UADER | FCyT

Sistemas Operativos

Procesos

Clase 5 - Unidad II

Lic. Alexis Sostersich
sostersich.alexis@uader.edu.ar

Teoría – Sistemas Operativos

Procesos

Unidad II – Clase 5

- Planificación
  - Comportamiento de un proceso, Cuándo planificar procesos, Categorías de los algoritmos de planificación, Metas de los algoritmos de planificación
- Planificación en sistemas de procesamiento por lotes
  - Primero en entrar, primero en ser atendido, El trabajo más corto primero, El menor tiempo restante a continuación
- Planificación en sistemas interactivos
  - Planificación por turno circular; Planificación por prioridad; Múltiples colas; El proceso más corto a continuación; Planificación garantizada; Planificación por sorteo; Planificación por partes equitativas
- Planificación en sistemas de tiempo real

Planificación

Planificación

Cuando una computadora se multiprograma, **con frecuencia tiene varios procesos o hilos que compiten por la CPU al mismo tiempo.** Esta situación ocurre cada vez que dos o más de estos procesos se encuentran al mismo tiempo en el estado listo. Si sólo hay una CPU disponible, hay que decidir cuál proceso se va a ejecutar a continuación. La parte del sistema operativo que realiza esa decisión se conoce como planificador de procesos y el algoritmo que utiliza se conoce como algoritmo de planificación.

Muchas de las mismas cuestiones que se aplican a la planificación de procesos también se aplican a la planificación de hilos, aunque algunas son distintas. **Cuando el kernel administra hilos, por lo general la planificación se lleva a cabo por hilo, e importa muy poco (o nada) a cuál proceso pertenece ese hilo.**

Introducción a la planificación

En los días de los sistemas de procesamiento por lotes, con entrada en forma de imágenes de tarjetas en una cinta magnética, el algoritmo de planificación era simple: bastaba con ejecutar el siguiente trabajo en la cinta.

Con los sistemas de multiprogramación, el algoritmo de planificación se volvió más complejo debido a que comúnmente había varios usuarios esperando ser atendidos.

Algunas mainframe todavía combinan los servicios de procesamiento por lotes y de tiempo compartido, requiriendo que el planificador decida si le toca el turno a un trabajo de procesamiento por lotes o a un usuario interactivo en una terminal.

Introducción a la planificación

Debido a que el tiempo de la CPU es un recurso escaso en estas máquinas, un buen planificador puede hacer una gran diferencia en el rendimiento percibido y la satisfacción del usuario. En consecuencia, se ha puesto gran esfuerzo en idear algoritmos de planificación astutos y eficientes.

Con la llegada de las computadoras personales, la situación cambió de dos maneras. En primer lugar, la mayor parte del tiempo sólo hay un proceso activo. Es muy poco probable que un usuario que entra a un documento en un procesador de palabras compile al mismo tiempo un programa en segundo plano. Cuando el usuario escribe un comando para el procesador de palabras, el planificador no tiene que esforzarse mucho en averiguar qué proceso ejecutar; el procesador de palabras es el único candidato.

Introducción a la planificación

En segundo lugar, **las computadoras se han vuelto tan veloces con el paso de los años que la CPU es raras veces un recurso escaso**. La mayoría de los programas para computadoras personales están limitados en cuanto a la velocidad a la que el usuario puede presentar los datos de entrada (al escribir o hacer clic), no por la velocidad con que la CPU puede procesarlos. Incluso las compilaciones, que en el pasado consumían muchos ciclos de la CPU, requieren unos cuantos segundos en la mayoría de los casos hoy en día.

Introducción a la planificación

Aún y cuando se estén ejecutando dos programas al mismo tiempo, como un procesador de palabras y una hoja de cálculo, raras veces es importante saber quién se ejecuta primero, ya que el usuario probablemente está esperando a que ambos terminen. Como consecuencia, la planificación no es de mucha importancia en las PCs simples.

Desde luego que hay aplicaciones que prácticamente se comen viva a la CPU; por ejemplo, para visualizar una hora de video en alta resolución, a la vez que se ajustan los colores en cada uno de los 108,000 cuadros (en NTSC) o 90,000 cuadros (en PAL) se requiere de un poder de cómputo a nivel industrial. Sin embargo, las aplicaciones similares son la excepción, en vez de la regla.

Introducción a la planificación

Al tratarse de servidores en red, la situación cambia en forma considerable. Aquí varios procesos compiten a menudo por la CPU, por lo que la planificación retoma importancia.

Además de elegir el proceso correcto que se va a ejecutar a continuación, el planificador también tiene que preocuparse por hacer un uso eficiente de la CPU, debido a que la conmutación de procesos es cara. Para empezar, debe haber un cambio del modo de usuario al modo kernel. Después se debe guardar el estado del proceso actual, incluyendo el almacenamiento de sus registros en la tabla de procesos para que puedan volver a cargarse más adelante.

Introducción a la planificación

En muchos sistemas, el mapa de memoria se debe guardar también. Luego hay que seleccionar un nuevo proceso mediante la ejecución del algoritmo de planificación. Después de eso, se debe volver a cargar en la MMU el mapa de memoria del nuevo proceso.

Por último, se debe iniciar el nuevo proceso. Además de todo esto, generalmente la conmutación de procesos hace inválida toda la memoria caché, por lo que tiene que volver a cargarse en forma dinámica desde la memoria principal dos veces (al momento de entrar al kernel y al salir de éste).

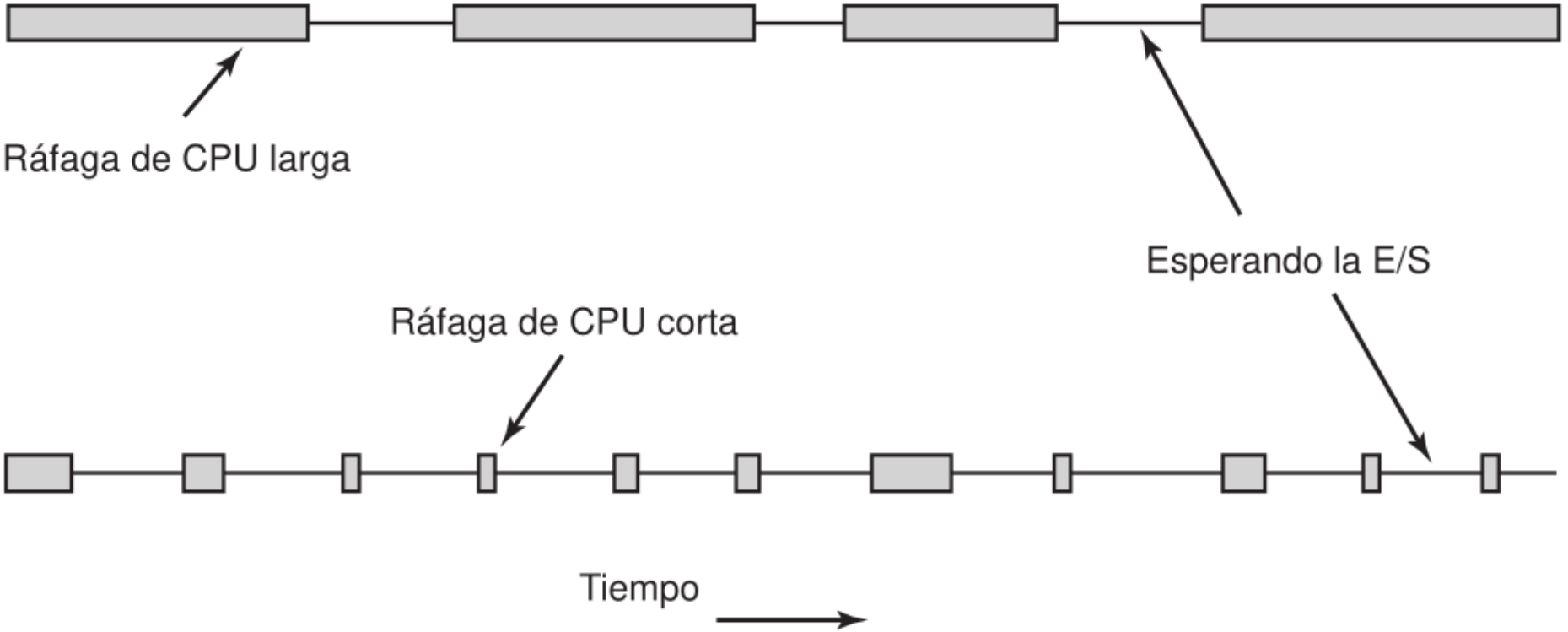
Con todo, **si se realizan muchas conmutaciones de procesos por segundo, se puede llegar a consumir una cantidad considerable de tiempo de la CPU, por lo cual se aconseja tener precaución.**

Comportamiento de un proceso

Casi todos los procesos alternan ráfagas de cálculos con peticiones de E/S (de disco). **Por lo general la CPU opera durante cierto tiempo sin detenerse, después se realiza una llamada al sistema para leer datos de un archivo o escribirlos en el mismo.** Cuando se completa la llamada al sistema, la CPU realiza cálculos de nuevo hasta que necesita más datos o tiene que escribir más datos y así sucesivamente. Hay que tener en cuenta que algunas actividades de E/S cuentan como cálculos. Por ejemplo, cuando la CPU copia bits a una RAM de video para actualizar la pantalla, está calculando y no realizando operaciones de E/S, ya que la CPU está en uso.

En este sentido, la E/S es cuando un proceso entra al estado bloqueado en espera de que un dispositivo externo complete su trabajo.

Comportamiento de un proceso



Comportamiento de un proceso

Algunos procesos, invierten la mayor parte de su tiempo realizando cálculos, mientras que otros invierten la mayor parte de su tiempo esperando la E/S. A los primeros se les conoce como limitados a cálculos; a los segundos como limitados a E/S (I/O-bound). Por lo general, los procesos limitados a cálculos tienen ráfagas de CPU largas y en consecuencia, esperas infrecuentes por la E/S, mientras que los procesos limitados a E/S tienen ráfagas de CPU cortas y por ende, esperas frecuentes por la E/S.

El factor clave es la longitud de la ráfaga de CPU, no de la ráfaga de E/S. Los procesos limitados a E/S están limitados a la E/S debido a que no realizan muchos cálculos entre una petición de E/S y otra, no debido a que tengan peticiones de E/S en especial largas. **Se requiere el mismo tiempo para emitir la petición de hardware para leer un bloque de disco**, sin importar qué tanto (o qué tan poco) tiempo se requiera para procesar los datos, una vez que lleguen.

Cuándo planificar procesos

El planificador de procesos tiene que tomar acción:

- **Cuando se crea un nuevo proceso** se debe tomar una decisión en cuanto a si se debe ejecutar el proceso padre o el proceso hijo
- **Cuando un proceso termina**
- **Cuando un proceso se bloquea** por esperar una operación de E/S, un semáforo o por alguna otra razón, hay que elegir otro proceso para ejecutarlo
- **Cuando ocurre una interrupción de E/S** tal vez haya que tomar una decisión de planificación

Algoritmos de planificación

Los algoritmos de planificación se pueden dividir en dos categorías con respecto a la forma en que manejan las interrupciones del reloj.

Un algoritmo de programación no apropiativo (nonpreemptive) selecciona un proceso para ejecutarlo y después sólo deja que se ejecute hasta que el mismo se bloquea (ya sea en espera de una operación de E/S o de algún otro proceso) o hasta que libera la CPU en forma voluntaria. Incluso aunque se ejecute durante horas, no se suspenderá de manera forzosa.

En efecto, no se toman decisiones de planificación durante las interrupciones de reloj. Una vez que se haya completado el procesamiento de la interrupción de reloj, se reanuda la ejecución del proceso que estaba antes de la interrupción, a menos que un proceso de mayor prioridad esté esperando por un tiempo libre que se acabe de cumplir.

Algoritmos de planificación

Por el contrario, **un algoritmo de planificación apropiativa selecciona un proceso y deja que se ejecute por un máximo de tiempo fijo.**

Si sigue en ejecución al final del intervalo de tiempo, se suspende y el planificador selecciona otro proceso para ejecutarlo (si hay uno disponible).

Para llevar a cabo la planificación apropiativa, es necesario que ocurra una interrupción de reloj al final del intervalo de tiempo para que la CPU regrese el control al planificador. Si no hay un reloj disponible, la planificación no apropiativa es la única opción.

Categorías de los algoritmos de planificación

Distintos entornos requieran algoritmos de planificación diferentes.

Esta situación se presenta debido a que las diferentes áreas de aplicación (y los distintos tipos de sistemas operativos) tienen diferentes objetivos. En otras palabras, **lo que el planificador debe optimizar no es lo mismo en todos los sistemas.**

Tres de los entornos que vale la pena mencionar son:

1. Procesamiento por lotes.
2. Interactivo.
3. De tiempo real.

Categorías de los algoritmos de planificación

Los sistemas de procesamiento por lotes siguen utilizándose ampliamente en el mundo de los negocios donde no hay usuarios que esperen impacientemente en sus terminales para obtener una respuesta rápida a una petición corta.

Por lo que, son aceptables los algoritmos no apropiativos (o apropiativos con largos periodos para cada proceso). Este método reduce la conmutación de procesos y por ende, mejora el rendimiento. En realidad, los algoritmos de procesamiento por lotes son bastante generales.

Categorías de los algoritmos de planificación

En un entorno con usuarios interactivos, la apropiación es esencial para evitar que un proceso acapare la CPU y niegue el servicio a los demás. Aun si no hubiera un proceso que se ejecutara indefinidamente de manera intencional, podría haber un proceso que deshabilitara a los demás de manera indefinida, debido a un error en el programa.

La apropiación es necesaria para evitar este comportamiento. Los servidores también entran en esta categoría, ya que por lo general dan servicio a varios usuarios (remotos), todos los cuales siempre tienen mucha prisa.

Categorías de los algoritmos de planificación

En los sistemas con restricciones de tiempo real, aunque parezca extraño, la apropiación a veces es no necesaria debido a que los procesos saben que no se pueden ejecutar durante periodos extensos, que por lo general realizan su trabajo y se bloquean con rapidez.

La diferencia con los sistemas interactivos es que los sistemas de tiempo real sólo ejecutan programas destinados para ampliar la aplicación en cuestión. Los sistemas interactivos son de propósito general y pueden ejecutar programas arbitrarios que no sean cooperativos, o incluso malintencionados.

Metas de los algoritmos de planificación

Para poder diseñar un algoritmo de programación, es necesario tener cierta idea de lo que debe hacer un buen algoritmo. Algunos objetivos dependen del entorno (procesamiento por lotes, interactivo o de tiempo real), pero hay también algunos otros que son deseables en todos los casos.

Vemos aquí algunas metas:

Todos los sistemas

Equidad - Otorgar a cada proceso una parte justa de la CPU

Aplicación de políticas - Verificar que se lleven a cabo las políticas establecidas

Balance - Mantener ocupadas todas las partes del sistema

Metas de los algoritmos de planificación

Sistemas de procesamiento por lotes

- **Rendimiento** - Maximizar el número de trabajos por hora
- **Tiempo de retorno** - Minimizar el tiempo entre la entrega y la terminación
- **Utilización de la CPU** - Mantener ocupada la CPU todo el tiempo

Sistemas interactivos

- **Tiempo de respuesta** - Responder a las peticiones con rapidez
- **Proporcionalidad** - Cumplir las expectativas de los usuarios

Sistemas de tiempo real

- Cumplir con los plazos - Evitar perder datos
- **Predictibilidad** - Evitar la degradación de la calidad en los sistemas multimedia

Planificación en sistemas de procesamiento por lotes

Primero en entrar, primero en ser atendido

Probablemente el más simple de todos los algoritmos de planificación es el de tipo primero en entrar, primero en ser atendido (FCFS, First-Come, First-Served) no apropiativo. Con este algoritmo, la CPU se asigna a los procesos en el orden en el que la solicitan. En esencia hay una sola cola de procesos listos. Cuando el primer trabajo entra al sistema desde el exterior en la mañana, se inicia de inmediato y se le permite ejecutarse todo el tiempo que desee. No se interrumpe debido a que se ha ejecutado demasiado tiempo. A medida que van entrando otros trabajos, se colocan al final de la cola. Si el proceso en ejecución se bloquea, el primer proceso en la cola se ejecuta a continuación. Cuando un proceso bloqueado pasa al estado listo, al igual que un trabajo recién llegado, se coloca al final de la cola.

Primero en entrar, primero en ser atendido

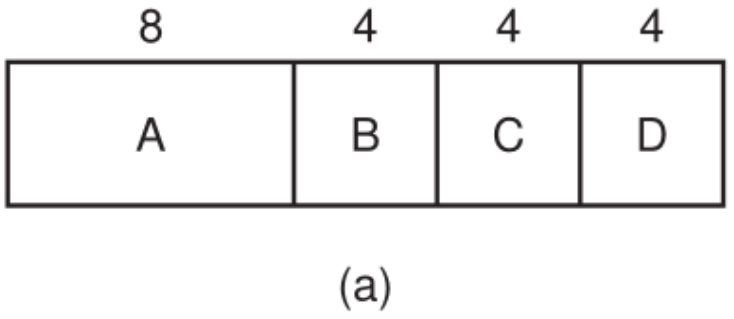
La gran fuerza de este algoritmo es que es fácil de comprender e igualmente sencillo de programar. También es equitativo. Con este algoritmo, una sola lista ligada lleva la cuenta de todos los procesos listos. Para elegir un proceso a ejecutar sólo se requiere eliminar uno de la parte frontal de la cola. Para agregar un nuevo trabajo o desbloquear un proceso sólo hay que adjuntarlo a la parte final de la cola.

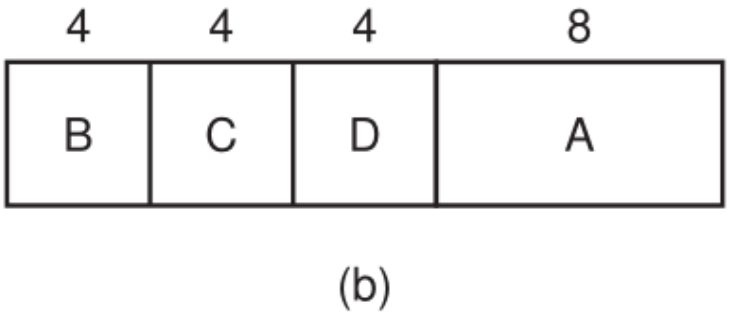
El trabajo más corto primero

Otro algoritmo de procesamiento por lotes no apropiativo, que supone que los tiempos de ejecución se conocen de antemano.

Cuando hay varios trabajos de igual importancia esperando a ser iniciados en la cola de entrada, el planificador selecciona el trabajo más corto primero (SJF, Shortest Job First).

Observemos que el trabajo más corto primero es sólo óptimo cuando todos los trabajos están disponibles al mismo tiempo.





El menor tiempo restante a continuación

Una versión apropiativa del algoritmo tipo el trabajo más corto primero es el menor tiempo restante a continuación (SRTN, Shortest Remaining Time Next).

Con este algoritmo, el planificador siempre selecciona el proceso cuyo tiempo restante de ejecución sea el más corto.

De nuevo, se debe conocer el tiempo de ejecución de antemano. Cuando llega un nuevo trabajo, su tiempo total se compara con el tiempo restante del proceso actual. Si el nuevo trabajo necesita menos tiempo para terminar que el proceso actual, éste se suspende y el nuevo trabajo se inicia. Ese esquema permite que los trabajos cortos nuevos obtengan un buen servicio.

Planificación en sistemas interactivos

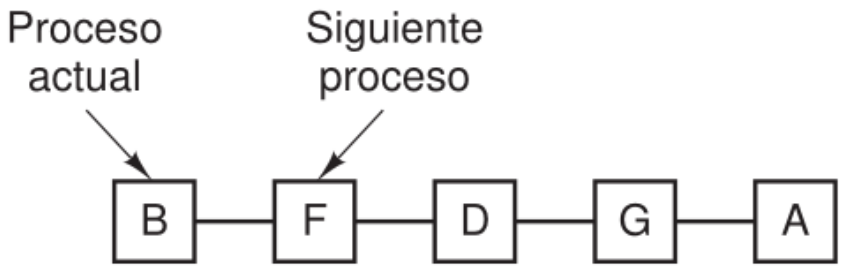
Planificación por turno circular

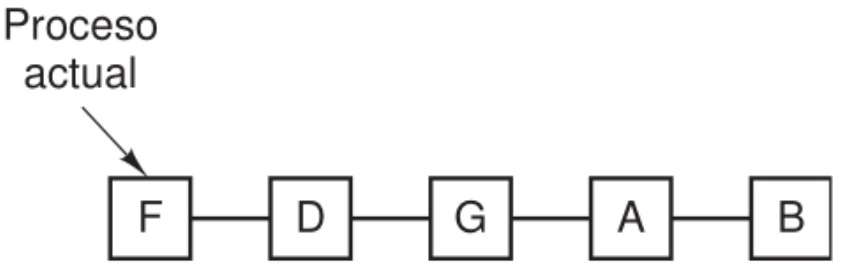
Uno de los algoritmos más antiguos, simples, equitativos y de mayor uso es el de turno circular (round-robin). A cada proceso se le asigna un intervalo de tiempo, conocido como quántum, durante el cual se le permite ejecutarse. Si el proceso se sigue ejecutando al final del cuanto, la CPU es apropiada para dársela a otro proceso. Si el proceso se bloquea o termina antes de que haya transcurrido el quántum, la conmutación de la CPU se realiza cuando el proceso se bloquea, desde luego. Es fácil implementar el algoritmo de turno circular. Todo lo se necesita es mantener una lista de procesos ejecutables, cuando el proceso utiliza su quántum, pasa al final de la lista.

Planificación por turno circular

La única cuestión interesante con el algoritmo de turno circular es la longitud del quántum. Para conmutar de un proceso a otro (conmutación de proceso o conmutación de contexto) se requiere cierta cantidad de tiempo para realizar la administración: guardar y cargar tanto registros como mapas de memoria, actualizar varias tablas y listas, vaciar y recargar la memoria caché y así sucesivamente.

Planificación por turno circular





Planificación por prioridad

La planificación por turno circular hace la suposición implícita de que todos los procesos tienen igual importancia. Con frecuencia, las personas que poseen y operan computadoras multiusuario tienen diferentes ideas en cuanto a ese aspecto.

La necesidad de tomar en cuenta los factores externos nos lleva a la planificación por prioridad. La idea básica es simple: a cada proceso se le asigna una prioridad y el proceso ejecutable con la prioridad más alta es el que se puede ejecutar.

Planificación por prioridad

Para evitar que los procesos con alta prioridad se ejecuten de manera indefinida, el planificador puede reducir la prioridad del proceso actual en ejecución en cada pulso del reloj (es decir, en cada interrupción del reloj). Si esta acción hace que su prioridad se reduzca a un valor menor que la del proceso con la siguiente prioridad más alta, ocurre una conmutación de procesos. De manera alternativa, a cada proceso se le puede asignar un quántum de tiempo máximo que tiene permitido ejecutarse. Cuando este quántum se utiliza, el siguiente proceso con la prioridad más alta recibe la oportunidad de ejecutarse.

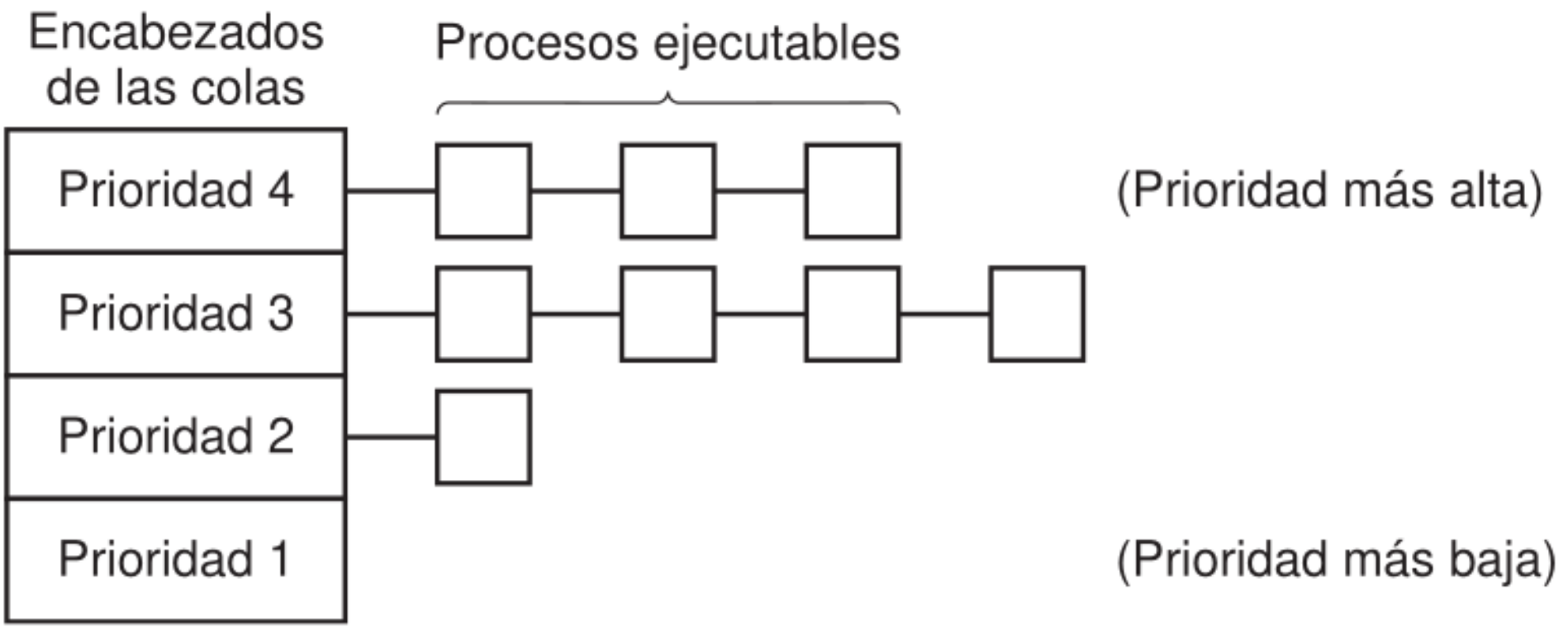
Múltiples colas

Uno de los primeros planificadores por prioridad estaba en CTSS, el Sistema de tiempo compartido compatible del M.I.T. que se ejecutaba en la IBM 7094 (Corbató y colaboradores, 1962). CTSS tenía el problema de que la conmutación de procesos era muy lenta, debido a que la 7094 sólo podía contener un proceso en la memoria. Cada conmutación de procesos ocasionaba intercambiar (swapping) el proceso actual al disco y leer uno nuevo del disco. Los diseñadores de CTSS se dieron cuenta rápidamente de que era más eficiente dar a los procesos limitados a CPU un quántum largo de vez en cuando, en vez de darles quántums pequeños con frecuencia (para reducir el intercambio). Por otro lado, al proporcionar a todos los procesos un quántum largo se obtendría un tiempo de respuesta pobre. **Su solución fue la de establecer clases de prioridades.**

Múltiples colas

Los procesos en la clase más alta se ejecutaban durante un quántum. Los procesos en la siguiente clase más alta se ejecutaban por dos quántums. Los procesos en la siguiente clase se ejecutaban por cuatro quántums, y así sucesivamente. Cada vez que un proceso utilizaba todos los quántums que tenía asignados, se movía una clase hacia abajo en la jerarquía.

Múltiples colas



El proceso más corto a continuación

Como el algoritmo tipo el trabajo más corto primero siempre produce el tiempo de respuesta promedio mínimo para los sistemas de procesamiento por lotes, sería bueno si se pudiera utilizar para los procesos interactivos también. Hasta cierto grado, esto es posible. Por lo general, los procesos interactivos siguen el patrón de esperar un comando, ejecutarlo, esperar un comando, ejecutarlo, etcétera.

Si consideramos la ejecución de cada comando como un “trabajo” separado, entonces podríamos minimizar el tiempo de respuesta total mediante la ejecución del más corto primero. El único problema es averiguar cuál de los procesos actuales ejecutables es el más corto.

Un método es realizar estimaciones con base en el comportamiento anterior y ejecutar el proceso con el tiempo de ejecución estimado más corto.

Planificación garantizada

Un método completamente distinto para la planificación es hacer promesas reales a los usuarios acerca del rendimiento y después cumplirlas. Una de ellas, que es realista y fácil de cumplir es: si hay n usuarios conectados mientras usted está trabajando, recibirá aproximadamente 1/n del poder de la CPU.

De manera similar, **en un sistema de un solo usuario con n procesos en ejecución, mientras no haya diferencias, cada usuario debe obtener 1/n de los ciclos de la CPU. Eso parece bastante justo.**

Para cumplir esta promesa, **el sistema debe llevar la cuenta de cuánta potencia de CPU ha tenido cada proceso desde su creación**. Después calcula cuánto poder de la CPU debe asignarse a cada proceso, a saber el tiempo desde que se creó dividido entre n. Como la cantidad de tiempo de CPU que ha tenido cada proceso también se conoce, es simple calcular la proporción de tiempo de CPU que se consumió con el tiempo de CPU al que cada proceso tiene derecho.

Planificación por sorteo

Este algoritmo se conoce como planificación por sorteo (Waldspurger y Weihl, 1994).

La idea básica es dar a los procesos boletos de lotería para diversos recursos del sistema, como el tiempo de la CPU. Cada vez que hay que tomar una decisión de planificación, se selecciona un boleto de lotería al azar y el proceso que tiene ese boleto obtiene el recurso.

Cuando se aplica a la planificación de la CPU, el sistema podría realizar un sorteo 50 veces por segundo y cada ganador obtendría 20 mseg de tiempo de la CPU como premio.

Planificación por partes equitativas

Algunos sistemas toman en consideración quién es el propietario de un proceso antes de planificarlo.

En este modelo, a cada usuario se le asigna cierta fracción de la CPU y el planificador selecciona procesos de tal forma que se cumpla con este modelo.

Por ende, si a dos usuarios se les prometió 50 por ciento del tiempo de la CPU para cada uno, eso es lo que obtendrán sin importar cuántos procesos tengan en existencia.

Planificación en sistemas de tiempo real

Planificación en sistemas de tiempo real

En un sistema de tiempo real, el tiempo desempeña un papel esencial. Por lo general, uno o más dispositivos físicos externos a la computadora generan estímulo y la computadora debe reaccionar de manera apropiada a ellos dentro de cierta cantidad fija de tiempo. Tener la respuesta correcta pero demasiado tarde es a menudo tan malo como no tenerla.

En general, **los sistemas de tiempo real se categorizan como de tiempo real duro, lo cual significa que hay tiempos límite absolutos que se deben cumplir, y como de tiempo real suave, lo cual significa que no es conveniente fallar en un tiempo límite en ocasiones, pero sin embargo es tolerable**. En ambos casos, el comportamiento en tiempo real se logra dividiendo el programa en varios procesos, donde el comportamiento de cada uno de éstos es predecible y se conoce de antemano.

Planificación en sistemas de tiempo real

Por lo general, estos procesos tienen tiempos de vida cortos y pueden ejecutarse hasta completarse en mucho menos de 1 segundo. Cuando se detecta un evento externo, es responsabilidad del planificador planificar los procesos de tal forma que se cumpla con todos los tiempos límite.

Política contra mecanismo

Es posible que el proceso principal tenga una idea excelente acerca de cuál de sus hijos es el más importante (o que requiere tiempo con más urgencia) y cuál es el menos importante.

De esta manera, es posible separar el mecanismo de planificación de la política de planificación, un principio establecido desde hace tiempo (Levin y colaboradores, 1975). **Esto significa que el algoritmo de planificación está parametrizado de cierta forma, pero los procesos de usuario pueden llenar los parámetros.**

Suponga que el kernel utiliza un algoritmo de planficación por prioridad, pero proporciona una llamada al sistema mediante la cual un proceso puede establecer (y modificar) las prioridades de sus hijos. De esta forma, el padre puede controlar con detalle la forma en que se planifican sus hijos, aun y cuando éste no se encarga de la planificación. Aquí el mecanismo está en el kernel, pero la política se establece mediante un proceso de usuario.

Planificación de hilos

Cuando varios procesos tienen múltiples hilos cada uno, tenemos dos niveles de paralelismo presentes: procesos e hilos.

La planificación en tales sistemas difiere en forma considerable, dependiendo de si hay soporte para hilos a nivel usuario o para hilos a nivel kernel (o ambos).

Planificación de hilos

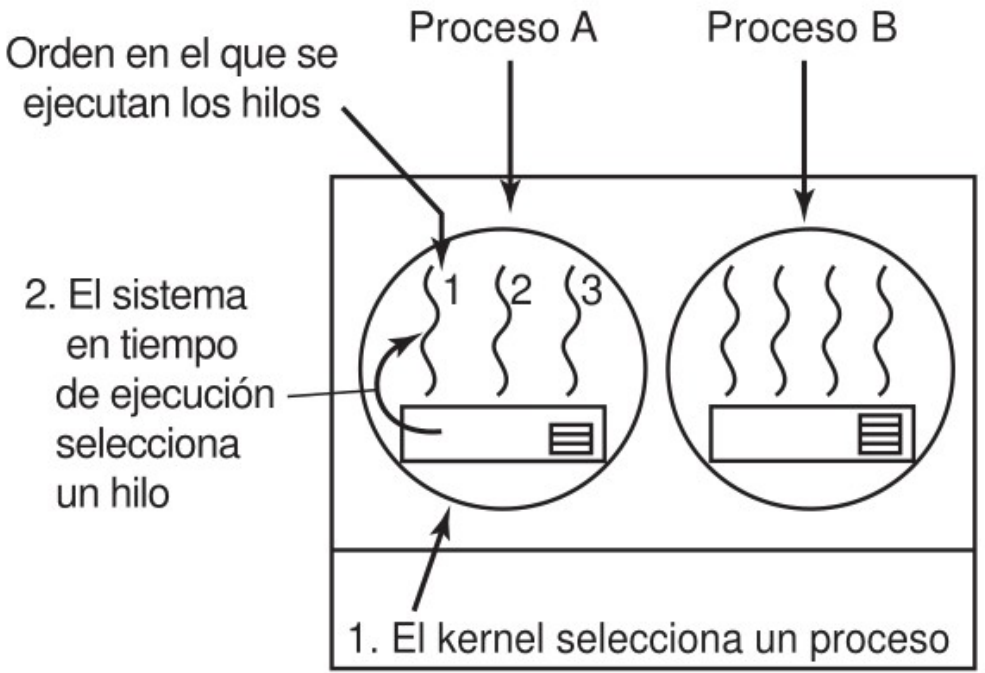
Consideremos primero los hilos a nivel usuario. Como el kernel no está consciente de la existencia de los hilos, opera en la misma forma de siempre: selecciona un proceso, por decir A, y otorga a este proceso el control de su quántum. El planificador de hilos dentro de A decide cuál hilo ejecutar, por decir A1. Como no hay interrupciones de reloj para multiprogramar hilos, este hilo puede continuar ejecutándose todo el tiempo que quiera. Si utiliza todo el quántum del proceso, el kernel seleccionará otro proceso para ejecutarlo. Cuando el proceso A por fin se ejecute de nuevo, el hilo A1 continuará su ejecución. Seguirá consumiendo todo el tiempo de A hasta que termine. Sin embargo, su comportamiento antisocial no afectará a los demás procesos.

Planificación de hilos

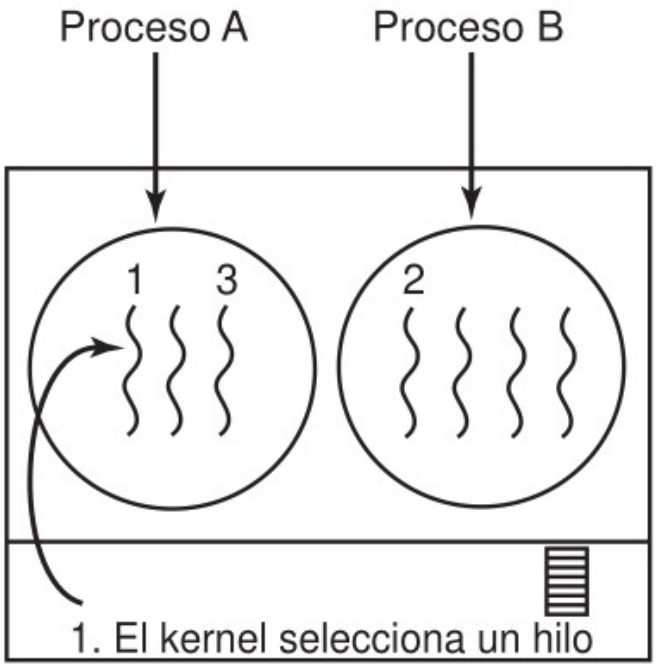
Si consideramos la situación con hilos a nivel kernel. Aquí el kernel selecciona un hilo específico para ejecutarlo. No tiene que tomar en cuenta a cuál proceso pertenece el hilo, pero puede hacerlo si lo desea. El hilo recibe un quántum y se suspende obligatoriamente si se excede de este quántum.

Una diferencia importante entre los hilos a nivel usuario y los hilos a nivel kernel es el rendimiento. Para realizar un conmutación de hilos con hilos a nivel usuario se requiere de muchas instrucciones de máquina. **Con hilos a nivel kernel se requiere una conmutación de contexto total, cambiar el mapa de memoria e invalidar la caché, lo cual es varias órdenes de magnitud más lento**. Por otro lado, con los hilos a nivel kernel, cuando un hilo se bloquea en espera de E/S no se suspende todo el proceso, como con los hilos a nivel usuario.

Planificación de hilos



Posible: A1, A2, A3, A1, A2, A3
Imposible: A1, B1, A2, B2, A3, B3



Posible: A1, A2, A3, A1, A2, A3
También posible: A1, B1, A2, B2, A3, B3

Bibliografía

- **Tanenbaum, A. S. (2009)**. Sistemas operativos modernos (3a ed.) (pp. 1-18). México: Pearson Educación.
- **Stallings, W. (2005)**. Sistemas operativos (5a ed.) (pp. 54-67). Madrid: Pearson Educación.