.

De esta forma, el SO maneja *n* imágenes de procesos, cada una de las cuales incluye, además de las regiones normales; áreas de programa, datos y pila para los programas del núcleo. Se utiliza una pila de núcleo separada para manejar llamadas/retornos cuando el proceso entra en modo núcleo. El código del SO y sus datos están en un espacio de direcciones compartidas entre todos los procesos.

Cuando ocurre una interrupción, *trap* o llamada al sistema, el procesador se pone en modo núcleo y el control pasa al SO. El contexto se salva y se cambia de modo a una rutina del SO, aunque la ejecución continúa dentro del proceso actual. No hay cambio de proceso, sino un cambio de modo dentro del mismo proceso.

Si el SO, tras realizar su trabajo, determina que el proceso actual debe continuar su ejecución, el cambio de modo continúa con el programa interrumpido. La ventaja es que se evita un doble cambio de proceso.

Sin embargo, si se determina que se debe realizar un cambio de proceso en lugar de continuar con el proceso anterior, el control pasa a la rutina de cambio de proceso, que puede ejecutarse o no dentro del proceso actual. En algún momento, el proceso actual se debe poner en un estado de no ejecución, designando a otro proceso como el siguiente a ejecutar. Durante esta fase, es más conveniente ver la ejecución como fuera de cualquiera de los procesos.

La razón por la que este esquema no se convierte en una situación arbitraria y caótica cuando un proceso elige a otro para su ejecución, es que, durante los instantes críticos, el código que está ejecutando es compartido de SO, no código usuario. Dentro de un proceso, pueden ejecutarse tanto programas de usuario como programas del SO. Los programas del SO que se ejecutan en varios procesos son idénticos.

Sistemas Operativos basados en procesos

Consiste en implementar el SO como una colección de procesos del sistema, pero que ejecutan en modo núcleo. Las principales funciones del núcleo se organizan como procesos independientes, por lo que debe haber una pequeña cantidad de código para intercambio de procesos que se ejecuta fuera de ellos.

Esta visión tiene ventajas. Impone una disciplina de diseño que implica modularidad e interfaces claras entre módulos. Además, las funciones del SO que no sean criticas están convenientemente separadas como otros procesos.

La implementación en entornos de multiprocesadores y multicomputadores, en los cuales determinados servicios del SO se pueden enviar a procesadores dedicados, incrementa el rendimiento.

Procesos e hilos

Multihilo

Multihilo se refiere a la capacidad de un SO de dar soporte a múltiples hilos de ejecución en un solo proceso.

En un entorno multihilo, un proceso se define como la unidad de asignación de recursos y una unidad de protección. Además, con un proceso se asocian:

- Un espacio de direcciones virtuales que soporta la imagen del proceso.
- Acceso protegido a procesadores, otros procesos, archivos, recursos E/S…

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

- Un bloque de control de hilo.
- Un estado de ejecución.
- Un contexto de hilo que se almacena cuando no está en ejecución.
- Una pila de ejecución.
- Espacio de almacenamiento para variables locales.
- Acceso a la memoria y los recursos de su proceso, que es compartido por todos los hilos del mismo.

Si un hilo modifica una variable, el resto ven dicho cambio. Si un hilo abre un archivo con permisos de lectura, los demás también pueden leerlo.

Los principales beneficios de los hilos provienen del rendimiento:

- Lleva mucho menos tiempo crear un hilo en un proceso existente que crear un nuevo proceso.
- Lleva menos tiempo finalizar un hilo que un proceso.
- Lleva menos tiempo cambiar entre dos hilos dentro del mismo proceso.
- Los hilos se pueden comunicar entre ellos sin necesidad de invocar al núcleo.

Ejemplos de uso de hilos en un sistema de multiprocesamieno de un solo usuario:

- Trabajo en primer y en segundo plano. En un Excel, un hilo podría mostrar menús y otro ejecutar los mandatos del usuario actualizando la hoja de cálculo.
- Procesamiento asíncrono. Los elementos asíncronos de un programa pueden implementarse con hilos, como uno que escriba en un *buffer* de la RAM una copia de seguridad de lo último escrito.
- Velocidad de ejecución. Un hilo puede estar computando datos mientras otro lee los siguientes.
- Estructura modular de programas. Cada función puede realizarse concurrentemente por distintos hilos.

En un SO que soporte hilos, la planificación y activación se realizan a nivel de hilo, por lo que la mayor parte de la información relativa a la información se mantiene en estructuras de datos a nivel de hilo. Sin embargo, también existen acciones que afectan a todos los hilos de un proceso, como suspender o finalizar un proceso, que el SO debe saber gestionar.

Funcionalidades de los hilos

Los hilos, al igual que los procesos, tienen estados de ejecución y pueden sincronizarse entre ellos.

Estados de los hilos

Los principales estados son: Ejecutando, Listo y Bloqueado. No tiene sentido aplicar estados de suspensión a los hilos, ya que eso se realiza a nivel de proceso y, si se expulsa un proceso, se expulsan todos sus hilos, pues comparten espacio de direcciones.

Hay cuatro operaciones básicas relacionadas con el cambio de estado del hilo:

- **Creación.** Cuando se crea un nuevo proceso, se crea un primer hilo para el mismo. Un hilo puede crear entonces otro hilo, proporcionando un puntero a sus instrucciones y argumentos. 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 por un evento se bloquea, almacenando los registros de usuario, contador de programa y punteros de pila. El procesador puede pasar a ejecutar otro hilo en estado Listo, dentro del mismo proceso o en otro diferente.
- **Desbloqueo.** Cuando sucede el evento por el que hilo está bloqueado, se pasa a la cola de Listos.
- **Finalización.** Cuando se completa un hilo, se liberan su registro de contexto y pilas.

Si el bloqueo de un hilo implica el bloqueo del proceso completo, se pierde la potencia y flexibilidad de los hilos.

En un uniprocesador, la multiprogramación permite el intercalado de múltiples hilos con múltiples procesos. La ejecución pasa de un hilo a otro cuando se bloquea el hilo en ejecución o su porción de tiempo se agota.

Sincronización de hilos

Todos los hilos de un proceso comparten el mismo espacio de direcciones y otros recursos. Cualquier alteración en dichos recursos por uno de los hilos afecta al resto del mismo proceso, por lo que es necesario sincronizar las actividades de los hilos para que no interfieran entre ellos.

Enfoques multihilo

Hilos de nivel de usuario

En un entorno ULT, la aplicación gestiona todo el trabajo de los hilos y el núcleo no es consciente de su existencia. Cualquier aplicación puede programarse para ser multihilo a través del uso de una biblioteca de hilos.

Por defecto, una aplicación comienza con un solo hilo y ejecutando ese hilo. Esta aplicación y su hilo se localizan en un solo proceso gestionado por el núcleo. En cualquier instante de ejecución, la aplicación pude crear un nuevo hilo llamando a la utilidad de creación de la biblioteca de hilos. Esta crea una estructura de datos para el nuevo hilo y pasa el control a uno de los hilos del proceso en estado Listo.

Toda esa actividad tiene lugar en el espacio de usuario y dentro de un solo proceso, sin que el núcleo sea consciente. El núcleo continua planificando el proceso como si fuera una unidad con un único estado.

El uso de ULT en lugar de KTL presenta las siguientes ventajas:

- Un cambio de hilo no requiere privilegios de modo núcleo, ahorrando la sobrecarga de dos cambios de modo.
- La planificación puede especificarse por parte de la aplicación, pudiendo tener cada una el esquema que más le favorezca.
- Los ULT pueden ejecutar en cualquier SO, pues no se necesitan cambios en el núcleo para dar soporte a los ULT. La biblioteca de hilos es compartida por todas las aplicaciones.

También tiene dos desventajas en comparación con los KLT:

- En un SO, muchas llamadas al sistema son bloqueantes, por lo que cuando un hilo realiza una, se bloquean todos los hilos del proceso.
- Una aplicación multihilo no puede sacar ventaja del multiproceso, ya que el núcleo asigna el proceso a un solo procesador al mismo tiempo.

Estas desventajas pueden salvarse escribiendo una aplicación de múltiples procesos en lugar de múltiples hilos, aunque cada cambio es un cambio de proceso en lugar de hilo, generando sobrecarga.

Otra opción es el ***jacketing*** (revestimiento), que intenta convertir una llamada al sistema en 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, verificando si el dispositivo de E/S está ocupado. Si lo está, el hilo entra en estado Bloqueado y pasa el control a otro hilo. Cuando el hilo recupera el control, chequea de nuevo el dispositivo.

Hilos a nivel de núcleo

En un entorno KLT, el núcleo gestiona todo el trabajo de gestión de hilos, sin haber código de gestión de hilos en la aplicación. Windows es un ejemplo de este enfoque.

Cualquier aplicación puede programarse para ser multihilo, y todos los hilos de una aplicación de mantienen en un solo proceso. El núcleo mantiene información de contexto del proceso como una entidad y de los hilos individuales del proceso. La planificación se realiza a nivel de hilo.

Este enfoque resuelve los inconvenientes de ULT, pues el procesador puede planificar simultáneamente múltiples hilos en múltiples procesadores; y si se bloquea un hilo de un proceso, el núcleo puede planificar otro hilo.

La principal desventaja del enfoque KLT, en comparación con ULT, es que la transferencia de control de un hilo a otro del mismo proceso requiere un cambio de modo al núcleo.

En la práctica, ninguno de los enfoques es 100% mejor que el otro, depende de la naturaleza de cada aplicación el beneficio o no de la respectiva ganancia. Si la mayor parte de los cambios de hilo en una aplicación requieren acceso al modo núcleo, por ejemplo, el esquema basado en ULT no sería tan superior al basado en KTL.

Enfoques combinados

Algunos SO, como Solaris, proporcionan utilidades combinadas ULT/KLT. En un sistema combinado, la creación de hilos se realiza por completo en el espacio de usuario, como la mayor parte de la planificación y la sincronización de hilos dentro de una aplicación. Los múltiples ULT de una aplicación se asocian en un número (menor o igual) de KLT.

En los enfoques combinados, múltiples hilos de la misma aplicación pueden ejecutar en paralelo en múltiples procesadores, y una llamada al sistema bloqueante no necesita bloquear el proceso completo.

Planificación de procesos

Tipos de planificación del procesador

El objetivo de la planificación de procesos es asignar procesos a ser ejecutados por el procesador o procesadores a lo largo del tiempo, cumpliendo los objetivos del sistema como tiempo de respuesta, rendimiento o eficiencia del procesador.

La planificación afecta al rendimiento porque determina qué proceso espera y que proceso progresa. Es un problema de manejo de colas para minimizar el retardo en la cola y optimizar el rendimiento general.

Planificación a largo plazo

El planificador a largo plazo determina qué programas se admiten en el sistema para su procesamiento, controlando el grado de multiprogramación. Una vez admitido, el programa se convierte en proceso, añadiéndose a la cola de Listos (gestionada por el planificador a corto plazo) o a la de Listos/Suspendidos (gestionada por el planificador a medio plazo).

En un sistema por lotes, los nuevos trabajos enviados se mandan al disco y se mantienen en una cola de lotes. El planificador a largo plazo creará procesos desde la cola cuando pueda, decidiendo cuándo el SO puede coger nuevos procesos y qué trabajos son los aceptados.

La primera decisión se toma en función del grado de multiprogramación deseado. A mayor número de procesos, menor será el porcentaje de tiempo en que cada proceso pueda ejecutar. El planificador a largo plazo puede limitar el grado de multiprogramación para proporcionar un servicio satisfactorio al actual conjunto de procesos.

La segunda decisión puede tomarse con un sencillo FIFO o puede ser una herramienta para gestionar el rendimiento del sistema, incluyendo prioridades, tiempos estimados de ejecución o requisitos de E/S.

Para los programas interactivos en un sistema de tiempo compartido, la petición de la creación de un proceso puede estar generada por un usuario. En este caso, el SO aceptará todos los usuarios autorizados hasta que se sature. Esta saturación se estima utilizando ciertas medidas prefijadas, y puede resultar en mensajes que indican que el sistema está completo.

Planificación a medio plazo

La planificación a medio plazo es parte de la función de intercambio. Esta decisión se basa en la necesidad de gestionar el grado de multiprogramación. En un sistema que no utiliza memoria virtual, la gestión de la memoria es otro aspecto a tener en cuenta.

Planificación a corto plazo

El planificador a corto plazo ejecuta con relativamente poca frecuencia y toma decisiones de grano grueso de admitir o no procesos. El planificador a medio plazo se ejecutará más frecuentemente para tomar decisiones de intercambio. El planificador a corto plazo, conocido también como **activador**, ejecuta mucho más frecuentemente y toma decisiones de grano fino sobre qué proceso ejecutar el siguiente.

El planificador a corto plazo se invoca siempre que ocurre un evento que pueda significar el bloqueo del proceso actual, como interrupciones de reloj, de E/S, llamadas al sistema o señales como semáforos.

Algoritmos de planificación

Criterios de planificación a corto plazo

El objetivo principal de la planificación a corto plazo es asignar el tiempo de procesador de tal forma que se optimicen aspectos del comportamiento del sistema. Estos criterios se pueden clasificar en dos dimensiones:

- Criterios orientados al usuario y orientados al sistema, relacionados con el comportamiento del sistema tal y como lo percibe un usuario o proceso, como el tiempo de respuesta.
- Criterios orientados al sistema, relacionados con el uso efectivo y eficiente el procesador, como el rendimiento.

Los primeros son importantes en casi todos los sistemas, mientras que los segundos son menos importantes en un sistema monousuario, donde no es tan importante el rendimiento como la respuesta del sistema a las aplicaciones del usuario.

También se pueden clasificar los criterios dependiendo de si están o no relacionados con las prestaciones. Los que sí lo están son cuantitativos y pueden ser medidos, como el tiempo de respuesta o el rendimiento; mientras que los que no lo están o son cuantitativos por naturaleza o no pueden ser medidos y analizados, como la previsibilidad.

Principales criterios de planificación:

- Orientados al usuario, relacionados con las prestaciones:
  - Tiempo de estancia: tiempo transcurrido desde que se lanza un proceso hasta que se finaliza. Tiempo de ejecución más tiempo de espera.
  - Tiempo de respuesta: tiempo desde que se lanza una petición hasta que se comienza a recibir la respuesta.
  - Fecha tope: el procesador debe maximizar el porcentaje de fechas tope conseguidas subordinando procesos.
- Orientados al usuario, otros:
  - Previsibilidad: un proceso dado debería ejecutarse aproximadamente en el mismo tiempo y con el mismo coste, independientemente de la carga del sistema.
- Orientados al sistema, relacionados con las prestaciones:
  - Rendimiento: número de procesos completados por unidad de tiempo.
  - Utilización del procesador: porcentaje de tiempo que el procesador esta ocupado.
- Orientados al sistema, otros:
  - Equidad: en ausencia de orientación a los usuarios, los procesos deben ser tratados de la misma manera, evitando inanición.
  - Imposición de prioridades: el planificador debe favorecer a procesos con prioridades más altas.
  - Equilibrado de recursos: los procesos que utilicen poco los recursos que en un determinado momento están sobreutilizados deberían ser favorecidos.

Conceptos relativos a los algoritmos de planificación

La **función de selección** determina qué proceso, entre los procesos listos, se selecciona para su ejecución. Puede estar basada en prioridades, requisitos sobre los recursos o las características de ejecución del proceso, dividiéndose estas últimas en:

- w = tiempo usado en el sistema hasta el momento, esperando o ejecutando.
- e = tiempo usado en ejecución hasta el momento.
- s = tiempo total de servicio requerido por el proceso, incluyendo e.

El **modo de decisión** especifica los instantes de tiempo en que se ejecuta la función de selección, habiendo dos categorías generales:

- Sin expulsión (no apropiativas). Una vez que un proceso está en estado Ejecutando, continua hasta que termina o se bloquea.
- Con expulsión (apropiativa). Un proceso ejecutando puede ser interrumpido y pasado al estado Listo por el SO cuando llega un nuevo proceso, cuando llega una interrupción que pasa un proceso Bloqueado a Listo, o periódicamente.

Las políticas expulsivas tienen mayor sobrecarga que las no expulsivas, pero pueden proporcionar mejor servicio a la población total de procesos, ya que evitan que cualquier proceso monopolice el procesador durante mucho tiempo. Además, el coste de expulsión puede ser relativamente bajo.

First-come-first-served (FCFS)

Es la directiva de planificación más sencilla. Cuando un proceso está Listo, se une a la cola de Listos. Cuando el proceso en ejecución termina, se selecciona para ejecutar el proceso que ha estado más tiempo en la cola de Listos.

FCFS funciona mucho mejor para procesos largos. Un proceso corto que llegue justo después de uno largo tendrá que esperar a que el largo termine de ejecutar.

Otro problema de FCFS es que tiende a favorecer procesos limitados por el procesador sobre los limitados por la E/S. Esto provoca que los dispositivos de E/S puedan estar ociosos, aunque exista trabajo potencial que puedan hacer.

FCFS no es una alternativa atractiva para un sistema uniprocesador, pero a menudo se combina con esquemas de prioridades para ser más eficaz. El planificador puede mantener varias colas, una por cada nivel de prioridad, y despachar dentro de cada cola con la estrategia FCFS.

Round robin (RR)

El turno rotatorio permite reducir el castigo a los procesos cortos utilizando las interrupciones de reloj que se producen cíclicamente. Cuando sucede una interrupción, el

proceso en ejecución pasa a la cola de Listos, y se selecciona el siguiente proceso utilizando la política FCFS.

La clave de este diseño es la longitud del *quantum* de tiempo: si es muy pequeño, los procesos se intercambiarán rápidamente con la consiguiente sobrecarga del manejo de la interrupción de reloj, la planificación y la activación. Se deben evitar los *quantums* pequeños.

Una idea es que el *quantum* sea ligeramente mayor que el tiempo requerido para una interacción o una función típica del proceso. Si es menor, la mayoría de los procesos necesitarán, al menos, dos *quantums*.

Por otro lado, un *quantum* demasiado grande degeneraría en FCFS.

La planificación *round robin* es particularmente efectiva en sistemas de tiempo compartido de propósito general o sistemas de procesamiento transaccional.

La principal desventaja es que esta planificación trata de forma desigual a los procesos limitados por el procesador y a los procesos limitados por la E/S. Generalmente, los primeros tienen ráfagas de procesador más cortas, empeorando el rendimiento de los segundos.

Esto se solucionaría con un refinamiento conocido como *virtual round robin* (VRR), que incluye una cola auxiliar FCFS a la que se mueven los procesos después de estar bloqueados en una E/S. En la activación, estos procesos tienen preferencia sobre los de la cola de Listos.

Shortest process next (SPN)

La estrategia “primero el más corto” trata de reducir la desventaja de los procesos cortos. Es una política no expulsiva en la que selecciona el proceso con el tiempo de procesamiento más corto que esté listo.

El rendimiento global mejora significativamente, pues se incrementa el número de procesos cortos servidos incrementando la variabilidad de los tiempos de los procesos más largos, reduciendo la predecibilidad. El problema de SPN es la necesidad de saber (o estimar) el tiempo de procesamiento requerido por cada proceso.

Un riesgo con SPN es la posibilidad de inanición para los procesos más largos, si hay una llegada constante de procesos más cortos. Tampoco es una política deseada para muchos sistemas por la carencia de expulsión, pues, como en FCFS, un proceso corto que llegue cuando uno muy largo estuviera ejecutando tendría que esperar a que terminase.

Shortest remaining time (SRT)

La política “menor tiempo restante” es una versión expulsiva de SPN que selecciona siempre el proceso que tiene el menor tiempo de proceso restante. Si un nuevo proceso se une a la cola de listos y tiene menor tiempo restante que el proceso en ejecución, el planificador podría expulsar al proceso actual.

Al igual que en SPN, se debe realizar una estimación del tiempo de proceso para la función de selección, y existe riesgo de inanición para los procesos largos.

SRT no tiene el sesgo a favor de los procesos largos de FCFS, ni la sobrecarga de interrupciones de *round robin*. Sí tiene una sobrecarga porque necesita almacenar los tiempos de servicio transcurridos.

Highest response ratio next (HRRN)

La política "primero el de mayor tasa de respuesta" trata de minimizar dicha tasa, que se puede aproximar de la siguiente manera:

$$R = \frac{(w + s)}{s}$$

donde R es la tasa de respuesta, w el tiempo invertido esperando por el procesador y s el tiempo de servicio esperado.

El valor mínimo de R es 1, que sucede cuanto un proceso acaba de entrar en el sistema. Si se despacha inmediatamente al proceso, R es igual al tiempo de estancia normalizado.

La política utiliza como función de selección el proceso listo con mayor valor de R. Se favorece a los procesos cortos, pero el envejecimiento incrementa la tasa, por lo que en un determinado momento un proceso largo podría competir con los más cortos.

Feedback

Si no es posible averiguar el tiempo de servicio de varios procesos, SPN, SRT y HRRN no se pueden utilizar. Otra forma de favorecer a los trabajos cortos es penalizar a los trabajos que hayan estado ejecutando más tiempo: se trata de basarse en el tiempo de ejecución utilizado hasta el momento.

Se trata de una planificación con expulsión que utiliza un mecanismo de prioridades dinámico. Los procesos que se crean se sitúan en una primera cola, y cada vez que son expulsados cambian a una cola de menor prioridad. Los procesos cortos se ejecutarán rápido, sin migrar muy lejos en la jerarquía de colas. Cada cola se ejecuta mediante FCFS, menos la última, que utiliza RR. Esto se conoce como **retroalimentación multinivel**.

Una versión más sencilla consiste en realizar la expulsión con intervalos periódicos, aunque el tiempo de estancia de los procesos más largos puede alargarse de forma alarmante, pudiendo llegar a la inanición. Para paliarlo, cada cola de una prioridad menor tiene un tiempo de expulsión mayor.

Planificación en sistemas multiprocesadores

Aspectos de diseño

En un multiprocesador, la planificación involucra tres aspectos interrelacionados:

Asignación de procesos a procesadores

En una arquitectura multiprocesador uniforme, el enfoque más simple consiste en tratar cada proceso como un recurso colectivo y asignar procesos a procesadores por demanda.

Si un proceso se vincula permanentemente a un procesador, se mantiene una cola a corto plazo por procesador, reduciendo la sobrecarga de la función de planificación y permitiendo la planificación de grupo o pandilla.

La desventaja de la asignación dinámica es que un procesador puede estar ocioso mientras otro tiene trabajo acumulado. Para evitarlo, puede usarse una cola común. De esta forma, un proceso puede ser ejecutado en diferentes procesadores, siendo especialmente beneficioso en una arquitectura de memoria compartida, en la que el contexto de todos los procesos está disponible para todos los procesadores. Otra opción sería el balanceo dinámico de carga, como utiliza Linux.

Independientemente de cómo se vinculan los procesos, hay dos enfoques para asignar procesos a procesadores: maestro/esclavo y camaradas.

En la arquitectura maestro/esclavo, ciertas funciones del núcleo se ejecutan siempre en un procesador concreto y el resto ejecutan programas de usuario. El maestro es el responsable de la planificación de trabajos.

En la arquitectura de camaradas, el núcleo puede ser ejecutado en cualquier procesador, y cada procesador se auto-planifica con los procesos que dispone. El SO debe asegurar que dos procesadores no escogen el mismo proceso ni se pierden procesos.

Uso de la multiprogramación en procesadores individuales

En los multiprocesadores tradicionales, cada procesador debe ser capaz de cambiar entre varios procesos para conseguir una alta utilización y un mejor rendimiento.

Para aplicaciones de grano medio ejecutando en un multiprocesador con muchos procesadores, conseguir que cada uno esté ocupado el mayor tiempo posible no es tan importante como proporcionar el mejor rendimiento medio de las aplicaciones. Una aplicación con varios hilos necesita que todos estén dispuestos a ejecutar simultáneamente.

Activación de procesos

En un monoprocesador multiprogramado, el uso de prioridades u otros algoritmos pueden mejorar a la estrategia FCFS. Con multiprocesadores, un enfoque más simple puede ser más eficaz con menos sobrecarga.

Planificación de procesos

En los esquemas tradicionales, los procesos se situaban en una única cola compartida por los procesadores o en colas con prioridades.

Planificación de hilos

En un monoprocesador, los hilos pueden usarse par asolapar E/S con procesamiento, beneficiándose del bajo coste del cambio entre hilos.

En un sistemas multiprocesador, su potencial es aún mayor, pues los hilos pueden suponer paralelismo real dentro de una aplicación ejecutándose en distintos procesadores.

Hay cuatro enfoques generales para la planificación multiprocesador de hilos: compartición de carga, planificación en pandilla, asignación de procesador dedicado y planificación dinámica.

Compartición de carga

Se trata de la técnica más simple, aunque una de las más utilizadas en los multiprocesadores actuales. Existe una cola global de hilos listos de la que cada procesador ocioso selecciona un hilo a ejecutar. Ventajas:

- La carga se distribuye uniformemente entre los procesadores, evitando que haya procesadores ociosos y trabajo pendiente.
- No se precisa un planificador centralizado, sino que la rutina de planificación del SO se ejecuta en cada procesador cuando queda disponible.
- Es compatible con todos los esquemas monoprocesador, por ejemplo:
    - FCFS: cuando se crea un trabajo, sus hilos se colocan al final de la cola global, de la que cada procesador va cogiendo hilos y los ejecuta hasta que terminan o se bloquean. Es la política superior en la compartición de carga.
    - Menor número de hilos primero: la cola global se organiza como una cola de prioridad, asignando la mayor prioridad a los hilos de los procesos con menos hilos. El hilo planificado se ejecuta al igual que en FCFS.
    - Menor número de hilos primero con expulsión: igual que la anterior, pero si llega un trabajo con menor número de hilos que el trabajo en ejecución, se expulsan los hilos del trabajo planificado.

Desventajas:

- La cola global debe cumplir la exclusión mutua, pudiendo convertirse en un cuello de botella a mayor número de procesadores.
- La *caché* local de los procesadores es menos eficaz puesto que si un hilo es expulsado, es poco probable que vuelva a ejecutar en el mismo procesador.
- Si todos los hilos se tratan como un conjunto común de hilos, es poco probable que todos los hilos de un programa se ejecuten a la vez. Esto es un problema si se necesita un alto grado de coordinación entre hilos de un programa, aumentando los cambios de proceso.

El SO Mach utiliza una técnica mejorada de compartición de carga en la que existe una cola global y una local por cada procesador. En la planificación, cada procesador examina primero su cola local, en la que estarían los hilos que ya han sido asignados antes a él.

Planificación en pandilla

Se trata de planificar un conjunto de hilos relacionados para ejecutar sobre un conjunto de procesadores al mismo tiempo, en una relación de uno-a-uno. La planificación en grupo tiene dos beneficios:

- Si se ejecutan en paralelo procesos relacionados, puede reducirse el bloqueo por sincronización, pueden necesitarse menos cambios de proceso y las prestaciones aumentan.
- La sobrecarga de planificación puede reducirse dado que una decisión puede afectar a varios procesadores y procesos a la vez.

La **coplanificación** se basa en planificar conjuntos de tareas relacionadas, llamados cargas de trabajo. Sus elementos individuales tienden a ser pequeños, próximos a la idea de hilo.

La **planificación en pandilla** se aplica simultáneamente a los hilos que componen un proceso, siendo buena para aplicaciones paralelas de grano medio o fino cuyo rendimiento se ve degradado cuando distintas partes de ellas tienen que esperar a otras para ejecutar.

La planificación en pandilla minimiza los cambios de proceso y reduce el tiempo de ubicación de recursos. Por ejemplo, múltiples hilos planificados en pandilla pueden acceder a un fichero sin la sobrecarga de bloquearse durante una operación de posicionamiento y lectura/escritura.

Sin embargo, crea un requisito para la ubicación del procesador, que sufre diferencias si se trabaja con aplicaciones con distintos números de hilos. Las aplicaciones con menos hilos producen un rendimiento menor del sistema, que puede corregirse asignando varias de esas aplicaciones a un mismo procesador o dando más tiempo de procesador a las aplicaciones con más hilos.

Asignación de procesador dedicado

Es una forma extrema de planificación en pandilla en la que un grupo de procesadores se dedica a una aplicación durante toda su ejecución. Es el opuesto a la compartición de carga.

Puede parecer un desperdicio de procesador, ya que si un hilo se bloquea, el procesador de ese hilo se queda ocioso. Sin embargo:

- En un sistema con muchos procesadores, la utilización del procesador deja de ser tan importante como medida de eficiencia y rendimiento.
- Evitar el cambio de proceso durante la vida de un programa puede mejorar su velocidad.

Una estrategia eficaz para esta técnica es limitar el número de hilos activos al número de procesadores del sistema.

Planificación dinámica

Para algunas aplicaciones es posible conseguir que el número de hilos del proceso pueda ser alterado dinámicamente, permitiendo al SO ajustar la carga para mejorar la utilización.

El SO particiona los procesadores entre los trabajos, y cada trabajo ejecuta sus hilos en los procesadores de su partición. Las mismas aplicaciones deben seleccionar el conjunto a ejecutar y qué hilos suspender cuando el proceso sea expulsado.

Por tanto, la responsabilidad de planificación del SO está limitada a la ubicación del procesador, siguiendo la siguiente política: cuando un trabajo solicita uno o más procesadores:

1. Si hay procesadores ociosos, los utiliza para satisfacer la solicitud.
2. En otro caso, y si el trabajo es nuevo, se ubica en un único procesador que se le quita a otro trabajo que tenga más de uno.
3. Si no puede satisfacerse cualquier parte de la solicitud, se queda pendiente hasta que un procesador esté disponible o no necesite procesadores extra.
4. Cuando se libera uno o más procesadores, se examina la cola de solicitudes no satisfechas. Primero se asignan procesadores a los trabajos que no tengan ninguno, y después, se sigue una estrategia FCFS.

Planificación en sistemas de tiempo real

Los distintos enfoques de planificación dependen de cuándo el sistema realiza análisis de planificación; y si lo hace, de si se realiza estática o dinámicamente; y de si el resultado del análisis produce un plan de planificación.

En base a esto, se identifican cuatro clases de algoritmos:

- **Enfoques estáticos dirigidos por tabla.** Se realiza un análisis estático de la factibilidad de la planificación que determina en qué instante de ejecución debe comenzar a ejecutarse cada tarea.
  Es aplicable a tareas periódicas y no es flexible, ya que cualquier cambio en los requisitos de las tareas requiere rehacer toda la planificación.
- **Enfoques estáticos expulsivos dirigidos por prioridad.** Se realiza un análisis estático que asigna prioridades a las tareas y utiliza un planificador expulsivo basado en prioridades.
  La asignación de prioridades se realiza en función de las restricciones de tiempo asociadas a cada tarea.
- **Enfoques dinámicos basados en un plan.** La factibilidad se determina en tiempo de ejecución, determinando en qué instante de ejecución debe ponerse en marcha cada tarea. Una nueva tarea será aceptada como ejecutable sólo si se pueden satisfacer sus restricciones de tiempo y no interfiere en los plazos de las tareas ya planificadas.
- **Enfoques dinámicos de mejor esfuerzo.** Es uno de los más utilizados hoy en día. No se realiza análisis de factibilidad. El sistema intenta cumplir todos los plazos y aborta la ejecución de procesos cuyo plazo haya fallado.
  A cada tarea se le asigna una prioridad basada en sus restricciones, utilizando planificaciones como la del plazo más cercano. Sin embargo, no sabremos si una determinada restricción será satisfecha hasta que venza su plazo o la tarea se complete. Esta es su principal desventaja, frente a su facilidad de implementación.

El problema de la inversión de prioridad

Es un fenómeno que puede suceder en cualquier esquema de planificación expulsivo basado en prioridades, pero es especialmente relevante en sistemas de tiempo real. En estos esquemas, el sistema debe ejecutar siempre la tarea de mayor prioridad.

El problema ocurre cuando las circunstancias del sistema fuerzan a una tarea de mayor prioridad a esperar por una de menor prioridad, como por ejemplo cuando la de mayor prioridad requiere un recurso que utiliza la de menor prioridad, quedando en estado bloqueado y haciendo posible que se violen las restricciones de tiempo, a menos que la tarea que tiene el recurso termine pronto.

Esto puede agravarse en la **inversión de prioridad ilimitada**, en la que depende no sólo del tiempo necesario para conseguir el recurso sino también de acciones impredecibles de otras tareas no relacionadas. Hay dos enfoques para evitarla:

- La **herencia de prioridad** hace que una tarea de menor prioridad herede la prioridad de cualquier tarea de mayor prioridad pendiente de un recurso que compartan. El cambio se produce cuando se bloquea la de mayor prioridad en el recurso; y finaliza cuando la tarea de menor prioridad lo libera.
- En el enfoque de **techo de prioridad**, se asigna una prioridad con cada recurso, que es un nivel más alta que la prioridad de su usuario más prioritario. El planificador asigna esta prioridad a cualquier tarea que acceda al recurso, hasta que termine su utilización.

Gestión de procesos e hilos en Linux

Tareas Linux

En Linux, un proceso o tarea se representa por una estructura de datos ***task_struct***, que contiene la siguiente información:

- **Estado.** Estado de ejecución (ejecutando, listo, suspendido, detenido, *zombie*).
- **Información de planificación.** Linux la necesita para planificar procesos, ya que pueden ser normales o de tiempo real y tener una prioridad. Los procesos de tiempo real se planifican antes que los normales; y dentro de cada categoría, hay prioridades relativas.
- **Identificadores.** Cada proceso tiene un identificador único de proceso e identificadores de usuario y grupo, utilizados para asignar privilegios de acceso.
- **Comunicación entre procesos.** Linux soporta el mecanismo IPC (*Interprocess Communication*).
- **Enlaces.** A sus padres, hermanos e hijos.
- **Tiempos y temporizadores.** Incluye el tiempo de creación del proceso y el tiempo de procesador consumido hasta el momento. Un proceso puede tener asociados varios temporizadores a través de llamadas al sistema.
- **Sistema de archivos.** Incluye punteros a cualquier archivo abierto por el proceso y a los directorios actual y raíz del proceso.
- **Espacio de direcciones.** Define el espacio de direcciones virtual asignado.
- **Contexto específico del procesador.** Información de los registros y de la pila.
- **Ejecutando.** Se corresponde con los estados de Ejecutando o Listo para ejecutar.
- **Interrumpible.** Es un estado bloqueado, en el que el proceso está esperando por un evento.
- **Ininterrumpible.** Otro estado de bloqueado, en el que el proceso espera sobre un estado del hardware, por lo que no manejará señales.
- **Detenido.** El proceso ha sido parado y sólo puede ser reanudado por la acción positiva de otro proceso.
- ***Zombie*.** Un proceso terminado que, por alguna razón, todavía debe tener su estructura de tarea en la tabla de procesos.

Hilos Linux

Las versiones antiguas del núcleo de Linux no ofrecían soporte multihilo, por lo que las aplicaciones debían escribirse como un conjunto de funciones de biblioteca a nivel de usuario. La más popular es la *biblioteca pthread* (*POSIX thread*), en donde se asociaban todos los hilos en un único proceso a nivel de núcleo.

Las versiones modernas de UNIX ofrecen hilos a nivel de núcleo. Linux no diferencia entre procesos e hilos. Los hilos de nivel de usuario se asocian con procesos de nivel de núcleo. Múltiples hilos de nivel de usuario que constituyen un único proceso de nivel de usuario se asocian con procesos Linux a nivel de núcleo y comparten el mismo ID de grupo. Esto permite

a estos procesos compartir recursos y evitar los cambios de contexto cuando el planificador cambia entre procesos del mismo grupo.

En Linux se crea un nuevo proceso copiando los atributos del proceso actual. Un nuevo proceso se *clona*, compartiendo archivos, manejadores de señales y memoria virtual. Cuando los dos procesos comparten memoria virtual, funcionan como hilos de un solo proceso.

En lugar del mandato *fork()*, los procesos se crean en Linux usando *clone()*, que incluye un conjunto de *flags* como argumentos. La llamada *fork()* se implementa con *clone()* sin *flags*.

Cuando el núcleo de Linux realiza un cambio de proceso, verifica si la dirección del directorio de páginas del proceso actual es la misma que la del que va a ser planificado, en cuyo caso estarían compartiendo el mismo espacio de direcciones y el cambio de contexto sería básicamente saltar de una posición del código a otra.

Aunque los procesos clonados que son parte del mismo grupo de procesos pueden compartir el mismo espacio de memoria, no comparten la misma pila de usuario. Por tanto, la llamada *clone()* crea espacios de pila separados para cada proceso.

Flags de la llamada *clone()* en Linux:

- **CLONE_CLEARID:** borra el ID de tarea.
- **CLONE_DETACHED:** el padre no quiere el envío de la señal SIGCHLD en su finalización.
- **CLONE_FILES:** compartir la tabla de identificación de archivos abiertos.
- **CLONE_FS:** compartir la tabla que identifica al directorio raíz y al actual de trabajo, así como el valor de la máscara de bits que enmascara los permisos iniciales de un nuevo archivo.
- **CLONE_IDLETASK:** establecer el PID a cero, que se refiere a la tarea *idle*, que se utiliza cuando todas las tareas disponibles están bloqueadas esperando recursos.
- **CLONE_NEWNS:** crea un nuevo espacio de nombres para el hijo.
- **CLONE_PARENT:** el llamante y la nueva tarea comparten proceso padre.
- **CLONE_PTRACE:** si el proceso padre está siendo trazado, el proceso hijo también lo hará.
- **CLONE_SETTID:** escribe el TID en el espacio de usuario.
- **CLONE_SETTLS:** crea un nuevo TLS para el hijo.
- **CLONE_SIGHAND:** comparte la tabla que identifica los manejadores de señales.
- **CLONE_SYSVSEM:** comparte la semántica SEM_UNDO de System V.
- **CLONE_THREAD:** inserta un proceso en el mismo grupo de hilos del padre. La activación de este *flag* fuerza de forma implícita a CLONE_PARENT.
- **CLONE_VFORK:** si está activado, el padre no se planifica para la ejecución hasta que el hijo invoque a la llamada al sistema *execve()*.
- **CLONE_VM:** comparte el espacio de direcciones (descriptor de memoria y todas las tablas de páginas).