1 Planificación de procesos

Cuando hay más de un proceso ejecutable, el sistema operativo debe decidir cuál ejecutará primero. La parte del sistema operativo que toma esta decisión se denomina **planificador**: el algoritmo que usa se denomina **algoritmo de planificación**.

Para asegurarse de que ningún proceso se ejecute durante demasiado tiempo, casi todas las computadoras tienen incorporado un cronómetro o reloj electrónico que genera interrupciones periodicamente. Es común que la frecuencia sea de 50 o 60 interrupciones por segundo, pero en muchas computadoras el sistema operativo puede ajustar la frecuencia del cronómetro al valor que desee. En cada interrupción de reloj, el sistema operativo se ejecuta y decide si debe permitirse que el proceso que se está ejecutando actualmente continúe o si ya disfrutó de suficiente tiempo de CPU por el momento y debe suspenderse para otorgar a otro proceso la CPU.

La estrategia de permitir que procesos lógicamente ejecutables se suspendan temporalmente se denomina **planificación expropiativa** o **planificación apropiativa** en contraste con el método de **ejecución hasta terminar** (o RTC: Run To Completion) de los primeros sistemas por lotes. La ejecución hasta terminar (RTC) se denomina **planificación no expropiativa**. El hecho de que un proceso pueda ser suspendido en un instante arbitrario, sin advertencia, para que otro proceso pueda ejecutarse, da lugar a **condiciones de competencia** y requiere semáforos, monitores[1], mensajes o algún otro método avanzado para prevenirlas.

Se examinarán algunos algoritmos de planificación específicos.

1.1 Planificación *round robin* (de torneo)

Uno de los algoritmos de planificación más antiguos, sencillos, equitativos y ampliamente utilizados es el de **round robin**. A cada proceso se le asigna un intervalo de tiempo, llamado **cuanto**, durante el cual se le permite ejecutarse. Si el proceso aun se está ejecutando al expirar su cuanto, el sistema operativo se apropia de la CPU y se la da a otro proceso. Si el proceso se bloquea, el sistema operativo asigna la CPU a otro proceso. Si el proceso termina antes de expirar el cuanto, naturalmente la CPU también se conmuta. El *round robin* es facil de implementar. Todo lo que el planificador tiene que hacer es mantener

[1] Una primitiva de sincronización propuesta en la decada de los 70s del siglo pasado, véase [1], página 69

una lista de procesos ejecutables como se muestra en las figuras siguientes:

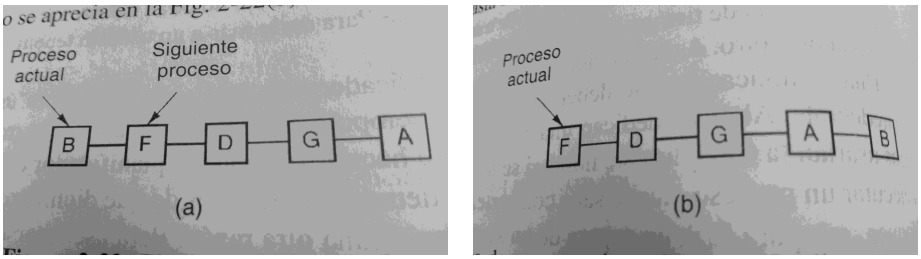

Figura 1 Planificación *round robin*

Cuando un proceso gasta su cuanto se le coloca al final de la lista como se muestra en la figura de la derecha. El punto clave en el planificador round robin es la duración del cuanto. La conmutación de un proceso a otro requiere cierto tiempo para llevar a cabo las tareas administrativas: guardar y cargar registros y mapas de memoria, actualizar diversas tablas y listas, etc. Supongamos que esta **conmutación de proceso** o **cambio de contexto** requiere 5 ms. Supongamos también que usamos cuantos de 20 ms. Con estos parámetros, después de realizar trabajo util durante 20 ms, la CPU tendrá que ocupar 5 ms en la conmutación de procesos. Se desperdiciará el 20% del tiempo de CPU en gastos extra administrativos. Como alternativa, podrían usarse cuantos de 500 ms, ahora el tiempo de "gasto" administrativo es aproximadamente del 1%, pero esto también puede acarrear situaciones adversas. Considere un sistema de tiempo compartido con 10 usuarios interactivos que pulsan la tecla de retorno de carro aproximadamente al mismo tiempo: diez procesos se pondrían en la lista de procesos ejecutables. Si la CPU está ociosa, el primero se iniciará de inmediato, el sgundo podría no iniciarse hasta cerca de medio segundo después, y así sucesivamente. El proceso que le haya tocado ser último podría tener que esperar 5 segundos antes de tener su oportunidad. Para casi cualquier usuario, un retardo de 5 segundos en la respuesta a un comando corto sería terrible. El mismo problema puede presentarse en una computadora personal que maneja multiprogramación.

En conclusión: escoger un cuanto demasiado corto causa demasiados cambios de contexto (conmutaciones de proceso) y reduce la eficiencia de la CPU, pero escogerlo demasiado largo puede dar pie a una respuesta deficiente a solicitudes interactivas. un cuanto de 100 ms suele ser un término medio razonable.

La planificación en *round robin* supone implícitamente que todos los procesos son igualmente importantes. Con fecuencia, las personas que poseen y operan sistemas de computadora consideran que algunos procesos deben ser considerados como más importantes que otros. En una computadora militar, los procesos iniciados por generales podrían iniciar con prioridad 100, los iniciados por coroneles con 90, por mayores con 80, por capitanes con 70, por tenientes con 60, etc. Incluso en una PC con un solo usuario, puede haber múltiples procesos, algunos más importantes que otros. Por ejemplo, un proceso demonio que envía correo electrónico en segundo plano debe tener menor prioridad que un proceso

que está exhibiendo video en tiempo real en la pantalla. La necesidad de tener en cuenta factores externos da origen a la **planificación por prioridad**.

1.2 Planificación por prioridad

La idea básica es sencilla: a cada proceso se le asigna una prioridad, y se permite que se ejecute el proceso ejecutable que tenga la prioridad más alta. Las prioridades se pueden asignar estática o dinámicamente.A fin de evitar que los procesos de alta prioridad se ejecuten indefinidamente, el planificador puede reducir la prioridad de los procesos que actualmente se ejecutan en cada tic del reloj (esto es, en cada interrupción de reloj). Si esta acción hace que la prioridad se vuelva menor que la de otro proceso, ocurrirá una conmutación de procesos. Como alternativa, se podría asignar a cada proceso un cuanto máximo en el que se le permitiera tener la CPU continuamente; cuando se agota este cuanto, se da oportunidad al proceso con la siguiente prioridad más alta de ejecutarse.

También puede asignarse prioridades dinámicamente a fin de lograr ciertos objetivos del sistema. Por ejemplo, algunos procesos están limitados para seguir avanzando principalmente debido a E/S y pasan la mayor parte del tiempo esperando que terminen operaciones de E/S, siempre que uno de estos procesos necesite la CPU, se le deberá otorgar de inmediato, con objeto de que pueda iniciar su siguiente solicitud de E/S, que entonces podrá proceder en paralelo con otro proceso que sí está realizando cálculos. Si hiciéramos que los procesos limitados por E/S esperaran mucho tiempo la CPU, implicaría tenerlos por ahí ocupando memoria duarnte un tiempo innecesariamente largo. Un algoritmo sencillo para dar buen servicio a los procesos limitados por E/S es asignarles la prioridad $1/f$, donde f es la fracción del último cuanto que un proceso utilizó. Un proceso que usó solo 2 ms de su cuanto de 100 ms recibiría una prioridad de 50, en tanto que un proceso que se ejecutó 50 ms antes de bloquearse obtendría una prioridad de 2, y uno que ocupó todo su cuanto obtendría una prioridad de 1.

En muchos casos es conveniente agrupar los procesos en clases de prioridad y usar planificación por prioridad entre las clases pero planificación *round robin* dentro de cada clase. Por ejemplo, en un sistema con 4 clases de prioridad con prioridades 1, 2, 3, 4 respectivamente, el algoritmo de planificación es el siguiente: en tanto haya procesos ejecutables en la clase de prioridad 4, se ejecutará cada uno durante un cuanto, con round robin, sin ocuparse de las clases de menor prioridad. Si la clase de prioridad 4 está vacia, se ejecutan los procesos de clase 3 con round robin. Si tanto la clase 4 como la 3 están vacias, se ejecutarán los procesos de la clase 2 con round robin, etc. Ocasionalmente se ajustarán las prioridades de las clases para que las clases de baja prioridad no mueran de inanición.

1.3 Colas múltiples

1.4 El primer trabajo más corto/Primero el trabajo más corto

1.5 Planificación garantizada

1.6 Planificación por lotería

1.7 Planificación en tiempo real

Un sistema de **tiempo real** es uno en el que el tiempo desempeña un papel esencial. Por lo regular, uno o más dispositivos externos a la computadora generan estímulos, y la computadora debe reaccionar a ellos de la forma apropiada dentro de un plazo fijo. Por ejemplo, la computadora de un reproductor de discos compactos recibe los bits conforme salen de la unidad de disco y los debe convertir en música dentro de un intervalo de tiempo muy estricto. Si el cálculo toma demasiado tiempo la música sonará raro. Otros sistemas de tiempo real son los de monitoreo de pacientes en las unidades de cuidado intensivo de los hospitales, el piló to automático de un avión y los controles de seguridad de un reactor nuclear. En todos estos casos, obtener la respuesta correcta pero demasiado tarde suele ser tan malo como no obtenerla.

Los sistemas de tiempo real generalmente de clasifican como de **tiempo real estricto**, lo que significa que hay plazos absolutos que deben cumplirse a como de lugar, y **tiempo real flexible**, lo que significa que es tolerable no cumplir ocasionalmente con un plazo. En ambos casos, el comportamiento de tiempo real se logra dividiendo el programa en varios procesos, cada uno de los cuales tiene un comportamiento predecible y conocido por adelantado. Estos procesos generalmente son de corta duración y pueden ejecutarse hasta terminar en menos de un segundo. Cuando se detecta un suceso externo, el planificador debe programar los procesos de modo tal que se cumplan todos los plazos.

Los sucesos a los que puede tener que responder un sistema de tiempo real pueden clasificarse también como **periódicos** (que ocurren a intervalos regulares) o **aperiódicos** (que ocurren de forma impredecible). Es posible que un sistema tenga que responder a múltiples corrientes de eventos periodicos. Dependiendo de cuánto tiempo requiere cada suceso por separado, tal vez ni siquiera sea posible manejarlos todos. por ejemplo, si hay m eventos periodicos y el evento i ocurre con el periodo P_i y requiere C_i segundos de tiempo de CPU para ser manejado, la carga solo puede manejarse si

$$\sum_{i=0}^{m} \frac{C_i}{P_i} \leq 1$$

Un sistema de tiempo real que satisface este criterio es **planificable**.

Por ejemplo, consideremos un sistema de tiempo real flexible con tres sucesos periodicos, con periodos de 100, 200 y 500 ms respectivamente. Si estos eventos requieren 50, 30 y 100 ms detiempo de CPU por evento, respectivamente, el

sistema es planificable porque

$$\frac{50}{100} + \frac{30}{200} + \frac{100}{500} = 0.5 + 0.15 + 0.2 = 0.85 < 1$$

Si se agrega un cuarto evento con un periodo de 1 s, el sistema seguirá siendo planificable en tanto que este evento no necesite más de 150 ms de tiempo de CPU por evento. Un supuesto implícito en este cálculo es que el gasto extra del cambio de contexto es tan pequeño que puede ignorarse.

Pag. 91 [1], sistemas operativos disenio e implementacion.

References

[1] Andrew S. Tanenbaum, "Sistemas Operativos, Diseño e Implementación," Editorial Pearson, 2002.