RESUMEN INTEGRADOR

UNIDAD 1

**¿Qué es un sistema operativo?**

Un sistema operativo es un programa que administra el hardware de una computadora, actúa como un intermediario entre el usuario de la computadora y el hardware de la misma y actúa como base para los programas de aplicación. El propósito de un sistema operativo es proporcionar un entorno en el que el usuario pueda ejecutar programas de una manera práctica y eficiente.

**Operaciones del sistema operativo**

Los sistemas operativos modernos están controlados mediante interrupciones. Si no hay ningún proceso que ejecutar, ningún dispositivo de E/S al que dar servicio y ningun usuario al que responder, el sistema operativo debe permanecer inactivo, esperando a que algo ocurra. Una excepción es una interrupción generada por software debido a un error o una solicitud específica de un programa de usuario de que se realice un servicio del sistema operativo. Para cada tipo de interrupción, diferentes segmentos de código del sistema determinan qué acción hay que llevar a cabo. Se utilizará una rutina de servicio a la interrupción que es encargada de tratarla.

**Historia de los sistemas operativos**

1. Primera generación: (1945-55)

* Máquinas binarias, tubos de vacío, algunas eran programables, primitivas y lentas.
* Toda la programación se realizó en lenguaje de máquina absoluto.
* Los lenguajes de programación y los sistemas operativos eran desconocidos.
* Prácticamente todos los problemas eran cálculos matemáticos y numéricos sencillos.
* A principios de la década de 1950, se introdujeron las tarjetas perforadas.

2. Segunda generación: (1955-65)

* La introducción del transistor a mediados de la década de 1950 cambió radicalmente la imagen. Las computadoras se volvieron lo suficientemente confiables como para que pudieran ser fabricadas y vendidas a clientes que pagaran (mainframes).
* Para ejecutar un trabajo, un programador primero escribiría el programa en papel (en FORTRAN o ensamblador) y luego lo perforaba en las tarjetas. Luego bajaba a la sala de entrada y se la pasaba a uno de los operadores, cuando la computadora terminaba cualquier trabajo que estaba ejecutando, un operador iba a la impresora, tomaba la salida y la llevaba a la sala de salida, para que el programador pudiera recogerla más tarde. Si se necesitaba el compilador FORTRAN, el operador tendría que sacarlo de un archivador y leerlo. Se perdía mucho tiempo en este proceso, dado el alto costo del equipo, se buscó rápidamente formas de reducir el tiempo perdido. La solución generalmente adoptada fue el sistema batch.
* Batch: La idea detrás de esto era recolectar una bandeja llena de trabajos en la sala de entrada y luego leerlos en una cinta magnética usando una computadora pequeña barata. Después de recopilar un lote de trabajos en las tarjetas, estas se leían en una cinta magnética, que se llevaba a la sala de máquinas, donde se montaba en una unidad de cinta. El operador luego cargaba un programa especial, que leía el primer trabajo de la cinta y lo ejecutaba. La salida se escribía en una segunda cinta, en lugar de imprimirse. Una vez finalizado el trabajo, el sistema operativo leía automáticamente el siguiente trabajo de la cinta y comienza a ejecutarlo. La cinta de salida se llevaba a otra máquina para imprimir fuera de línea.
* Las computadoras grandes de segunda generación se utilizaron principalmente para cálculos científicos y de ingeniería.
* Fueron en gran parte programados en FORTRAN y lenguaje ensamblador.

3. Tercera generación: (1965-80)

* Se incorporaron los circuitos integrados y los SO de propósito general.
* **Multiprogramación:** este sistema operativo tenia la memoria dividida en varias partes, con un trabajo diferente en cada partición. Entonces, mientras un trabajo estaba esperando que se completara la E / S, otro trabajo podría estar usando la CPU. Si se pudieran mantener suficientes trabajos en la memoria principal a la vez, la CPU podría mantenerse ocupada casi el 100% del tiempo.
* **Spooling:** capacidad de leer trabajos de tarjetas en el disco tan pronto como fueron llevados a la sala de computación. Luego, cuando finaliza un trabajo en ejecución, el sistema operativo puede cargar un nuevo trabajo del disco en la partición ahora vacía y ejecutarlo.
* **Timesharing**, una variante de multiprogramación, en la que cada usuario tiene un terminal en línea. En un sistema de tiempo compartido, si 20 usuarios han iniciado sesión y 17 de ellos están pensando o hablando o tomando café, la CPU puede asignarse a los tres trabajos que desean servicio.
* Se crea MULTICS, diseñado para admitir a cientos de usuarios en una máquina solo un poco más potente que una PC con procesador Intel 386, aunque tenía mucha más capacidad de E / S. Hubo muchas razones por las que MULTICS no se apoderó del mundo, una de las cuales fue que estaba escrito en el lenguaje de programación PL / I, y el compilador de PL / I llegó años tarde y apenas funcionó cuando finalmente llegó. Además, MULTICS era enormemente ambicioso para su época. A pesar de su falta de éxito comercial, MULTICS tuvo una gran influencia en los sistemas operativos posteriores (especialmente UNIX y sus derivados, FreeBSD, Linux, iOS y Android).
* Crecimiento de las minicomputadoras.

4. Cuarta generación: (1980-90)

* Desarrollo de circuitos LSI (Integración a gran escala), son chips de silicio que contienen miles de transistores.
* Se crean las computadoras personales comenzó.
* Se crean nuevos sistemas operativos.
* Aparece la GUI.

5. Quinta generación (1990-)

* Sistemas operativos para celulares.
* Teléfonos inteligentes.
* Lenguajes de programación nuevos.

**Tipos de sistemas operativos**

Por la cantidad de procesadores:monoprocesador y multiprocesador.

Sistemas por propósito:

1. General: los sistemas de propósito general se caracterizan por tener un gran número de usuarios trabajando sobre un amplio abanico de aplicaciones. Estos sistemas están diseñados para múltiples usos. Por ejemplo: Linux, Unix, Windows.
2. Específico: están diseñados para un uso en particular, por ejemplo un sistema operativo para heladera, control industrial, etc.

Sistemas por tiempo:

* 1. procesamiento a tiempo real: en estos sistemas una tarea no solo tiene que estar bien hecha, sino que debe ser terminada en un tiempo determinado. En el tiempo real duro, la tarea debe hacerse estrictamente en un tiempo determinado, sino no se hace. Por ejemplo cohetes de defensa, ascensores, etc. En el tiempo real blando la tarea puede terminarse con unos segundos de atraso, por ejemplo una transmisión de TV.
  2. Sistemas no condicionados al tiempo: son aquellos en donde el proceso o tarea a realizar no está condicionado por ningún periodo de tiempo.

Sistemas por hardware: ESTA EN OTRO ARCHIVO.

**Revisión de hardware**

1. Procesadores:

El "cerebro" de la computadora es la CPU. Obtiene instrucciones de la memoria y las ejecuta. Cada CPU tiene un conjunto específico de instrucciones que puede ejecutar. Debido a que acceder a la memoria para obtener una instrucción o palabra de datos lleva mucho más tiempo que ejecutar una instrucción, todas las CPU contienen algunos registros internos para contener variables clave y resultados temporales. Cuando se comparte el tiempo de la CPU, el sistema operativo a menudo detiene el programa en ejecución para (re) iniciar otro. Cada vez que detiene un programa en ejecución, el sistema operativo debe guardar todos los registros para que puedan restaurarse cuando el programa se ejecute más tarde.  
La mayoría de las CPU, excepto las muy simples que se usan en sistemas integrados, tienen dos modos, modo de kernel y modo de usuario.

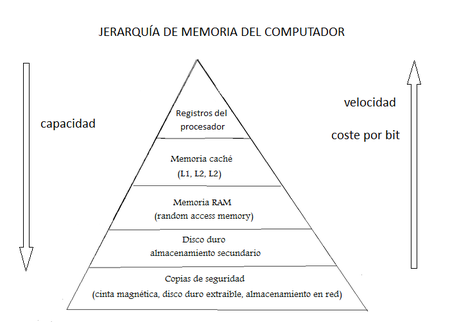
1. Estructuras de almacenamiento:

**Memoria RAM:** es una memoria de acceso directo, rápido e indistinto. Los programas de la computadora deben hallarse en esta memoria principal para ejecutarse. Se guardan en ella las variables y conjunto de instrucciones de los procesos que se ejecutan, en esta memoria también se almacena el kernel. Se dice que es una memoria rápida porque la CPU, a través de instrucciones propias, tiene acceso a ella.

**Memoria secundaria:** es una extensión de la memoria principal, lo importante de esta memoria es que se pueden almacenar grandes cantidades de datos de forma permanente. El dispositivo de almacenamiento secundario más común es el disco magnético.

**Unidades de cinta:** esta se denomina memoria terciaria, tienen la característica de ser removibles y soportar gran cantidad de datos.

Existen muchos sistemas de almacenamiento, que difieren en velocidad, coste y tamaño. Según estos parámetros se origina una jerarquía, donde los niveles superiores corresponden a sistemas de almacenamiento caros pero rápidos. A medida que se desciende el coste por bit generalmente disminuye mientras que el tiempo de acceso aumenta.



3. Estructura de E/S:

Gran parte del código del sistema operativo se dedica a gestionar la entrada y la salida, debido a su importancia para la fiabilidad y rendimiento del sistema, y también debido a la variada naturaleza de los dispositivos.

Una computadora consta con una CPU y drivers que se conectan a través de buses en común. Cada driver maneja un tipo específico de dispositivo.

La controladora del canal contiene algunos búferes locales y un conjunto de registros de propósito especial, es responsable de transferir los datos entre los dispositivos periféricos que controla y su búfer local.

Normalmente los SO tienen un driver para cada controladora del dispositivo. Estos drivers son capaces de entenderse con la controladora hardware del dispositivo.

Al iniciar un operación de E/S el controlador del dispositivo recibe los comandos a ejecutar desde el driver, el cual pone esos comandos en un lenguaje que la controladora entiende.. La controladora inicia la transferencia de datos desde el dispositivo hasta su búfer local y, una vez finalizada la ejecución, informa al controlador del canal que a través de una interrupción avisa a la CPU que terminó la operación. El controlador del canal entonces devuelve el control al SO, devolviendo los datos o la información.

Esta forma es buena opción para transferir cantidades pequeñas de datos, pero es un desperdicio de capacidad de proceso cuando se usa para movimientos masivos de datos, como en la E/S de disco. Para resolver esto se usa el acceso directo a memoria (DMA) que es el canal de comunicación entre periféricos rápidos y la unidad de control de la memoria. De esta forma la controladora de un dispositivo transfiere un bloque entero de datos entre su propio búfer y la memoria, sin que intervenga la CPU. El controlador del dispositivo solo genera una interrupción por cada bloque, para decir al controlador del canal, a través del driver, que la operación se ha completado, en vez de la interrupción por byte generada en los dispositivos de baja velocidad. Y así, la CPU está disponible para realizar otros trabajos.

El SO puede comunicarse con los periféricos a través de interrupciones, o de escrutinio. En el caso de las interrupciones puede ser por hardware y generalmente son hechas por los periféricos de alta velocidad como el disco duro. Y en el caso del escrutinio la UC del canal pregunta constantemente a los periféricos si hay algo que atender, esta forma se usa en los periféricos de baja velocidad como el teclado.

4. Protección y seguridad:  
Si un sistema tiene múltiples usuarios y permite la ejecución concurrente de múltiples procesos, entonces el acceso a los datos debe regularse. Para eso, se emplean mecanismos que aseguren que sólo puedan utilizar los recursos (archivos, segmentos de memoria, CPU, etc) aquellos procesos que hayan obtenido la autorización del sistema operativo.

Por tanto, **protección** es cualquier mecanismo que controle el acceso de procesos y usuarios a los recursos definidos por un sistema informático.

Un sistema puede tener la protección adecuada pero estar expuesto a fallos y permitir acciones inapropiadas. Es responsabilidad de los mecanismos de **seguridad** defender el sistema frente a ataques internos y externos. Tales ataques abarcan un enorme rango en el que incluyen los virus y gusanos, los ataques de denegación de seguridad, el robo de identidad, entre otros.

5. Modo dual:

Para asegurar la correcta ejecución del sistema operativo, tenemos que diferenciar entre la ejecución de código del sistema operativo y el código definido por el usuario. El método que utilizan la mayoría de los sistemas operativos consiste en proporcionar soporte hardware que nos permita diferenciar entre varios modos de ejecución.

Como mínimo necesitamos dos modos diferentes de operación: modo usuario y modo kernel. Un bit, denominado bit de modo, se agrega al hardware de la computadora para indicar el modo actual: usuario o kernel. El modo dual de operación nos proporciona los medios para proteger al sistema operativo de los usuarios que puedan causar errores, y también para proteger a los usuarios de los errores de otros usuarios. Esta protección se consigue asignando algunas de las instrucciones de máquina que pueden causar daño como instrucciones privilegiadas, el hardware hace que las instrucciones privilegiadas sólo se ejecuten en modo kernel, si se hace un intento de ejecutar una instrucción privilegiada en modo usuario el hardware no solo no ejecuta la instrucción sino que la trata como ilegal y manda una interrupción al sistema operativo.

**Estructuras del sistema operativo**

**Interfaz de usuario:**

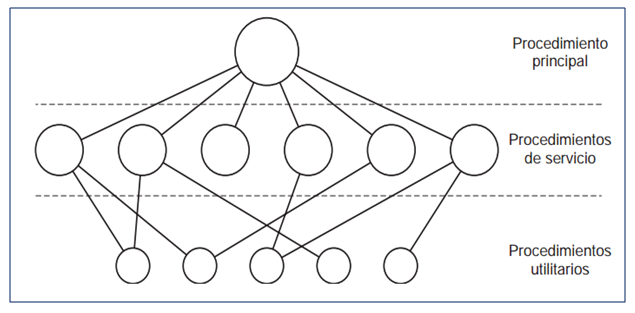
A. interprete de comandos: este permite al usuario introducir directamente comandos que el sistema operativo pueda ejecutar. Cuando los sistemas disponen de varios intérpretes entre los que elegir, los intérpretes se conocen como shells. La función principal del intérprete de comandos es obtener y ejecutar el siguiente comando especificado por el usuario. Existen dos formas en que pueden implementarse los comandos: una el que el propio intérprete contenga el código que el comando tiene que ejecutar, y otra es que se implementan los comandos a través de una serie de programas del sistemas, en este caso el intérprete no “entiende” el comando sino que simplemente lo usa para identificar el archivo que hay que cargar en memoria y ejecutar.

B. Interfaces gráficas de usuario: esta manera de interactuar con el sistema operativo es más amigable, permite a los usuarios emplear un sistema de ventanas y menús desplegables controlable mediante el mouse. Proporciona una especie de escritorio en el que el usuario mueve el ratón para colocar su puntero sobre imágenes, o íconos, que se muestran en la pantalla y que representan programas, archivos, directorios y funciones del sistema.

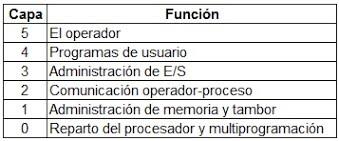
**Llamadas al sistema:** es la manera que los procesos tienen para invocar servicios que el sistema operativo ofrece.

**Estructuras**

1. Sistemas monolíticos: en el enfoque monolítico, todo el sistema operativo se ejecuta como un solo programa en modo kernel. Cada procedimiento en el sistema es libre de llamar a cualquier otro. Ser capaz de llamar a cualquier procedimiento que desee es muy eficiente, pero tener miles de procedimientos que pueden llamarse entre sí sin restricciones también puede llevar a un sistema que es difícil de entender y difícil de manejar. Además, una falla en cualquiera de estos procedimientos eliminará todo el sistema operativo.

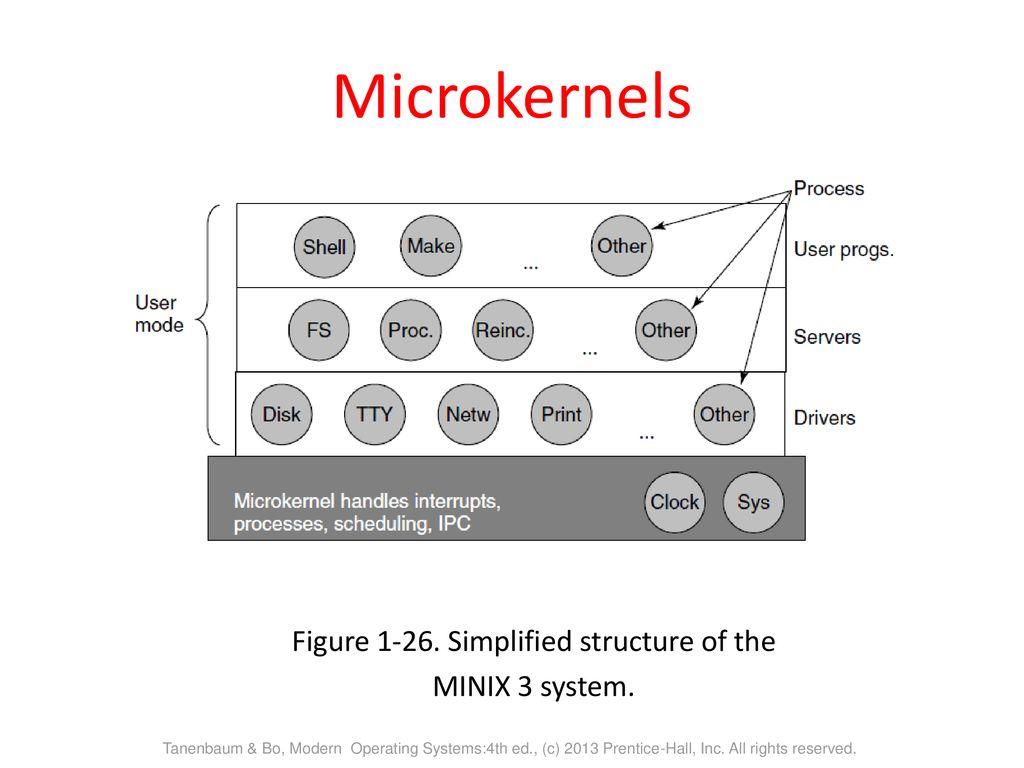


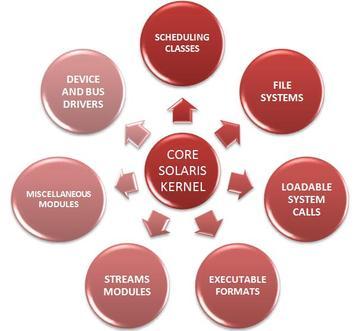
2. Sistemas de capas: este sistema operativo está organizado como una jerarquía de capas, cada una construida sobre la que está debajo. Y cada capa solo puede comunicarse directamente con las capas que tiene al lado. Un ejemplo es el sistema THE, construido de esta forma:



3. Microkernels: La idea básica detrás del diseño del microkernel es lograr una alta confiabilidad dividiendo el sistema operativo en pequeños módulos bien definidos, solo uno de los cuales, el microkernel, se ejecuta en modo kernel y el resto se ejecuta como procesos de usuario ordinarios relativamente impotentes. En particular, al ejecutar cada controlador de dispositivo y sistema de archivos como un proceso de usuario separado, un error en uno de estos puede bloquear ese componente, pero no puede bloquear todo el sistema. Por lo tanto, un error en el controlador de audio hará que el sonido se distorsione o se detenga, pero no bloqueará la computadora. En contraste, en un sistema monolítico con todos los controladores en el kernel, un controlador de audio defectuoso puede fácilmente hacer referencia a una dirección de memoria no válida y hacer que el sistema se detenga de forma instantánea.

Fuera del kernel, el sistema está estructurado como tres capas de procesos que se ejecutan en modo de usuario. De la siguiente manera:



4. Sistemas modulares: Quizá la mejor metodología actual para diseñar sistemas operativos es la que usa las técnicas de programación orientada a objetos para crear un *kernel* modular. En este caso, el *kernel* dispone de un conjunto de componentes fundamentales y enlaza dinámicamente los servicios adicionales, bien durante el arranque o en tiempo de ejecución. Tal estrategia utiliza módulos que se cargan dinámicamente y resulta habitual en las implementaciones modernas de UNIX, como Solaris, Linux y Mac OS X.

5. Sistemas híbridos.

UNIDAD 2

El proceso

Un proceso es un programa en ejecución más todos los recursos necesarios para que el programa se ejecute.

Estado del proceso

A medida que se ejecuta el proceso, este va cambiando de estado. El estado del proceso se define según la actividad actual de dicho proceso. Cada proceso puede estar en uno de los siguientes estados:

Nuevo: el proceso acaba de ser creado.

En ejecución: se están ejecutando las instrucciones.

En espera: el proceso está esperando a que se produzca un suceso (como la terminación de una operación de E/S o a la recepción de una señal).

Preparado: el proceso está a la espera de que le asignen a un procesador.

Terminado: ha terminado la ejecución del proceso.

Solo puede haber un solo proceso ejecutándose en cualquier procesador en cada instante concreto.

Bloque de control de proceso: PCB.

Cada proceso se representa con uno de estos, tienen información específica para cada proceso entre lo que se incluyen:

Estado del proceso: nuevo, preparado, en ejecución, en espera, detenido, etc.

Contador de programa: el contador indica la dirección de la siguiente instrucción que va a ejecutar dicho proceso.

Prioridad del proceso, valor de los registros base y límite,cantidad de CPU y de tiempo real empleados, límites de tiempo asignado, número de proceso, lista de los dispositivos de E/S asignados al proceso, valor de los registros de la CPU, etc.

Colas de planificación

A medida que los procesos entran en el sistema, se colocan en una cola de procesos. Los procesos que están en la memoria principal y están preparados o en espera de ejecutarse se mantienen en una cola de procesos preparados.

Cuando se asigna la CPU a un proceso, este se ejecuta durante un rato y finalmente termina, es interrumpido o espera a que se produzca un determinado suceso, como la terminación de una solicitud de E/S de algún otro proceso. Dado que hay muchos procesos en el sistema, el dispositivo de E/S puede tener varios procesos esperando para usarlo, por lo tanto se crea una cola del dispositivo. Cada dispositivo tiene su propia cola.

Cada proceso nuevo se coloca inicialmente en la cola de procesos preparados, donde se espera hasta que es seleccionado para ejecución, es decir, hasta que es despachado. Una vez que se asigna la CPU al proceso y este comienza a ejecutarse puede pasar:

El proceso podría...

1. ejecutar una solicitud de E/S y ser colocado en una cola de E/S.

2. crear un nuevo subproceso y esperar a que este termine.

3. ser desalojado de la CPU como resultado de una interrupción y puesto de nuevo en la cola de procesos preparados.

En los dos primeros casos, el proceso terminará, antes o después, por cambiar del estado de espera al estado preparado y será colocado de nuevo en la cola de procesos preparados. Los procesos siguen este ciclo hasta que termina su ejecución, momento en el que se elimina el proceso de todas las colas y su PCB, y se liberan sus recursos.

Planificadores.

Durante su tiempo de vida, los procesos se mueven entre las diversas colas de planificación. El sistema operativo, como parte de la tarea de planificación, debe seleccionar, mediante un planificador apropiado, de alguna manera los procesos que se encuentran en estas colas.

A menudo, en un sistema de procesamiento por lotes, se envían más procesos de los que pue­den ser ejecutados de forma inmediata. Estos procesos se guardan en cola en un dispositivo de almacenamiento masivo, donde se mantienen para su posterior ejecu­ción.

El planificador a largo plazo selecciona procesos de esta cola y los carga en memoria para su ejecución.

El planificador a corto plazo selecciona de entre los procesos que ya están preparados para ser ejecutados y asigna la CPU a uno de ellos.

La principal diferencia entre estos dos planificadores se encuentra en la frecuencia de ejecu­ción:

El planificador a corto plazo debe seleccionar un nuevo proceso para la CPU frecuentemen­te. Un proceso puede ejecutarse sólo durante unos pocos milisegundos antes de tener que esperar por una solicitud de E/S. Normalmente, el planificador a corto plazo se ejecuta al menos una vez cada 100 milisegundos. Debido al poco tiempo que hay entre ejecuciones, el planificador a corto plazo debe ser rápido.

El planificador a largo plazo se ejecuta mucho menos frecuentemente; pueden pasar minutos entre la creación de un nuevo proceso y el siguiente. El planificador a largo plazo controla el grado de multiprogramación (el número de procesos en memoria). Si el grado de multiprogramación es estable, entonces la tasa promedio de creación de procesos debe ser igual a la tasa promedio de salida de procesos del sistema. Por tanto, el planificador a largo plazo puede tener que invocarse sólo cuando un proceso abandona el sistema. Puesto que el intervalo entre ejecuciones es más largo, el planificador a largo plazo puede permitirse emplear más tiempo en decidir qué proceso debe seleccionarse para ser ejecutado.

Es importante que el planificador a largo plazo haga una elección cuidadosa. En general, la mayoría de los procesos pueden describirse como limitados por la E/S o limitados por la CPU.

Un proceso limitado por E/S es aquel que invierte la mayor parte de su tiempo en operaciones de E/S en lugar de en realizar cálculos. Por el contrario, un proceso limitado por la CPU genera solicitu­des de E/S con poca frecuencia, usando la mayor parte de su tiempo en realizar cálculos. Es importante que el planificador a largo plazo seleccione una adecuada mezcla de procesos, equili­brando los procesos limitados por E/S y los procesos limitados por la CPU. Si todos los procesos son limitados por la E/S, la cola de procesos preparados casi siempre estará vacía y el planifica­dor a corto plazo tendrá poco que hacer. Si todos los procesos son limitados por la CPU, la cola de espera de E/S casi siempre estará vacía, los dispositivos apenas se usarán, y de nuevo el sistema se desequilibra. Para obtener un mejor rendimiento, el sistema dispondrá entonces de una com­binación equilibrada de procesos limitados por la CPU y de procesos limitados por E/S.

Un planificador a mediano plazo lo que hace es controlar cada cierto tiempo que la planificación hecha por el planificador a largo plazo se esté llevando de la mejor manera. Si esta planificación esta de alguna manera alterada, el planificador a corto plazo le avisa al de largo plazo.

Cambio de contexto.

Las interrupciones hacen que el sistema operativo obligue a la CPU a abandonar su tarea actual, para ejecutar una rutina del kernel. Estos sucesos se producen con frecuencia en los sistemas de propósito general. Cuando se produce una interrupción el sistema tiene que guardar el contexto actual del proceso que se está ejecutando en la CPU, de modo que pueda restaurar dicho contexto cuando su procesamiento concluya, suspendiendo el proceso y reanudándose después. El contexto se almacena en el PCB del proceso e incluye el valor de los registros de la CPU, el estado del proceso y la información de gestión de memoria.

La conmutación de la CPU a otro proceso requiere una salvaguarda del estado del proceso actual y una restauración del estado de otro proceso diferente. Esta tarea se conoce como cambio de contexto. Cuando se produce un cambio de contexto, el SO guarda el contexto del proceso antiguo en su PCB y carga el contexto almacenado del nuevo proceso que se ha decidido ejecutar.

El tiempo dedicado al cambio de contexto es tiempo desperdiciado, dado que el sistema no realiza ningún trabajo útil durante la conmutación.

Operaciones sobre los procesos

En la mayoría de los sistemas, los procesos pueden ejecutarse de forma concurrente y pueden cre­arse y eliminarse dinámicamente. Por tanto, estos sistemas deben proporcionar un mecanismo para la creación y terminación de procesos.

Creación de procesos

Un proceso puede crear otros varios procesos nuevos mientras se ejecuta; para ello se utiliza una llamada al sistema específica para la creación de procesos. En general, un proceso necesitará ciertos recursos (tiempo de CPU, memoria, archivos, disposi­tivos de E/S) para llevar a cabo sus tareas. Cuando un proceso crea un subproceso, dicho subproceso puede obtener sus recursos directamente del sistema operativo o puede estar restringido a un subconjunto de los recursos del proceso padre. Restringir un proceso hijo a un subconjunto de los recursos del padre evita que un proceso pueda sobrecargar el sistema creando demasiados subprocesos.

Además de los diversos recursos físicos y lógicos que un proceso obtiene en el momento de su creación, el proceso padre puede pasar datos de inicialización (entrada) al proceso hijo..

Cuando un proceso crea otro proceso nuevo, existen dos posibilidades en términos de ejecu­ción:

1. El padre continúa ejecutándose concurrentemente con su hijo.

2. El padre espera hasta que alguno o todos los hijos han terminado de ejecutarse.

También existen dos posibilidades en función del espacio de direcciones del nuevo proceso:

1. El proceso hijo es un duplicado del proceso padre (usa el mismo programa y los mismos datos que el padre).

2. El proceso hijo carga un nuevo programa.

Comunicación interprocesos

Los procesos que se ejecutan concurrentemente pueden ser procesos independientes o procesos cooperativos.

Un proceso es independiente si no puede afectar o verse afectado por los restantes procesos que se ejecutan en el sistema. Cualquier proceso que no comparte datos con ningún otro proceso es un proceso independiente.

Un proceso es cooperativo si puede afectar o verse afecta­do por los demás procesos que se ejecutan en el sistema, cualquier proceso que comparte datos con otros procesos es un proceso cooperativo.

Hay varias razones para proporcionar un entorno que permita la cooperación entre procesos:

• Compartir información. Dado que varios usuarios pueden estar interesados en la misma información, debemos proporcionar un entorno que permita el acceso concurrente a dicha información.

• Acelerar los cálculos. Si deseamos que una determinada tarea se ejecute rápidamente, debemos dividirla en subtareas, ejecutándose cada una de ellas en paralelo con las demás. Observe que tal aceleración sólo se puede conseguir si la computadora tiene múltiples ele­mentos de procesamiento, como por ejemplo varias CPU o varios canales de E/S.

• Modularidad. Podemos querer construir el sistema de forma modular, dividiendo las fun­ciones del sistema en diferentes procesos o hebras.

• Conveniencia. Incluso un solo usuario puede querer trabajar en muchas tareas al mismo tiempo.

La cooperación entre procesos requiere mecanismos de comunicación interprocesos (IPC, interprocess communication) que les permitan intercambiar datos e información.

Existen dos modelos fundamentales de comunicación interprocesos:

(1) memoria compartida se establece una región de la memoria para que sea compartida por los procesos cooperativos. De este modo, los procesos pueden intercambiar información leyendo y escribiendo datos en la zona compartida.

(2) paso de mensajes la comunicación tiene lugar mediante el intercambio de mensajes entre los procesos coopera­tivos.

La memoria compartida es más rápida que el paso de mensa­jes, ya que el último se implementa normalmente usando llamadas al sistema y, por tanto, requiere que intervenga el SO.

En los siste­mas de memoria compartida, las llamadas al sistema sólo son necesarias para establecer las zonas de memoria compartida. Una vez establecida, todos los accesos se tratan como accesos a memoria rutinarios y no se precisa la ayuda del SO.

Sistemas de memoria compartida

La comunicación interprocesos que emplea memoria compartida requiere que los procesos que se estén comunicando establezcan una región de memoria compartida. Normalmente, una región de memoria compartida reside en el espacio de direcciones del proceso que crea el segmento de memoria compartida. Otros procesos que deseen comunicarse usando este segmento de memoria compartida deben conectarse a su espacio de direcciones. El siste­ma operativo intenta evitar que un proceso acceda a la memoria de otro proceso. La memoria compartida requiere que dos o más procesos acuerden eliminar esta restricción. Entonces podrán intercambiar información leyendo y escribiendo datos en las áreas compartidas. El formato de los datos y su ubicación están determinados por estos procesos, y no se encuentran bajo el control del sistema operativo. Los procesos también son responsables de verificar que no escriben en la misma posición simultáneamente.

Un proceso productor genera información que consume un proceso consumidor.

Una solución para el problema del productor-consumidor es utilizar mecanismos de memoria compartida. Para permitir que los procesos productor y consumidor se ejecuten de forma concu­rrente, debemos tener disponible un búfer de elementos que pueda rellenar el productor y vaciar el consumidor. Este búfer residirá en una región de memoria que será compartida por ambos pro­cesos, consumidor y productor. Un productor puede generar un elemento mientras que el consu­midor consume otro. El productor y el consumidor deben estar sincronizados, de modo que el consumidor no intente consumir un elemento que todavía no haya sido producido.

Pueden emplearse dos tipos de búferes:

El sistema de búfer no limitado no pone límites al tamaño de esa memoria compartida. El consumidor puede tener que esperar para obtener elemen­tos nuevos, pero el productor siempre puede generar nuevos elementos.

El sistema de búfer limi­tado establece un tamaño de búfer fijo. En este caso, el consumidor tiene que esperar si el búfer está vacío y el productor tiene que esperar si el búfer está lleno.

Sea la comunicación directa o indirecta, los mensajes intercambiados por los procesos que se están comunicando residen en una cola temporal. Básicamente, tales colas se pueden implementar de tres maneras:

• Capacidad cero. La cola tiene una longitud máxima de cero; por tanto, no puede haber nin­gún mensaje esperando en el enlace. En este caso, el transmisor debe bloquearse hasta que el receptor reciba el mensaje.

• Capacidad limitada. La cola tiene una longitud finita n; por tanto, puede haber en ella n mensajes como máximo. Si la cola no está llena cuando se envía un mensaje, el mensaje se introduce en la cola (se copia el mensaje o se almacena un puntero al mismo), y el transmi­sor puede continuar la ejecución sin esperar. Sin embargo, la capacidad del enlace es finita. Si el enlace está lleno, el transmisor debe bloquearse hasta que haya espacio disponible en la cola.

• Capacidad ilimitada. La longitud de la cola es potencialmente infinita; por tanto, puede haber cualquier cantidad de mensajes esperando en ella. El transmisor nunca se bloquea.

Sistemas de paso de mensajes

El paso de mensajes proporciona un mecanismo que permite a los procesos comunicarse y sin­cronizar sus acciones sin compartir el mismo espacio de direcciones, y es especialmente útil en un entorno distribuido, en el que los procesos que se comunican pueden residir en diferentes com­putadoras conectadas en red.

Una facilidad de paso de mensajes proporciona al menos dos operaciones: envío de mensajes (send) y recepción de mensajes (receive).

Si los procesos P y Q desean comunicarse, tienen que enviarse mensajes entre sí; debe existir un enlace de comunicaciones entre ellos. Existen varios métodos para implementar lógicamente un enlace y las operaciones de envío y recepción:

• Comunicación directa o indirecta.

• Comunicación síncrona o asíncrona.

Hebras

Una hebra es una unidad básica de utilización de la CPU. Comparte con otras hebras que pertenecen al mismo proceso la sección de código, la sección de datos y otros recursos del sistema operativo. Un proceso tradicional tiene una sola hebra de control. Si un proceso tiene, por el contrario, múltiples hebras de control, puede realizar más de una tarea a la vez.

Ventajas

Las ventajas de la programación multihebra pueden dividirse en cuatro categorías principales:

1. Capacidad de respuesta. El uso de múltiples hebras en una aplicación interactiva permite que un programa continúe ejecutándose incluso aunque parte de él esté bloqueado o reali­zando una operación muy larga, lo que incrementa la capacidad de respuesta al usuario.

2. Compartición de recursos. Por omisión, las hebras comparten la memoria y los recursos del proceso al que pertenecen.

3. Economía. La asignación de memoria y recursos para la creación de procesos es costosa. Dado que las hebras comparten recursos del proceso al que pertenecen, es más económico crear y realizar cambios de contexto entre unas y otras hebras.

4. Utilización sobre arquitecturas multiprocesador. Las ventajas de usar configuraciones multihebra pueden verse incrementadas significativamente en una arquitectura multiproce­sador, donde las hebras pueden ejecutarse en paralelo en los diferentes procesadores.

El soporte para hebras puede proporcionarse en el nivel de usua­rio (para las hebras de usuario) o por parte del kernel (para las hebras del kernel). El soporte para las hebras de usuario se proporciona por encima del kernel y las hebras se gestionan sin soporte del mismo, mientras que el sistema operativo soporta y gestiona directamente las hebras del ker­nel.

En último término, debe existir una relación entre las hebras de usuario y las del kernel; en esta sección, vamos a ver tres formas de establecer esta relación.

Modelo muchos-a-uno

El modelo muchos-a-uno asigna múltiples hebras del nivel de usuario a una hebra del kernel, el proceso completo se bloquea si una hebra realiza una llamada blo­queante al sistema. También, dado que sólo una hebra puede acceder al kernel cada vez, no podrán ejecutarse varias hebras en paralelo sobre múltiples procesadores.

Modelo uno-a-uno

El modelo uno-a-uno asigna cada hebra de usuario a una hebra del kernel. Proporciona una mayor concurrencia que el modelo muchos-a-uno, permitiendo que se ejecute otra hebra mientras una hebra hace una llamada bloqueante al sistema; también permite que se ejecuten múltiples hebras en paralelo sobre varios procesadores. El único inconveniente de este modelo es que crear una hebra de usuario requiere crear la correspondiente hebra del kernel. La carga de trabajo administrativa para la creación de hebras del kernel puede repercutir en el rendimiento de una aplicación.

Modelo muchos-a-muchos

El modelo muchos-a-muchos multiplexa muchas hebras de usuario sobre un número menor o igual de hebras del kernel.

Mientras que el modelo muchos-a-uno permite al desarrollador crear tantas hebras de usuario como desee, no se consigue una concurrencia real, ya que el kernel sólo puede planificar la ejecución de una hebra cada vez.

El modelo uno-a-uno permite una mayor concurrencia, pero el desarrollador debe tener cuidado de no crear demasiadas hebras dentro de una aplicación (y, en algunos casos, el número de hebras que pueda crear estará limitado).

El modelo muchos-a-muchos no sufre ninguno de estos inconvenientes. Los desarrolladores pueden crear tantas hebras de usuario como sean nece­sarias y las correspondientes hebras del kernel pueden ejecutarse en paralelo en un multiprocesador (varias CPU). Asimismo, cuando una hebra realiza una llamada bloqueante al sistema, el kernel puede planificar otra hebra para su ejecución.

Una popular variación del modelo muchos-a-muchos multiplexa muchas hebras del nivel de usuario sobre un número menor o igual de hebras del kernel, pero también permite acoplar una hebra de usuario a una hebra del kernel.

**Planificadores**

Conceptos básicos

En un sistema de un único procesador, sólo puede ejecutarse un proceso cada vez; cualquier otro proceso tendrá que esperar hasta que la CPU quede libre y pueda volver a planificarse. El objeti­vo de la multiprogramación es tener continuamente varios procesos en ejecución, con el fin de maximizar el uso de la CPU. La idea es bastante simple: un proceso se ejecuta hasta que tenga que esperar, normalmente porque es necesario completar alguna solicitud de E/S. En un sistema infor­mático simple, la CPU permanece entonces inactiva y todo el tiempo de espera se desperdicia; no se realiza ningún trabajo útil. Con la multiprogramación, se intenta usar ese tiempo de forma pro­ductiva. En este caso, se mantienen varios procesos en memoria a la vez. Cuando un proceso tiene que esperar, el sistema operativo retira el uso de la CPU a ese proceso y se lo cede a otro proceso. Este patrón se repite continuamente y cada vez que un proceso tiene que esperar, otro proceso puede hacer uso de la CPU.

Planificador de la CPU

Cuando la CPU queda inactiva, el sistema operativo debe seleccionar uno de los procesos que se encuentran en la cola de procesos preparados para ejecución. El planificador a corto plazo (o pla­nificador de la CPU) lleva a cabo esa selección del proceso. El planificador elige uno de los proce­sos que están en memoria preparados para ejecutarse y asigna la CPU a dicho proceso.

Observe que la cola de procesos preparados no necesariamente tiene que ser una cola FIFO. Sin embargo, conceptualmente, todos los procesos que se encuentran en la cola de procesos preparados se ponen en fila esperando la oportuni­dad de ejecutarse en la CPU. Los registros que se almacenan en las colas son, generalmente, blo­ques de control de proceso (PCB) que describen los procesos en cuestión.

Planificación con desalojo.

Puede ser necesario tomar decisiones sobre planificación de la CPU en las siguientes cuatro cir­cunstancias:

1. Cuando un proceso cambia del estado de ejecución al estado de espera (por ejemplo, como resultado de una solicitud de E/S o de una invocación de wait para esperar a que termine uno de los procesos hijo).

2. Cuando un proceso cambia del estado de ejecución al estado preparado (por ejemplo, cuan­do se produce una interrupción).

3. Cuando un proceso cambia del estado de espera al estado preparado (por ejemplo, al com­pletarse una operación de E/S).

4. Cuando un proceso termina.

En las situaciones 1 y 4, no hay más que una opción en términos de planificación: debe seleccio­narse un nuevo proceso para su ejecución (si hay algún proceso en la cola de procesos preparados). Sin embargo, en las situaciones 2 y 3 sí que existe la opción de planificar un nuevo proceso o no.

Cuando las decisiones de planificación sólo tienen lugar en las circunstancias 1 y 4, decimos que el esquema de planificación es sin desalojo.

En caso contrario, se trata de un esquema apropiativo (con desalojo). En la planificación sin desalojo, una vez que se ha asignado la CPU a un proceso, el proceso se mantiene en la CPU hasta que ésta es liberada bien por la terminación del pro­ceso o bien por la conmutación al estado de espera.

Despachador

Otro componente implicado en la función de planificación de la CPU es el despachador. El despa­chador es el módulo que proporciona el control de la CPU a los procesos seleccionados por el planificador a corto plazo.

El despachador debe ser lo más rápido posible, ya que se invoca en cada conmutación de pro­ceso. El tiempo que tarda el despachador en detener un proceso e iniciar la ejecución de otro se conoce como latencia de despacho.

Algoritmos de planificación

La cuestión de la planificación de la CPU aborda el problema de decidir a qué proceso de la cola de procesos preparados debe asignársele la CPU.

Planificación FCFS (batch)

El algoritmo más simple de planificación de la CPU es, con mucho, el algoritmo FCFS. Con este esquema, se asigna primero CPU al proceso que primero la solicite. La implementación de la política FCFS se gestiona fácilmente con una cola FIFO. Cuando un proceso entra en la cola de procesos preparados, su PCB se coloca al final de la cola. Cuando la CPU queda libre, se asigna al proceso que esté al principio de la cola y ese proceso que pasa a ejecutarse se elimina de la cola.

Sin embargo, el tiempo medio de espera con el algoritmo FCFS es a menudo bastante larga.

El problema con este algoritmo es que se produ­ce lo que se denomina un efecto convoy, suponga que hay procesos orientados a E/S esperando a usar la CPU y un proceso orientado a CPU utilizandola, todos los otros procesos deben esperar a que ese proceso termine y se produce un retraso. Este efecto da lugar a una utilización de la CPU y de los dispositivos menor que la que se conseguiría si se permitiera a los procesos más cortos ejecutarse primero.

El algoritmo de planificación FCFS es sin desalojo. Una vez que la CPU ha sido asignada a un proceso, dicho proceso conserva la CPU hasta que la libera, bien porque termina su ejecución o porque realiza una solicitud de E/S. Sería desastroso permitir que un proceso mantuviera la CPU durante un largo período de tiempo.

Planificación SJF (batch)

Otro método de planificación de la CPU es el algoritmo de planificación con selección del trabajo más corto (SJF). Este algoritmo asocia con cada proceso la duración de la siguien­te ráfaga de CPU del proceso. Cuando la CPU está disponible, se asigna al proceso que tiene la siguiente ráfaga de CPU más corta. Si las siguientes ráfagas de CPU de dos procesos son iguales, se usa algún algoritmo de desempate para determinar quién va primero (algoritmo de prioridad, FcFs, etcétera).

El algoritmo de planificación SJF es probablemente óptimo, en el sentido de que proporciona el tiempo medio de espera mínimo para un conjunto de procesos dado. Anteponer un proceso corto a uno largo disminuye el tiempo de espera del proceso corto en mayor medida de lo que incre­menta el tiempo de espera del proceso largo. Consecuentemente, el tiempo medio de espera dismi­nuye.

La dificultad real del algoritmo SJF es conocer la duración de la siguiente solicitud de CPU. En una planificación a largo plazo de trabajos en un sistema de procesamiento por lotes, podemos usar como duración el límite de tiempo del proceso que el usuario especifique en el momento de enviar el trabajo.

Aunque el algoritmo SJF es óptimo, no se puede implementar en el nivel de la planificación de la CPU a corto plazo, ya que no hay forma de conocer la duración de la siguiente ráfaga de CPU. Un método consiste en intentar aproximar la planificación SJF: podemos no conocer la duración de la siguiente ráfaga de CPU, pero podemos predecir su valor, por el procedimiento de confiar en que la siguiente ráfaga de CPU será similar en duración a las anteriores.

Planificación por prioridades

A cada proceso se le asocia una prioridad y la CPU se asigna al proceso que tenga la prioridad más alta. Los procesos con la misma prioridad se planifican con algún algoritmo de desempate para determinar quién va primero (FcFs, etcétera).

La planificación por prioridades puede ser con o sin desalojo. Cuando un proceso llega a la cola de procesos preparados, su prioridad se compara con la prioridad del proceso actualmente en ejecución. Un algoritmo de planificación por prioridades con desalojo, expulsará de la CPU al proceso actual si la prioridad del proceso que acaba de llegar es mayor. Si es sin desalojo, pondrá al proceso en la cola nuevamente.

Un problema importante de los algoritmos de planificación por prioridades es el bloqueo indefinido o la muerte por inanición, un algoritmo de planificación por prioridades puede dejar a algunos procesos de baja prioridad esperando indefinidamente. Generalmente, ocurrirá alguna de dos cosas: o bien el proceso se ejecutará finalmente o bien el sistema informático ter­minará fallando y se perderán todos los procesos con baja prioridad no terminados.

Una solución al problema del bloqueo indefinido de los procesos de baja prioridad consiste en aplicar mecanismos de envejecimiento. Esta técnica consiste en aumentar gradualmente la prio­ridad de los procesos que estén esperando en el sistema durante mucho tiempo. Otra, es poner un límite que diga cuantas veces ese proceso puede ser evitado.

Planificación round robin

Es similar a la planificación FCFS, pero se añade la técnica de des­alojo para conmutar entre procesos. En este tipo de sistema se define una pequeña unidad de tiempo, denominada quantum de tiempo. La cola de procesos preparados se trata como una cola circular. El planificador de la CPU recorre la cola de procesos preparados, asignando la CPU a cada proceso durante un intervalo de tiempo de hasta un quantum de tiempo.

Puede ocurrir una de dos cosas. El proceso puede tener una ráfaga de CPU cuya duración sea menor que el quantum de tiempo; en este caso, el propio proceso liberará voluntariamente la CPU. El planificador continuará entonces con el siguiente proceso de la cola de procesos preparados. En caso contrario, si la ráfaga de CPU del proceso actualmente en ejecución tiene una duración mayor que el quantum de tiempo, se producirá un fin de cuenta del temporizador y éste enviará una inte­rrupción al sistema operativo; entonces se ejecutará un cambio de contexto y el proceso se coloca­rá al final de la cola de procesos preparados. El planificador de la CPU seleccionará a continuación el siguiente proceso de la cola de procesos preparados. El algoritmo de planificación por turnos incluye, por tanto, un mecanismo de desalojo.

Round Robin dos colas

Este algoritmo presenta dos colas, una activa y una pasiva. En primer lugar todos los algoritmos están en la cola activa, peleando por la CPU, cada uno de ellos tiene un quantum de tiempo que van usando hasta que se les agote. A medida que se les agota el quantum a los procesos, estos pasan a la cola de pasivos. Una vez que se le termina el quantum a todos los procesos, y se encuentran todos en la cola de pasivos, se les asigna un nuevo quantum y pasan nuevamente a la cola de activos.

Aunque todos los procesos tienen una asignación equitativa de los recursos, el problema en este algoritmo es que no se premia al que libera la CPU antes.

Planificación mediante colas multinivel realimentadas (round robin)

El algoritmo de planificación mediante colas multinivel realimentadas per­mite mover un proceso de una cola a otra. La idea es separar los procesos en función de las carac­terísticas de sus ráfagas de CPU. Si un proceso utiliza demasiado tiempo de CPU, se pasa a una cola de prioridad más baja. Este esquema deja los procesos limitados por E/S y los procesos interacti­vos en las colas de prioridad más alta. Además, un proceso que esté esperando demasiado tiem­po en una cola de baja prioridad puede pasarse a una cola de prioridad más alta. Este mecanismo de envejecimiento evita el bloqueo indefinido.

Planificación mediante colas multinivel (para solucionar el problema de si usar batch o interactivo)

Otra clase de algoritmos de planificación es la que se ha desarrollado para aquellas situaciones en las que los procesos pueden clasificarse fácilmente en grupos diferentes. Por ejemplo, una clasifi­cación habitual consiste en diferenciar entre procesos de primer plano (interactivos) y procesos de segundo plano (batch). Estos dos tipos de procesos tienen requisitos diferentes de tiempo de respuesta y, por tanto, pueden tener distintas necesidades de planificación. Además, los procesos de primer plano pueden tener prioridad (definida externamente) sobre los procesos de segundo plano.

Un algoritmo de planificación mediante colas multinivel divide la cola de procesos prepara­dos en varias colas distintas. Los procesos se asignan permanentemente a una cola, generalmente en función de alguna propiedad del proceso, como por ejemplo el tamaño de memoria, la prioridad del proceso o el tipo de proceso. Cada cola tiene su propio algoritmo de pla­nificación. Por ejemplo, pueden emplearse colas distintas para los procesos de primer plano y de segundo plano. La cola de primer plano puede planificarse mediante un algoritmo por turnos, mientras que para la cola de segundo plano puede emplearse un algoritmo FCFS.

Además, debe definirse una planificación entre las colas, la cual suele implementarse como una planificación con desalojo y prioridad fija. Por ejemplo, la cola de procesos de primer plano puede tener prioridad absoluta sobre la cola de procesos de segundo plano.

Otra posibilidad consiste en repartir el tiempo entre las colas. En este caso, cada cola obtiene : una cierta porción del tiempo de CPU, con la que puede entonces planificar sus distintos procesos.

Planificación de sistemas multiprocesador

Vamos a concentrarnos en los sistemas en los que los procesado­res son idénticos, homogéneos en cuanto a su funcionalidad; en este tipo de sistemas, podemos usar cualquiera de los procesadores disponibles para ejecutar cualquier proceso de la cola. (Sin embargo, observe que incluso con múltiples procesadores homogéneos, en ocasiones hay limita­ciones que afectan a la planificación; considere un sistema con un dispositivo de E/S conectado a un bus privado de un procesador. Los procesos que deseen emplear este dispositivo deberán pla­nificarse para ser ejecutados en dicho procesador.)

Métodos de planificación en los sistemas multiprocesador

Un método para planificar las CPU en un sistema multiprocesador consiste en que todas las deci­siones sobre la planificación, el procesamiento de E/S y otras actividades del sistema sean gestio­nadas por un mismo procesador, el servidor maestro. Los demás procesadores sólo ejecutan código de usuario. Este multiprocesamiento asimétrico resulta simple, porque sólo hay un pro­cesador que accede a las estructuras de datos del sistema, reduciendo la necesidad de compartir datos.

Un segundo método utiliza el multiprocesamiento simétrico (SMP), en el que cada uno de los procesadores se auto-planifica. Todos los procesos pueden estar en una cola común de procesos preparados, o cada procesador puede tener su propia cola privada de procesos preparados. Independientemente de esto, la planificación se lleva a cabo haciendo que el planificador de cada procesador examine la cola de procesos preparados y seleccione un proceso para ejecutarlo.

Afinidad al procesador

Considere lo que ocurre con la memoria caché cuando un proceso se ha estado ejecutando en procesador específico: los datos a los que el proceso ha accedido más recientemente se almacenan en la caché del procesador y, como resultado, los sucesivos accesos a memoria por parte del proceso suelen satisfacerse sin más que consultar la memoria caché. Ahora considere lo que ocurre si el proceso migra a otro procesador: los contenidos de la memoria caché del procesador de origen deben invalidar y la caché del procesador de destino debe ser rellenada. Debido al alto coste de invalidar y rellenar las memorias caché, la mayoría de los sistemas SMP intentan evitar la migración de procesos de un procesador a otro, y en su lugar intentan mantener en ejecución cada proceso en el mismo procesador. Esto se conoce con el nombre de afinidad al procesador, lo que significa que un proceso tiene una afinidad hacia el procesador en que está ejecutándose actual­mente.

Equilibrado de carga

En los sistemas SMP, es importante mantener la carga de trabajo equilibrada entre todos los pro­cesadores, para aprovechar por completo las ventajas de disponer de más de un procesador. Si no se hiciera así, podría darse el caso de que uno o más procesadores permanecieran inactivos mien­tras otros procesadores tuvieran cargas de trabajo altas y una lista de procesos esperando a acce­der a la CPU. Los mecanismos de equilibrado de carga intentan distribuir equitativamente la carga de trabajo entre todos los procesadores del sistema SMP. Es importante observar que el equi­librado de carga, normalmente, sólo es necesario en aquellos sistemas en los que cada procesador tiene su propia cola privada de procesos preparados para ejecutarse. En los sistemas con una cola de ejecución común, el equilibrado de carga es a menudo innecesario, ya que una vez que un pro­cesador pasa a estar inactivo, inmediatamente extrae un proceso ejecutable de la cola de ejecución común. No obstante, también es importante destacar que en la mayoría de los sistemas operativos actuales que soportan SMP cada procesador tiene una cola privada de procesos preparados.

Existen dos métodos generales para equilibrar la carga: migración comandada (push migration) y migración solicitada (pull migration). Con la migración comandada, una tarea específica comprueba periódicamente la carga en cada procesador y, si encuentra un desequilibrio, distribu­ye equitativamente la carga moviendo (o cargando) procesos de los procesadores sobrecargados en los menos ocupados o inactivos. La migración solicitada se produce cuando un procesador inactivo extrae de un procesador ocupado alguna tarea que esté en espera. Las migraciones comandadas y solicitadas no tienen por qué ser mutuamente excluyentes y, de hecho, a menudo se implementan en paralelo en los sistemas de equilibrado de carga.

Es interesante comentar que el equilibrado de carga a menudo contrarresta los beneficios de la afinidad al procesador. Es decir, la ventaja de mantener un proceso eje­cutándose en el mismo procesador es que el proceso se aprovecha de que sus datos se encuentran en la memoria caché de dicho procesador. Al migrar procesos de un procesador a otro, anulamos esta ventaja.

SEGUNDO PARCIAL UNIDADES 3 Y 4

PAGINACIÓN

La memoria principal se divide en porciones de tamaño fijo relativamente pequeños, y que cada proceso también se divide en porciones pequeñas del mismo tamaño fijo. A dichas porciones de proceso, conocidas como páginas, se les asigna porciones disponibles de memoria, conocidas como marcos, o marcos de páginas.

El sistema operativo mantiene una tabla de páginas por cada proceso.

La tabla de páginas muestra la ubicación del marco por cada página del proceso. Dentro del programa, cada dirección lógica está formada por un número de página y un desplazamiento dentro de la página.

Una tabla de páginas contiene una entrada por cada página del proceso, de forma que la tabla se indexe fácilmente por el número de página (iniciando en la página 0). Cada entrada de la tabla de páginas contiene el número del marco en la memoria principal, si existe, que contiene la página correspondiente.

Inicialmente la tabla de páginas estaba en la MMU, pero al crecer la memoria se necesitaron más tablas de páginas y se creó la PTBR, esta tabla contiene la dirección base de cada tabla de página, la PTBR paso a estar en la MMU y las tablas de páginas en la memoria.

Adicionalmente, el sistema operativo mantiene una única lista de marcos libres de todos los marcos de la memoria que se encuentran actualmente no ocupados y disponibles para las páginas.

Al cambiar las tablas de páginas en memoria por la PTBR, quedan registros libres y en esos registros se pone la TLB, esta funciona como una memoria caché guardando las últimas páginas usadas con sus respectivas direcciones base.

Cuando se busca por la dirección física de una página la CPU lanza dos líneas de ejecución, una va a buscar a la memoria y otra va a buscar a la TLB, si la encuentra en la TLB ya sale con la dirección base, sino la tiene que calcular.

Cuando la memoria se agrandó mucho apareció la fragmentación externa del espacio de memoria que fue asignado para las tablas de páginas y estas ya no pueden asignarse de forma contigua. En ese momento se aplicó la paginación externa-interna donde la PTBR pasa a tener las direcciones de cada página externa de los procesos, y la página externa contiene las direcciones base de cada página interna del mismo proceso (la página interna sería una tabla de página en sí).

La PTBR y la TLB son requerimientos del hardware necesarios para la paginación simple.

Por tanto vemos que la paginación simple, tal y como se describe aquí, es similar al particionamiento fijo (MFT). Las diferencias son que, con la paginación, las particiones son bastante pequeñas; un programa podría ocupar más de una partición; y dichas particiones no necesitan ser contiguas.

SEGMENTACIÓN

La segmentación de memoria es una técnica de gestión de memoria que pretende acercarse más al punto de vista del programador.

Desde este punto de vista, un programa es un conjunto de componentes lógicos de tamaño variable o un conjunto de segmentos, es decir, el espacio lógico de direcciones se considera como un conjunto de segmentos, cada uno definido por un identificador, y consistente de un punto de inicio y el tamaño asignado.

La segmentación de un programa la realiza el compilador y en ella cada dirección lógica se expresará mediante dos valores: Número de segmento y desplazamiento dentro del segmento.

Esta técnica permite reducir la fragmentación interna de la memoria provocada por la paginación, ya que asigna a cada programa la cantidad de memoria que requiere.

La carga de un programa en memoria exige la búsqueda de los huecos adecuados a sus segmentos, y puesto que éstos son de tamaño variable, se ajustarán los más posible a las necesidades, produciéndose huecos pequeños. En este caso se produce fragmentación externa.

MEMORIA VIRTUAL

Técnica que permite la ejecución de procesos cuyo código no está totalmente en la memoria real, el beneficio de esto es poder escribir código sin tener el límite de la memoria física.

Cargador perezoso: carga en memoria las páginas que necesita y luego va cargando el resto bajo demanda. La ejecución es más lenta pero el arranque es más rápido. Se pueden ejecutar procesos con mayor demanda de la que realmente se tiene.

Cargador completo: Carga todas las páginas en memoria. Con memoria virtual se le da a un proceso la cantidad de direcciones que el compilador dice, pero el espacio de memoria que se le da es menor, el resto del proceso se guarda en la memoria virtual (a través de swapping se manda al disco duro).

Selección de página victima:

1. **FiFo:** las primeras páginas cargadas son las que primero se eliminan.

Ventajas: es un algoritmo conocido.

Desventaja: no se puede asegurar que las primeras páginas cargadas no se estén usando. Anomalía de Belady: aumentar la cantidad de marcos no asegura que sea más eficiente.

Se puede utilizar un contador donde se elimina la pagina con el número más chico ó guardar la hora en la que fue cargada la página, pero esto no es muy eficiente.

1. **Óptimo (MTFVU):** en este caso se reemplazan las páginas que no van a ser usadas en mucho tiempo, pero esto no es implementable porque no tengo certeza sobre lo que va a pasar en el futuro. Este método sirve como marco de referencia.
2. **LRU (least recently used):** se elige la página que lleva mayor tiempo sin ser usada. Basado en el óptimo. Tiene un **contador** (reloj): Cada vez que accedo a una página copio el valor del clock en ese momento. Saco el que tiene el mayor tiempo:

+No sufre la anomalía de Belady.

+Es un algoritmo de **pila** yesta cacheada en nivel procesador para que sea mas rapida

-En propósito general no sirve por tener una pila muy grande en la caché, en cambio se usa esto:

1. **Aproximación LRU**: Se mantienen 8 bits para cada página en una tabla de memoria. Estos registros sirven como historial de uso de página, cada tanto tiempo se ejecuta un shift (rutina de CPU para desplazar los bits). La página con el número más bajo se puede sustituir por ser la menos recientemente usada.
2. **Algoritmo de segunda oportunidad** ( o reloj): Cola circular. Se mira el bit de referencia:

Las paginas que estan en 1 pasan a 0 teniendo segunda chance y no se sustituye hasta que las demás hayan sido sustituidas o hayan tenido segunda chance. Si la página es usada su bit de referencia pasa a ser 1. El puntero baja. Es muy usado, muy simple y suficientemente eficiente.

**Evitar fallo de páginas:**

Para evitar el fallo de página, la MMU puede:

1. Sacrificar páginas que no se estén usando para tener marcos libres.
2. Pre cargar: carga páginas que supone que van a ser usadas, sin que la CPU se lo pida, entonces cuando la CPU pide que cargue esas páginas, estas ya están en memoria.

Generalmente, se sacrifican las páginas que nunca se usaron con la pre carga.

**Asignación de marcos:**

**Equitativa**: se reparten m marcos entre n procesos de forma equitativa.

**Proporcional**: se asigna la memoria disponible dependiendo el tamaño del proceso.

**Política de sacrificio o sustitución:**

**Local**: solo se efectúa la selección de marco entre su propio conjunto de marcos asignados.

**Global**: permite a un proceso obtener un marco del conjunto de todos los marcos, incluso si ese marco está asignado a otro proceso (un proceso le puede quitar un marco a otro).

**Trashing** (sobrepaginación):

Se cae en trashing cuando se dedica más tiempo a la sustitución de páginas que al procesamiento de datos.

Ocurre al hacer nacer demasiados procesos con muchas páginas asociadas, por lo que cada proceso tiene pocas páginas cargadas en RAM. Hay un gran porcentaje de fallo de página.

Causas:

1. **Usuario**. Solución: capacitación
2. **ERROR** de codificación. Solución: control de código?
3. **PMP**: Planificador de mediano plazo ( “como no hay muchos procesos pongo a cargar más”. Se sacrifican páginas. quedan muchos procesos con pocas pags y alto % de fallos de página) de una misma página.

PFF(modelo de porcentaje de fallos): Si un proceso supera el 3% de fallos de página le tengo que dar mas marcos y sino lo mando a swap si falla mucho->si consigo mas marcos lo traigo de nuevo.

No se sacrifican aquellas rutinas que sé que voy a utilizar. Ni las páginas de rutinas

ADMINISTRACION DE PERIFERICOS

1. Almacenamiento: como pendrive, memorias secundarias. Estos deben tener memoria ‘permanente’, es decir, que almacene datos por cierto tiempo.
2. Salida: como impresoras, monitores. Son lentos, se manejan a través de los buses y no usan mecanismos bloqueantes, sino que usan interrupciones.
3. Comunicación y control: como puertos y redes.

Generalmente los drivers de los periféricos los hacían los fabricantes pero después de los 90 se hicieron un conjunto de normas para hacerlos estándar y evitar problemas tales como el colapso de los drivers, por ejemplo por usar los mismos recursos.

FILESYSTEM

Cada sistema operativo soporta por lo menos un filesystem, estos generalmente guardan del archivo el id, nombre, version, ubicación, tamaño, tipo, fecha, dueño, permisos, etc, esos datos que guarda dependen de sus necesidades de organización y del servicio que quiera dar.

Un filesystem tiene diferentes acciones de archivo: leer, escribir, modificar, crear, borrar (estas son las operaciones básicas).

El poder de un filesystem depende de la cantidad de acciones que puede realizar sobre un archivo.

Al principio el SO se comunicaba directamente con el filesystem, luego se creó un filesystem interface (FSI) la cual se comunicaba con los otros filesystems. Al crecer la cantidad de filesystems, esta comunicación se volvió muy complicada y por lo tanto nació el virtual filesystem (VFS) donde cada periférico trae consigo un driver que sabe comunicarse con el virtual filesystem, y este es el que se comunica con la interface.

En el arranque se levantan los filesystems de los periféricos (si están en el fstab).

El filesystem administra:

1. El espacio de los archivos:
2. Asignación contigua: como el CD de música donde ya se conoce el espacio que ocupa el archivo, no admite acceso directo ni cambiar el tamaño de los archivos. Es muy utilizado para backups por ejemplo.
3. Asignación encadenada: permite que un archivo cambie el tamaño. Puede ser simplemente encadenada o doblemente encadenada. Esta forma es lenta cuando el archivo se recorre a través de acceso directo.

Fat Allocation System:

* Reemplaza a la asignación encadenada.
* Es una tabla cargada en memoria, donde cada columna le corresponde a un archivo y tiene tantas entradas como sectores haya.
* Es la misma para todos los periféricos.
* Es más rápido que la lectura convencional ya que la búsqueda es en la RAM, y es es electrónica, no mecánica.

1. Indexado: existen dos tipos: simple y combinado. Hay una estructura que tiene los datos del inodo en las primeras posiciones y en el resto tiene punteros, cuando tiene hasta 12 punteros es un indexado simple, o sea que esos punteros apuntan directamente a sectores en el disco duro. Cuando la estructura tiene los punteros 13, 14 y 15 ya es un indexado combinado, porque esos punteros apuntan a sectores que tienen punteros. El puntero 13 tiene un nivel de indirección, el 14 tiene dos niveles de indirección y el nivel 15 tiene tres niveles de indirección.
2. El espacio libre:
3. Bits array: se guarda un vector de bits en un sector, 0 para bit libre y 1 para espacio ocupado. Este método es para dispositivos pequeños, porque para dispositivos de almacenamiento muy grandes el arreglo se vuelve muy grande y es ineficiente. Ejemplos: disquetes, discos de 10M, floppys.
4. Encadenado: en este caso contamos con un puntero al proximo sector libre, y cada sector tiene un puntero al próximo. Funciona muy bien en la FAT porque trabaja en el mismo formato.
5. Agrupar: también es una lista encadenada. Cuando se crea un archivo puede que se sepa el tamaño del archivo, pero si no se sabe se le asigna una pre cantidad de sectores y cuando necesita más se le van asignando. (antes se le pre asignaba uno solo y se iba sumando de a 1 sector)

Ventaja: el archivo no se encuentra desparramado sino que tiene un conjunto de sectores agrupados de forma contigua.

El sector que tiene los datos del inodo y los punteros ahora tiene un nuevo campo que se llama cantidad libre de conteo, que tiene la cantidad de sectores que ocupa un archivo, la ventaja de esto es que cuando borras un archivo el SO sabe la cantidad contigua de espacio de disco que tendrá.

ARCHIVOS

Un archivo informático es una estructura contenedora e identificable de bits que normalmente son datos, donde el nombre es el identificador.

Tipos de archivos:

1. Manejados por aplicaciones: el so tiene una tabla donde asocia la extensión del archivo con la aplicación que lo maneja.
2. Manejados por el SO:
3. Tipo texto: son archivos con cierta codificación legible el humano. Son definidos por los caracteres de fin de párrafo y el de retorno de línea, dependiendo el filesystem pueden estar los dos o uno solo. Se leen secuencialmente byte por byte comparándolos contra la tabla ASCII.
4. Tipo binario: son del tipo ejecutable, no son entendibles por el humano. El SO hace una lectura secuencial pero lee todo como archivo unitario y recien ahi lo sube a memoria.
5. Tipo fichero: se utilizan para guardar datos, son registros que en cada campo contienen tipos de datos diferentes, existen de longitud variable y fija, los de longitud variable se trabajan con punteros pero el SO interpreta todos los campos como si fueran de longitud fija.

Hay dos tipos de registros:

1. Físicos: están definidos por el filesystem, y deben tener no menos de ‘x’ cantidad de bytes.
2. Lógicos: están definidos por el programador. Factor de bloqueamiento: cuántos registros lógicos entran en uno físico (estos deberían ser potencia de 2).

Los filesystems que tienen archivos de tipo fichero soportan dos formas de acceso directo: relativo y hash.

**Formas de apertura de archivo:**

**Open file:** lee dirección, asigna buffer cache de disco para trabajar: si nos sobraba ram, lo cargamos ahí.

**MMAP:** es más moderno, con los sistemas actuales tenemos RAM de sobra: se abren los archivos y se cargan en la RAM

**Ventaja:** la CPU no se bloquea por la system call para ir a leer el archivo por veces siguientes. Si no entra todo el archivo se levanta de a partes (concepto de memoria virtual).

DIRECTORIOS

Un directorio es un archivo donde se guardan las direcciones de otros archivos, la estructura lógica que se usa es de grafos o árbol.

Árbol jerárquico general: crea un árbol raíz y subárboles.

MNT: subdirectorio donde se encuentran los directorios de los periféricos.

PATH: es el camino que hay recorrer para llegar a un archivo, el nombre completo de un archivo incluye el PATH, el ‘.’ o ‘..’ se denominan PATH relativos.

RAID

RAID (conjunto redundante de discos Independiente), hace referencia a un sistema de almacenamiento de datos en tiempo real que utiliza múltiples unidades de almacenamiento de datos (discos) entre los que se distribuyen o replican los datos.

Dependiendo de su configuración (nivel), los beneficios de un RAID respecto a un único disco son uno o varios de los siguientes: mayor integridad, mayor tolerancia a fallos, mayor rendimiento y mayor capacidad.

Niveles:

* **RAID 0**: distribuye los datos equitativamente (a nivel de bloques) entre dos o más discos sin proporcionar redundancia alguna. (AID)
* **RAID 1**: crea una copia exacta (o espejo) de un conjunto de datos en dos o más discos. Esto resulta útil cuando queremos tener más seguridad desaprovechando capacidad, ya **RAID 4**: Este tipo de RAID, básicamente, divide los datos entre los discos, siendo uno de esos discos exclusivo para paridad. La diferencia entre el nivel 4 y el nivel 3, es que en caso de falla de uno de los discos, los datos pueden ser reconstruidos en tiempo real a través de la utilización de la paridad calculada a partir de los otros discos, siendo que cada uno puede ser accedido de forma independiente.
* **RAID 5**: Este es muy semejante al nivel 4, excepto por el hecho de que la paridad no está destinada a un único disco, sino a toda la matriz. Eso hace que la grabación de datos sea más rápida, pues no es necesario acceder a un disco de paridad en cada grabación.
* **RAID 6**: amplía el nivel RAID 5 añadiendo otro bloque de paridad, por lo que divide los datos a nivel de bloques y distribuye los dos bloques de paridad entre todos los miembros del conjunto. Esto ayuda a la hora de múltiples fallos de disco.
* RAID 0: distribuye los datos equitativamente (a nivel de bloques) entre dos o más proporcionar redundancia alguna.
* RAID 1: crea una copia exacta (o espejo) de un conjunto de datos en dos o más discos. Esto resulta útil cuando queremos tener más seguridad desaprovechando capacidad, ya que si perdemos un disco, tenemos el otro con la misma información.
* RAID 2: usa división a nivel de bits con un disco de paridad dedicado y usa un código para la corrección de errores (ECC). Este tipo de RAID, adapta el mecanismo de detección de fallas en discos rígidos para funcionar en memoria. Así, todos los discos de la matriz están siendo "monitorizados" por el mecanismo.
* RAID 3: En este nivel, los datos son divididos entre los discos de la matriz, excepto uno, que almacena información de paridad. Así, todos los bytes de los datos tienen su paridad (aumento de 1 bit, que permite identificar errores) almacenada en un disco específico. A través de la verificación de esta información, es posible asegurar la integridad de los datos.
* RAID 4: Este tipo de RAID, básicamente, divide los datos entre los discos, siendo uno de esos discos exclusivo para paridad. La diferencia entre el nivel 4 y el nivel 3, es que en caso de falla de uno de los discos, los datos pueden ser reconstruidos en tiempo real a través de la utilización de la paridad calculada a partir de los otros discos, siendo que cada uno puede ser accedido de forma independiente.
* RAID 5: Este es muy semejante al nivel 4, excepto por el hecho de que la paridad no está destinada a un único disco, sino a toda la matriz. Eso hace que la grabación de datos sea más rápida, pues no es necesario acceder a un disco de paridad en cada grabación.
* RAID 6: amplía el nivel RAID 5 añadiendo otro bloque de paridad, por lo que divide los datos a nivel de bloques y distribuye los dos bloques de paridad entre todos los miembros del conjunto. Esto ayuda a la hora de múltiples fallos de disco.
* RAID 0+1: es una combinación de los niveles 0 (Striping) y 1 (Mirroring), donde los datos son divididos entre los discos para mejorar el ingreso, pero también utilizan otros discos para duplicar la información. Así, es posible utilizar el buen ingreso del nivel 0 con la redundancia del nivel 1. Sin embargo, es necesario por lo menos 4 discos para montar un RAID de este tipo. Estas características hacen del RAID 0 + 1 el más rápido y seguro, sin embargo es el más caro de ser implementado.
* RAID 1+0: a veces llamado RAID 10, es parecido a un RAID 0+1 con la excepción de que los niveles RAID que lo forman se invierte: el RAID 10 es una división de espejos.

Fin, suerte mañana.