Planificación de CPU
5

Sistemas operativos y distribuidos

Gustavo Distel
gd@cs.uns.edu.ar

DCIC - UNS

Planificación de CPU

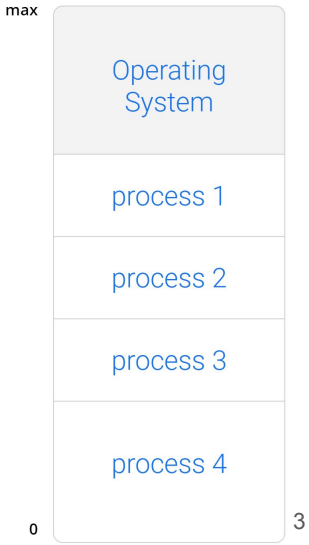
Contenido

- Conceptos básicos.
- Criterios de planificación.
- Algoritmos de planificación.
- Planificación de hilos.
- Planificación multiprocesador.
- Planificación de *CPU* en tiempo real.
- Ejemplos de sistemas operativos.
- Evaluación de algoritmos.

Conceptos básicos

- La multiprogramación maximiza la utilización de la *CPU* manteniendo varios procesos en memoria a ser ejecutados.
- Un proc. se ejecuta hasta que debe esperar, generalmente para completar alguna solicitud de E/S.

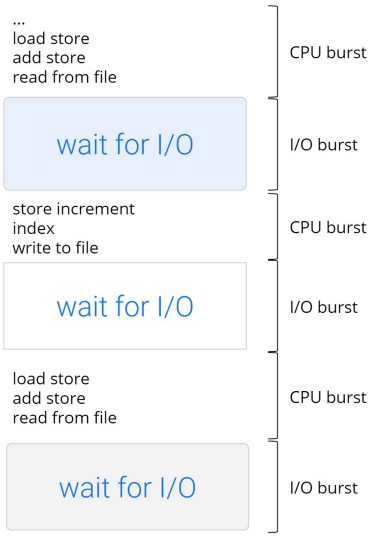
- El SO le quita la *CPU* al proc. en espera y se la otorga a otro proc.
- Este patrón continúa. Y cada vez que un proc. tiene que esperar, otro puede utilizar la *CPU*.
- Este tipo de planificación es una función fundamental del SO. Casi todos los recursos informáticos se planifican antes de su uso.
- La *CPU* es, desde luego, uno de los principales recursos, por lo tanto su planificación es fundamental para el diseño del SO.

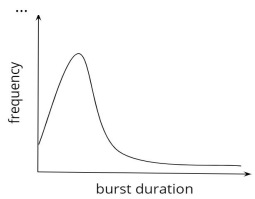


Conceptos básicos

Ciclo de ráfaga *CPU* - E/S

- El éxito de la planificación de la *CPU* depende de una propiedad de los procesos:
  - La ejecución de un proc. comienza con una ráfaga de *CPU*, seguida por una ráfaga de E/S, que es seguida por otra ráfaga de *CPU*, luego otra ráfaga de E/S, y así sucesivamente.
  - Finalmente, la ráfaga final de *CPU* concluye con una solicitud al sistema para terminar la ejecución.
- Un programa ligado por E/S tiene generalmente muchas ráfagas de *CPU* cortas.
- Un programa ligado por *CPU* tiene generalmente pocas ráfagas de *CPU* largas.
- Esta distribución es importante al implementar un algoritmo de planificación de *CPU*.





Conceptos básicos

Planificador de *CPU*

- Cada vez que la *CPU* queda inactiva, el SO debe seleccionar uno de los proc. de la cola de listos para ser ejecutado.
- El planificador de la *CPU* lleva a cabo el proceso de selección, escogiendo un proc. de memoria que esté listo para ejecutarse asignándole la *CPU*.
- La cola de listos no es necesariamente una cola de primero en entrar primero en salir (*FIFO*).
- Conceptualmente todos los procesos en la cola de listos están alineados esperando la oportunidad de ejecutarse en la *CPU*.
- Las entradas en las colas son generalmente los *PCBs* de los procesos.

Conceptos básicos

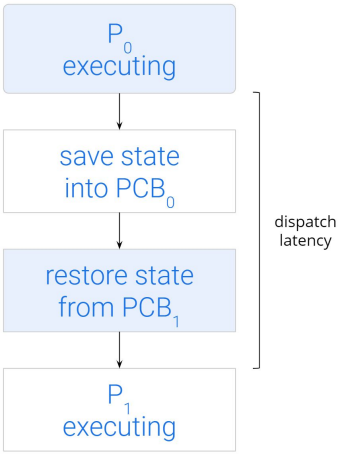
Planificación apropiativa y no apropiativa

- Decisiones de planificación:
  - 1. Cuando un proc. cambia del estado de ejecución al estado de espera (`E/S` o `wait()`).
  - 2. Cuando un proc. cambia del estado de ejecución al estado listo (interrupción).
  - 3. Cuando un proc. cambia del estado de espera al estado listo (finalizó E/S).
  - 4. Cuando termina un proc.
- 1 y 4: no hay opción, se debe seleccionar un nuevo proc. para su ejec. → planificación no apropiativo o cooperativa.
  - El proc. mantiene la *CPU* hasta que la libera, ya sea terminando o cambiando al estado de espera.
- 2 y 3: hay opción → planificación apropiativa.
- Prácticamente todos los SOs modernos (Windows, Mac OS, Linux y UNIX) utilizan algoritmos de planificación apropiativa.
- La planificación apropiativa puede resultar en *race conditions* así como también afectar el diseño del *kernel*.

Conceptos básicos

Dispatcher

- El despachador es el módulo que da el control del *core* de la *CPU* al proc. seleccionado por el planificador de la *CPU*. Esta función implica:
  - Cambiar el contexto de un proc. a otro
  - Cambiar a modo de usuario
  - Saltar a la ubicación adecuada en el programa de usuario para reanudarlo.
- El despachador debe ser lo más rápido posible, ya que se invoca durante cada cambio de contexto.
- El tiempo que tarda el despachador en detener un proc. e iniciar otra ejecución se conoce como latencia de despacho.



Criterios de planificación

- La elección de un algoritmo particular puede favorecer una clase de proc. sobre otra, por lo que debemos considerar las propiedades de los diversos algoritmos.
- Las características que se utilizan en la comparación pueden marcar una diferencia sustancial en qué algoritmo se considera mejor. Los criterios incluyen lo siguiente:
  - **Utilización de la *CPU*:** mantener la *CPU* lo más ocupada posible. Conceptualmente puede variar de 0 a 100%, pero en un sistema real debe oscilar entre un 40% y un 90%.
  - ***Throughput* (Rendimiento):** número de proc. que se completan por unidad de tiempo.
  - ***Turnaround time* (Tiempo de retorno o tiempo de ejecución):** corresponde al tiempo transcurrido desde el lanzamiento de un proc. hasta su finalización. Es la suma de los períodos pasados esperando en la cola de listos, ejecutándose en la *CPU* y haciendo E/S.
  - ***Waiting time* (Tiempo de espera):** suma de los períodos invertidos en esperar en la cola de proc. listos
  - ***Response time* (Tiempo de respuesta):** es el tiempo que el proc. tarda en empezar a responder (no el tiempo que tarda en enviar a la salida toda la información de respuesta). Generalmente, el tiempo de respuesta está limitado por la velocidad del dispositivo de salida.

Criterios de planificación

- Es deseable maximizar la utilización y el *throughput* y minimizar el *turnaround time*, el tiempo de espera y el tiempo de respuesta.
- En la mayoría de los casos, se optimizan los valores promedio. Sin embargo, en algunas circunstancias es preferible optimizar los valores mínimos o máximos en lugar del promedio.
- Por ejemplo, para garantizar que todos los usuarios obtengan un buen servicio, se puede minimizar el tiempo de respuesta máximo.
- Explicaremos los diversos algoritmos de planificación con la ayuda de un ejemplo.
  - Para poder ilustrar el funcionamiento de forma precisa, sería ideal utilizar muchos procesos, cada uno compuesto por una secuencia de varios cientos de ráfagas de *CPU* y E/S.
  - No obstante, lo simplificaremos considerando solo una ráfaga de *CPU* (en milisegundos) por cada proc.

Algoritmos de planificación

- Considerando el siguiente conjunto de procesos, donde la ráfaga de *CPU* está dada en milisegundos.

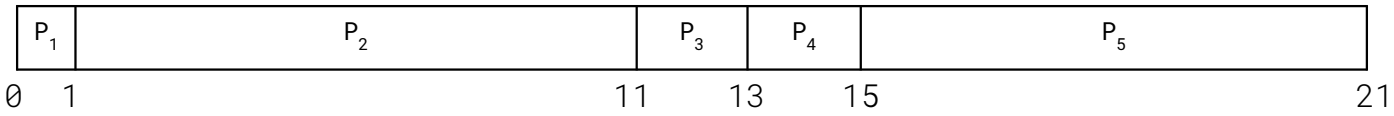
Proceso	Tiempo de ráfaga	Prioridad
P_1	1	1
P_2	10	4
P_3	2	3
P_4	2	4
P_5	6	2

- Se asume que los procesos arriban en el orden P1, P2, P3, P4, P5, todos en el tiempo (instante) 0.

Algoritmos de planificación

FCFS (*First-Come, First-Served;* primero en llegar, primero en ser servido)

- Asigna la *CPU* al proc. que primero la solicite.
- **Cooperativo**: Una vez que la *CPU* ha sido asignada a un proc., dicho proc. la conserva hasta que la libera porque realiza una E/S o porque termina.

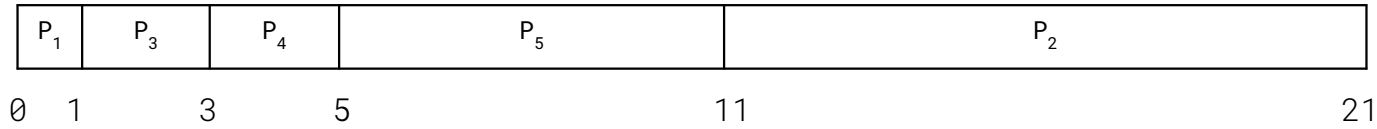


- Si bien es uno de los algoritmos más simples (se puede implementar con una cola *FIFO*), el tiempo medio de espera es a menudo bastante largo.
- Padece el efecto *convoy* que ocurre cuando tenemos un proc. limitado por *CPU* y muchos limitados por E/S. Los limitados por E/S se alinean en un convoy (uno detrás de otro) esperando que el proc. limitado por CPU finalice.

Algoritmos de planificación

SJF (*Shortest-Job-First*; primero tarea más corta)

- Asigna la *CPU* al proc. que tenga la siguiente ráfaga más corta.
- Si las siguientes ráfagas de *CPU* de dos procesos son iguales, se usa la planificación *FCFS* para romper el empate (P_3 y P_4).

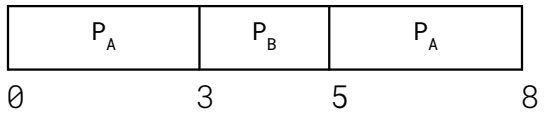


- Favorece a los trabajos cortos a costa de los más largos.
- Es probablemente óptimo, en el sentido de que proporciona el mínimo tiempo medio de espera para un conjunto de procesos dado.
- La dificultad real del algoritmo es que no hay forma de conocer la duración de la siguiente ráfaga de *CPU*. Un método consiste en intentar aproximarla asumiendo que la siguiente ráfaga de *CPU* será similar en duración a las anteriores.

Algoritmos de planificación

SRT(*SRTF*) (*Shortest-Remaining-Time-first*; tiempo restante más corto primero)

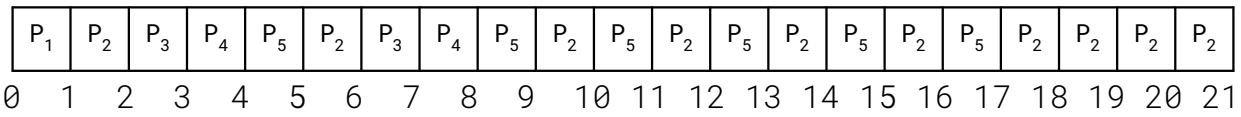
- Es la contraparte apropiativa de *SJF*, que incluye las nuevas llegadas.
- En nuestro ej., si bien arriban en orden, todos llegan en tiempo 0. Como no llega ninguno nuevo y se parte del más corto en tiempo, siempre seguirá siendo el más corto a medida que se ejecute, por lo que es igual que en el algoritmo *SJF*.
- La diferencia tiene lugar cuando arriban nuevos trabajos, ya que de esta forma se pone de manifiesto la cualidad apropiativa del algoritmo.
- Por ej.: si se está ejecutando un proc. P_A con un tiempo de ejecución de 6ms y han pasado 3ms y entra un proc. P_B que requiere 2ms para ejecutarse, lo expropiará.



Algoritmos de planificación

RR(*Round-Robin*; planificación por turnos)

- Es similar a la planificación *FCFS*, pero se añade la técnica de apropiación para conmutar entre procesos.
- Además se define una pequeña unidad de tiempo: cuanto de tiempo (*time quantum*).
- La cola de procesos listos se trata como una cola circular. El planificador recorre la cola asignando la *CPU* a cada proc. durante un cuanto de tiempo.

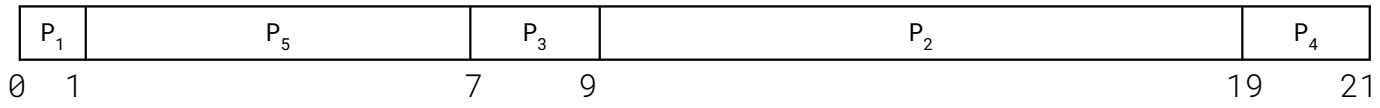


- El tiempo medio de espera es generalmente largo.
- El rendimiento del algoritmo depende enormemente del tamaño del cuanto de tiempo:
  - Si cuanto de tiempo es largo → la planificación *RR* es igual a la *FCFS*.
  - Si cuanto de tiempo es corto → habrá gran número de cambios de contexto.
- Es conveniente que el cuanto de tiempo sea grande con respecto al tiempo de cambio de contexto, para que un porcentaje pequeño de la utilización de la *CPU* se use para cambios de contexto.

Algoritmos de planificación

Prioridades no apropiativo

- A cada proc. se le asocia una prioridad y se asigna la *CPU* al proc. que tenga la prioridad más alta. Los procesos con igual prioridad se planifican en orden *FCFS*.
- El algoritmo *SJF* es un caso especial del algoritmo de prioridades, donde la prioridad es la ráfaga (a mayor ráfaga menor será la prioridad y viceversa).
- No hay consenso en si 0 es la prioridad más alta o más baja. Nosotros asumimos que los números bajos representan una prioridad alta (Ejemplo: Windows, Linux).
- En nuestro ejemplo la planificación es cooperativa, pero puede ser también apropiativa.



Algoritmos de planificación

- **Tiempo de retorno (o tiempo de ejecución)**: corresponde al tiempo transcurrido desde el lanzamiento de un proc. hasta su finalización.

Proceso	FCFS	SJF	RR	Prioridad
P_1	1	1	1	1
P_2	11	21	21	19
P_3	13	3	7	9
P_4	15	5	8	21
P_5	21	11	17	7
Tiempo medio	12.2	8.2	10.8	11.4

Algoritmos de planificación

- **Tiempo de espera**: es la suma de los períodos invertidos en esperar en la cola de procesos listos.
  - Matemáticamente se puede expresar como el tiempo de retorno "menos" el tiempo de ráfaga.

Proceso	FCFS	SJF	RR	Prioridad
P_1	0	0	0	0
P_2	1	11	11	9
P_3	11	1	5	7
P_4	13	3	6	19
P_5	15	5	11	1
Tiempo medio	8	4	6.6	7.2

Algoritmos de planificación

- **Tiempo de respuesta**: es el tiempo que el proc. tarda en empezar a responder, no el tiempo que tarda en enviar a la salida toda la información de respuesta.
  - Generalmente, el tiempo de respuesta está limitado por la velocidad del dispositivo de salida.

Proceso	FCFS	RR	SJF	Prioridad
P_1	0	0	0	0
P_2	1	1	11	9
P_3	11	2	1	7
P_4	13	3	3	19
P_5	15	4	5	1
Tiempo medio	8	2	4	7.2

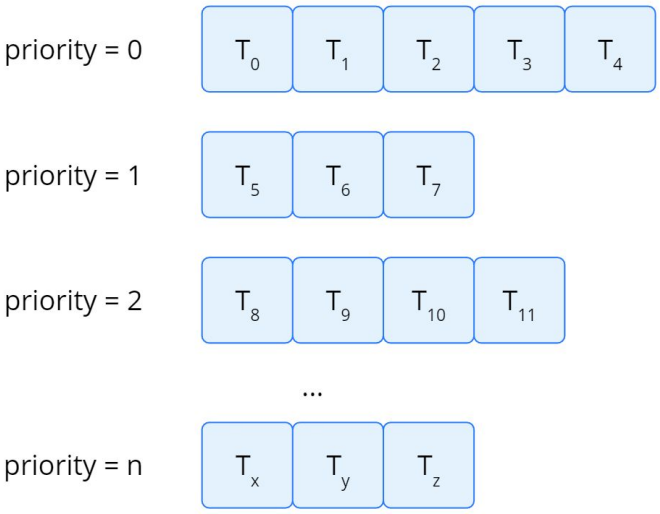
Algoritmos de planificación

- De los algoritmos anteriores *SJF* y el de prioridades podrían dar lugar a la inanición.
- **Inanición**: un proc. que está preparado para ejecutarse pero está esperando acceder a la *CPU* puede considerarse bloqueado; un algoritmo de planificación por prioridades puede dejar a algunos procesos de baja prioridad esperando indefinidamente.
  - **Solución**: el envejecimiento que consiste en aumentar gradualmente la prioridad de los procesos que estén esperando en el sistema duramente mucho tiempo.
- Del análisis de las tablas anteriores se puede concluir que los mejores resultados se obtienen para el algoritmo *SJF*:
  - El menor tiempo de retorno medio (8.2).
  - El tiempo de espera medio más bajo (4).
- La evaluación que hicimos sobre los algoritmos se basa en un modelado determinista.
  - **Modelado determinista**: es un tipo de evaluación analítica en la cual se toma una carga de trabajo predeterminada y se define el rendimiento de cada algoritmo para dicha carga de trabajo.

Algoritmos de planificación

Colas multinivel

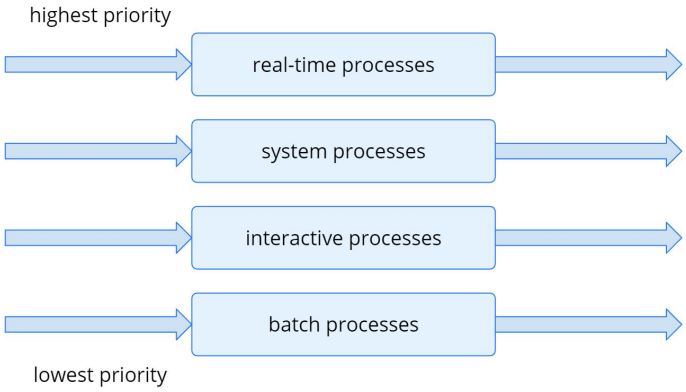
- Colas separadas por prioridad, el planificador selecciona el proc. en la cola de mayor prioridad.
- La planificación por prioridad se puede combinar con *RR*: si hay varios procesos en la cola de mayor prioridad, se ejecutan en orden de *round-robin*.
- En la forma más generalizada de este enfoque, se asigna una prioridad estática a cada proc., y un proc. permanece en la misma cola durante su ejecución.



Algoritmos de planificación

Colas multinivel

- Otra opción es dividir procesos en varias colas separadas según el tipo (*foreground, background*).
- Ejemplo: alg. de planificación de colas multinivel con 4 colas, enumeradas en orden de prioridad:
  - Procesos en tiempo real
  - Procesos del sistema
  - Procesos interactivos
  - Procesos por lotes
- Cada cola tiene prioridad absoluta sobre las colas de menor prioridad.

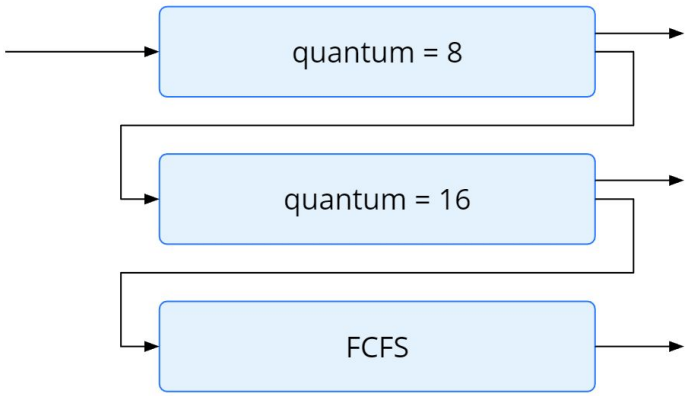


- Si un proc. interactivo ingresa a la cola de listos mientras se ejecuta un proc. por lotes, este será reemplazado.
- Otra posibilidad es el *time-slice* entre colas. Cada cola obtiene una porción del tiempo de *CPU* para planificar entre sus diversos procesos, por ej.:
  - *Foreground* 80% para *RR*.
  - *Background* 20% para *FCFS*.

Algoritmos de planificación

Colas multinivel retroalimentadas

- Permite que un proc. se mueva entre colas.
- Separa los procesos de acuerdo con las características de sus ráfagas de *CPU*.
  - Si usa mucha *CPU*, se moverá a una cola de menor prioridad.
  - Los proc. ligados por E/S e interactivos, con ráfagas de *CPU* cortas, en las colas de mayor prioridad.
  - Un proc. que espera demasiado en una cola de menor prioridad se puede mover a una de mayor prioridad.
  - Envejecimiento (*aging*) que previene la inanición (*starvation*).



Algoritmos de planificación

Colas multinivel retroalimentadas

- Se define mediante los siguientes parámetros:
  - El número de colas.
  - El algoritmo de planificación para cada cola.
  - El método utilizado para determinar cuándo actualizar un proc. a una cola de mayor prioridad.
  - El método utilizado para determinar cuándo degradar un proc. a una cola de menor prioridad
  - El método utilizado para determinar a qué cola ingresará un proc. cuando necesite servicio.
- Algoritmo de planificación de *CPU* general. Se puede configurar para un sistema específico, aunque se torna complejo ya que hay seleccionar valores para todos los parámetros.

Planificación de hilos

- Recordemos que se distingue entre hilos a nivel usuario e hilos a nivel de *kernel*.
- En la mayoría de los SOs modernos, éste planifica los hilos a nivel de *kernel*.
- Los hilos a nivel de usuario son administrados por una biblioteca de hilos, que el *kernel* desconoce.
- Para ejecutarse en la *CPU*, los hilos a nivel de usuario deben mapearse en última instancia a un hilo de nivel *kernel*, aunque este mapeo puede ser indirecto y puede usar un proc. ligero (*LWP*).

Planificación de hilos

Alcance de contención (*Contention Scope*)

- Una distinción entre hilos a nivel de usuario e hilos a nivel de *kernel* radica en cómo se planifican.
- En los sistemas que implementan los modelos **muchos a uno** y **muchos a muchos**, la biblioteca de hilos planifica hilos a nivel de usuario para que se ejecuten en un *LWP* disponible.
- Este esquema se conoce como alcance de contenido de proc. (*Process Contention Scope - PC*S), ya que la competencia por la *CPU* tiene lugar entre los hilos que pertenecen al mismo proc.
- Cuando afirmamos que la biblioteca de hilos planifica los hilos de los usuarios en los *LWP* disponibles, no nos referimos a que los hilos realmente se estén ejecutando en una *CPU*, ya que eso requiere que el SO planifique el hilo del *kernel* del *LWP* en un core de *CPU* físico.
- Para decidir qué hilo a nivel de *kernel* se planifica en una *CPU*, el *kernel* utiliza el alcance de contención del sistema (*SCS*). La competencia por la *CPU* con la planificación *SCS* se lleva a cabo entre todos los hilos del sistema.
- Sistemas que utilizan el modelo **uno a uno**, como Windows y Linux, planifican hilos utilizando *SCS*.

Planificación de hilos

Alcance de contención (*Contention Scope*)

- Normalmente, *PCS* se realiza con prioridad:
  - El planificador selecciona el hilo con la prioridad más alta para ejecutar.
- Lo programadores establecen las prioridades de hilos a nivel de usuario y no la biblioteca, aunque algunas bibliotecas de hilos permiten al programador cambiar las prioridades.
- Es importante tener en cuenta que *PCS* generalmente apropiará al hilo que se ejecuta actualmente en favor de un hilo de mayor prioridad; sin embargo, no hay garantía de *time slicing* entre hilos de igual prioridad.

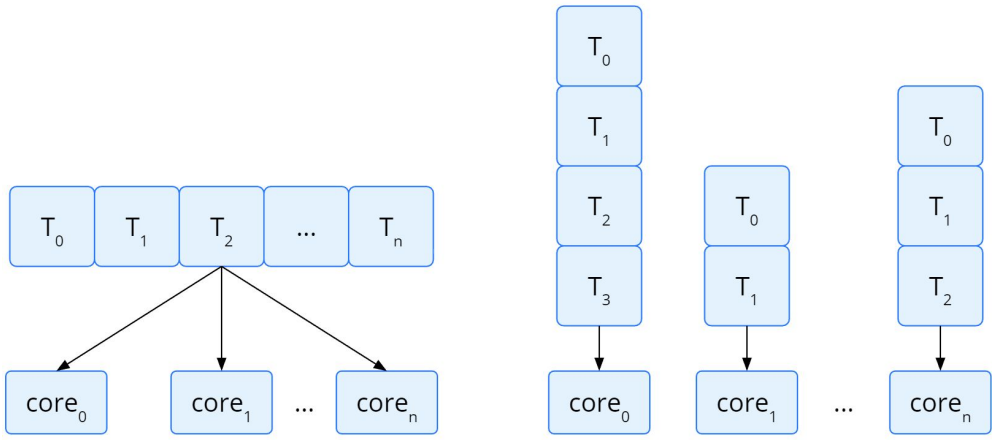
Planificación multiprocesador

- Si hay varias *CPUs* disponibles, es posible compartir la carga, donde múltiples hilos pueden ejecutarse en paralelo; sin embargo, los problemas de planificación se vuelven más complejos.
- Hay muchas posibilidades y, como vimos con la planificación de *CPU*, con *CPU* de un solo *core* no hay una mejor solución.
- Tradicionalmente, el término multiprocesador se refería a sistemas que proporcionaban múltiples procesadores físicos, donde cada procesador contenía una *CPU* de un solo *core*.
- Sin embargo, la definición de multiprocesador ha evolucionado significativamente y en los sistemas informáticos modernos, el multiprocesador se aplica a las siguientes arquitecturas:
  - *CPUs multicore*.
  - *Multithreaded cores*.
  - *Sistemas NUMA*.
  - *Multiprocesamiento heterogéneo*.

Planificación multiprocesador

Enfoques para la planificación de múltiples procesadores

- En el multiprocesamiento simétrico (*SMP*) cada procesador se planifica individualmente.
  - Todos los hilos pueden estar en una cola de listos común.
  - Cada procesador puede tener su propia cola privada de hilos.



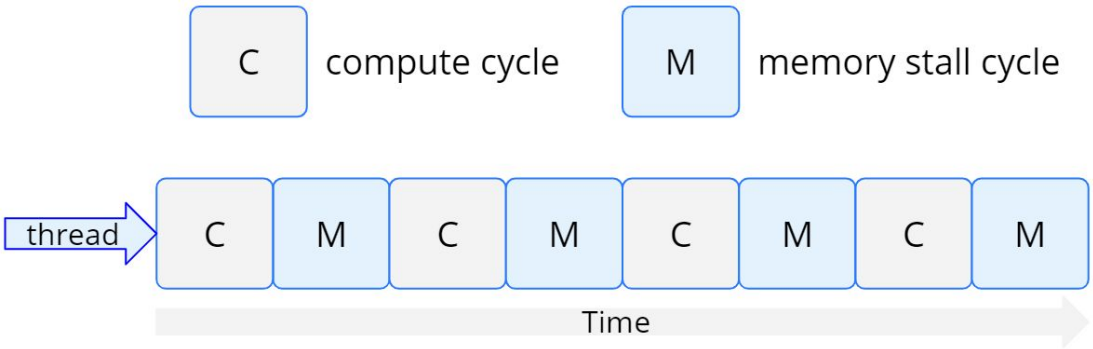
common ready queue

per-core run queues

Planificación multiprocesador

Procesadores *Multicore*

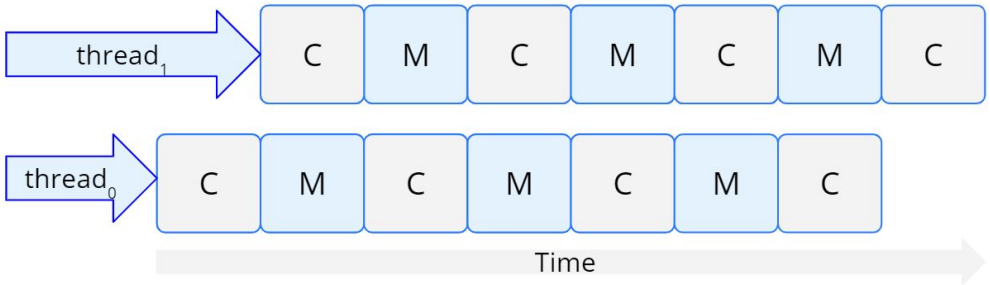
- Recientemente, existe una tendencia a colocar múltiples *cores* de procesador en el mismo chip físico.
- Es más rápido y consume menos energía.
- Se da un incremento de múltiples hilos por núcleo.
  - Se aprovechan los *memory stall* para avanzar en otro hilo mientras se realiza la recuperación de memoria.



Planificación multiprocesador

Procesadores *Multicore*

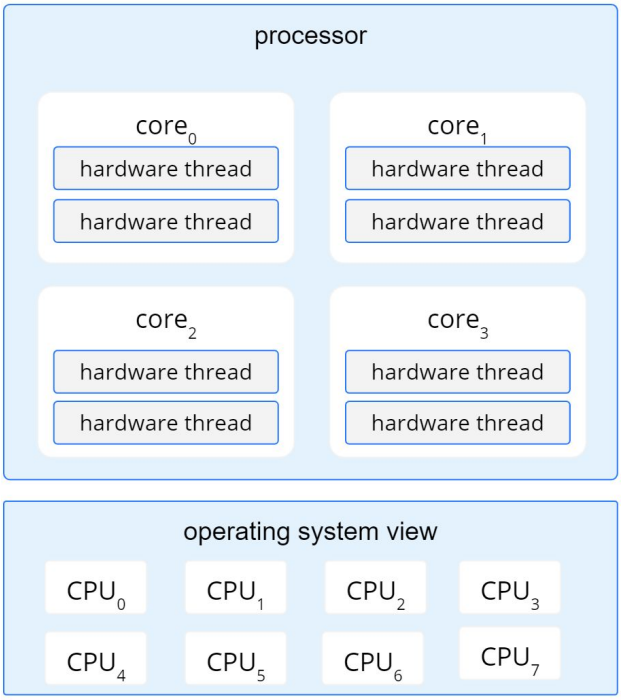
- Cada core tiene dos o más hilos de *hardware*.
- Si un hilo tiene un *memory stall*, se debe cambiar a otro hilo.



Planificación multiprocesador

Procesadores *Multicore*

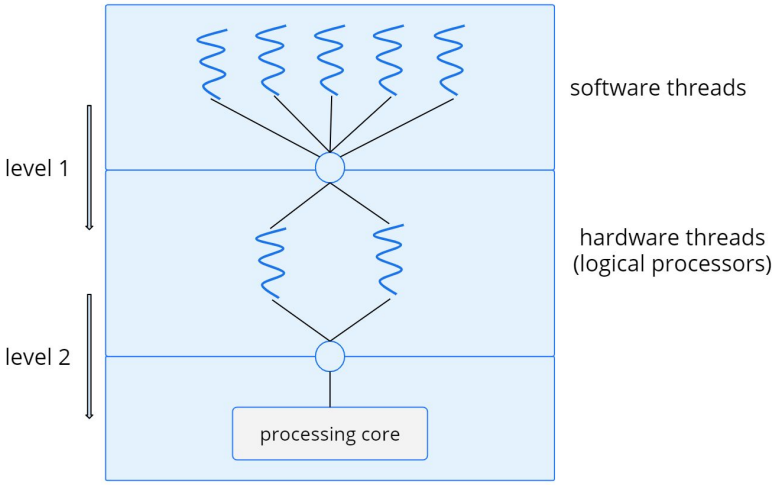
- *Chip-multithreading* (*CMT*) asigna a cada *core* múltiples hilos de *hardware*. (Intel se refiere a esto como *hyperthreading*).
- En un sistema de cuatro *cores* con 2 hilos de hardware por *core*, el sistema operativo ve 8 procesadores lógicos.



Planificación multiprocesador

Procesadores *Multicore*

- Dos niveles de planificación:
  - El SO decide qué hilo de *software* ejecutar en una *CPU* lógica.
  - Cada *core* decide qué hilo de *hardware* ejecutar en el *core* físico.



Planificación multiprocesador

Balanceo de carga (*Load Balancing*)

- Si es *SMP*, debe mantener todas las *CPU* cargadas para mayor eficiencia.
- *Load balancing*: intenta mantener la carga de trabajo distribuida uniformemente.
- *Push migration*: la tarea periódica verifica la carga en cada procesador y, si se encuentra, envía la tarea de la *CPU* sobrecargada a otras *CPU*.
- *Pull migration*: los procesadores inactivos extraen la tarea de espera del procesador ocupado.

Planificación multiprocesador

Afinidad del procesador (*Processor Affinity*)

- Cuando un hilo se ha estado ejecutando en un procesador, el contenido de la memoria caché de ese procesador almacena los accesos de memoria de ese hilo.
- Esto se define como un hilo que tiene afinidad por un procesador (es decir, "afinidad del procesador")
- El *load balancing* puede afectar la afinidad del procesador, ya que un hilo se puede mover de un procesador a otro para equilibrar las cargas; sin embargo, ese hilo pierde el contenido que tenía en la memoria caché del procesador del que se retiró.
- Afinidad suave: el sistema operativo intenta mantener un hilo ejecutándose en el mismo procesador, pero no existen garantías.
- Afinidad dura: permite que un proc. especifique un conjunto de procesadores en los que puede ejecutarse.

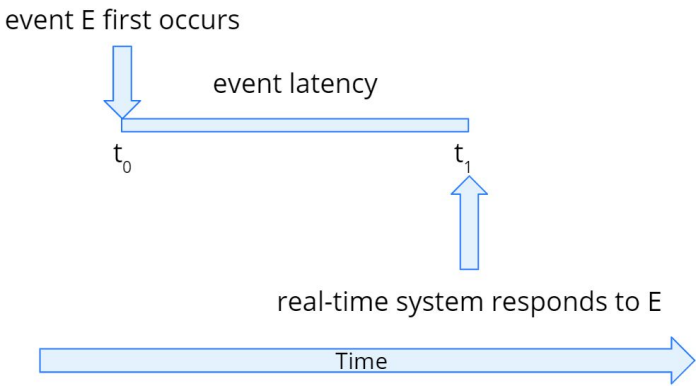
Planificación de CPU en tiempo real

- La planificación de la *CPU* para SOs en tiempo real implica problemas especiales.
- Se puede distinguir entre dos tipos:
  - ***Soft real-time systems***: no proporciona ninguna garantía de cuándo se planificará un proc. crítico en tiempo real. Solo garantizan que el proceso tendrá preferencia sobre los procesos no críticos.
  - ***Hard real-time systems***: Tiene requisitos más estrictos. Una tarea debe ser atendida en su *deadline* (brindar el servicio luego de que el *deadline* haya expirado es lo mismo que no ofrecer ningún servicio).

Planificación de CPU en tiempo real

Minimizando la latencia

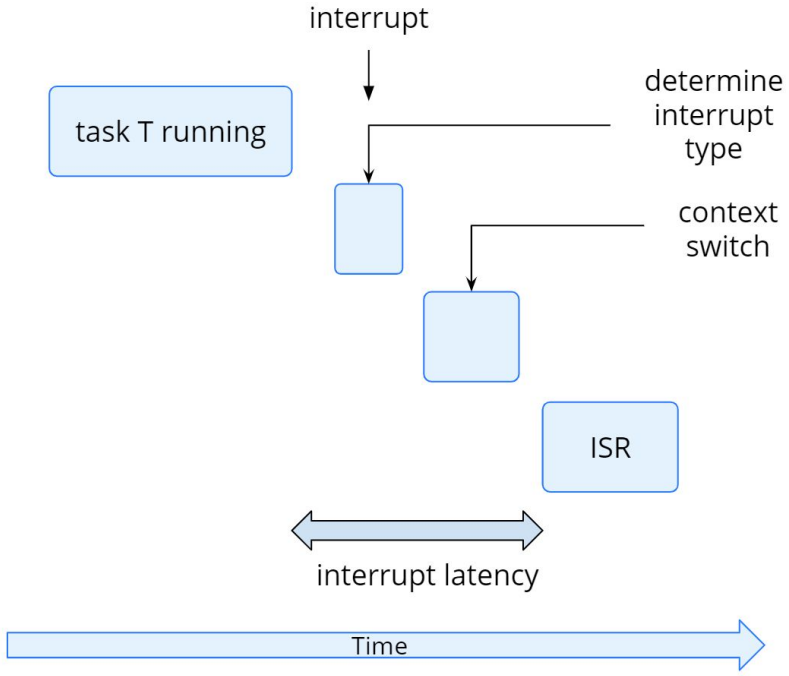
- Cuando ocurre un evento (*SW* o *HW*), el sistema debe responder y atenderlo lo más rápido posible.
- ***Event latency***: la cantidad de tiempo que transcurre desde que ocurre un evento hasta que es atendido.
- Diferentes eventos tienen diferentes requisitos de latencia (sistema antibloqueo de frenos vs. sistema de control de radar).
- Dos tipos de latencias afectan el rendimiento:
  - ***Interrupt latency***: período de tiempo desde la llegada de una interrupción a la *CPU* hasta el inicio de la rutina que da servicio a la interrupción.
  - ***Dispatch latency***: cantidad de tiempo requerida para que el despachador de planificador detenga un proc. e inicie otro.



Planificación de CPU en tiempo real

Minimizando la latencia

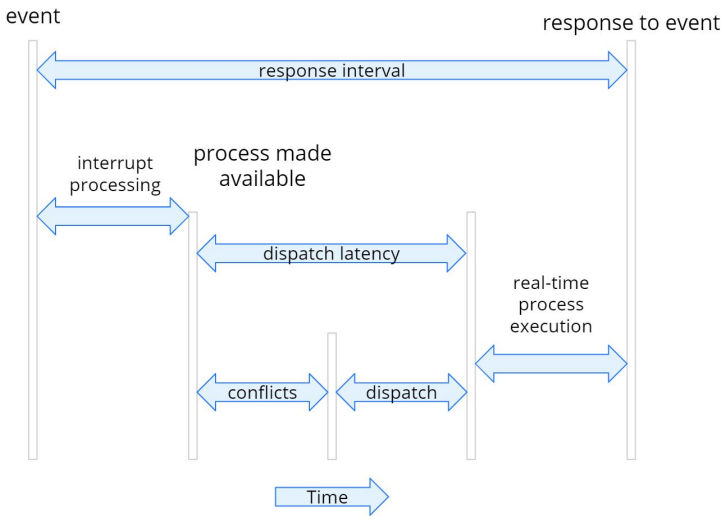
Interrupt latency



Planificación de CPU en tiempo real

Minimizando la latencia

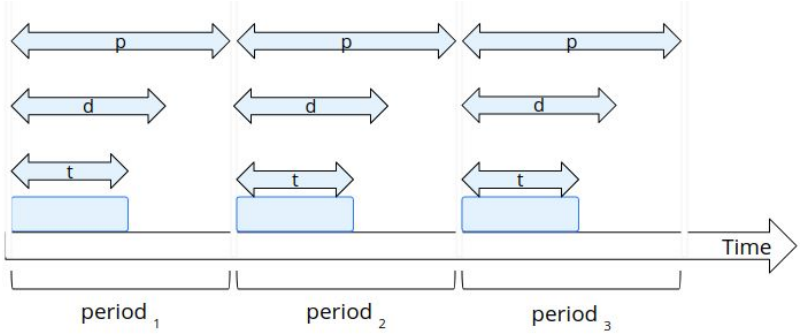
- Proporcionar tareas en tiempo real con acceso inmediato a la *CPU,* exige que los SO minimicen esta latencia.
- **Fase de conflicto** de latencia de despacho:
  - Apropiación de cualquier proc. que se ejecute en el *kernel*.
  - Liberar recursos en procesos de baja prioridad en favor de recursos necesarios por un proc. de alta prioridad.



Planificación de CPU en tiempo real

Planificación basada en prioridades (*Priority-Based Scheduling*)

- El planificador para un SO en tiempo real debe admitir un algoritmo basado en prioridades con apropiación.
  - Solo garantizado para *soft real-time*.
  - *Hard real-time* requiere *deadlines* por lo que se necesitan características adicionales.
- Nuevas características en procesos:
  - Los procesos se consideran periódicos (*periodic*), es decir que requieren la *CPU* a intervalos constantes (períodos - *periods*).
  - Una vez que un proc. periódico ha adquirido la *CPU*, tiene un tiempo de procesamiento fijo ***t***, un *deadline* ***d*** por la cual debe ser atendido por la *CPU* y un período ***p***.
  - La relación entre estos valores es: $0 \leq t \leq d \leq p$.
  - El *rate* de la tarea periódica es 1/p.



Planificación de CPU en tiempo real

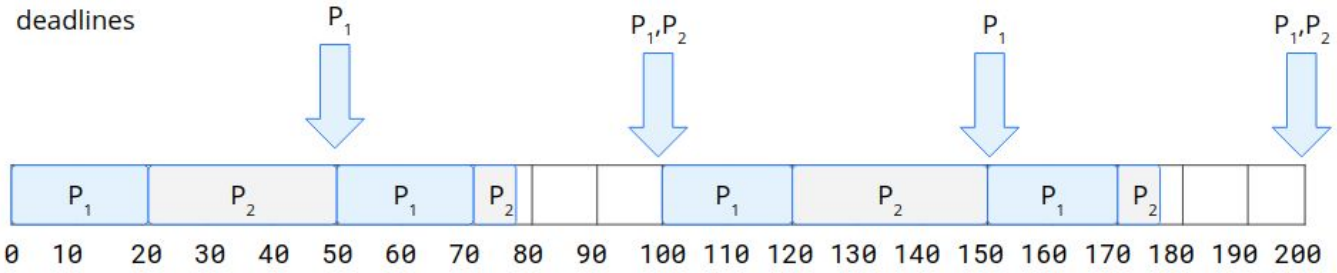
Planificación *Rate-Monotonic*

- Planifica tareas periódicas utilizando una política de prioridad estática con apropiación.
- Si está ejecutando un proc. de menor prioridad y está disponible un proc. de mayor prioridad, apropiará al de menor prioridad.
- Al ingresar al sistema, a cada tarea periódica se le asigna una prioridad que se relaciona inversamente con de su período (cuanto más corto es el período, mayor es la prioridad; cuanto más largo sea el período, menor será la prioridad).
- La razón detrás de esta planificación es asignar una mayor prioridad a las tareas que requieren la *CPU* con más frecuencia.
- Asume que el tiempo de procesamiento de un proc. periódico es el mismo para cada ráfaga de *CPU*, es decir que cada vez que un proc. adquiere la *CPU*, la duración de su ráfaga de *CPU* es la misma.

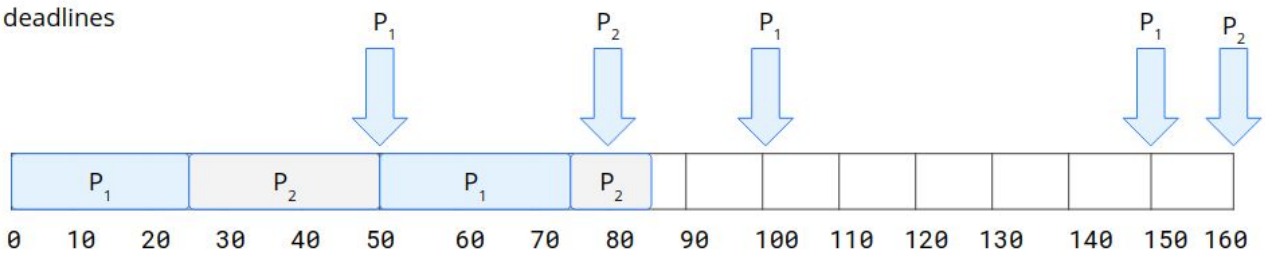
Planificación de CPU en tiempo real

Planificación *Rate-Monotonic*

- P1 tiene asignada una prioridad más alta que P2.



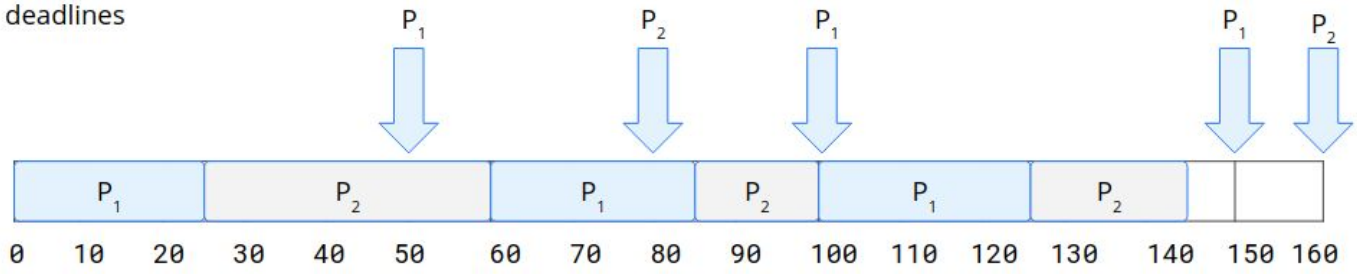
- El proc. P2 falla al finalizar su fecha límite en el momento 80.



Planificación de CPU en tiempo real

Planificación *Earliest-Deadline-First*

- Asigna prioridades dinámicamente de acuerdo con el *deadline*.
- Cuanto antes sea el *deadline*, mayor será la prioridad y cuanto más tarde el *deadline*, menor la prioridad.
- Cuando un proc. se vuelve ejecutable, debe anunciar su *deadline* al sistema.
- Las prioridades pueden tener que ajustarse para reflejar la fecha límite del nuevo proc. ejecutable.
- Esta planificación difiere de la *rate-monotonic*, donde las prioridades son fijas.



Planificación de CPU en tiempo real

Planificación *Proportional Share*

- Asigna de recursos compartidos (*shares*) T entre todas las aplicaciones.
- Una aplicación puede recibir N *shares* de tiempo, asegurando así que la aplicación tendrá N/T del tiempo total del procesador.
- Como ej., consideremos que un total de T = 100 *shares* se dividirá entre tres procesos, A, B y C.
  - Al proceso A se le asignan 50 *shares*, a B se le asignan 15 *shares* y a C se le asignan 20 *shares*.
- Este esquema asegura que A tendrá el 50% del tiempo total del procesador, B tendrá el 15% y C tendrá el 20%.
- Se debe trabajar junto con una política de control de admisión para garantizar que una aplicación reciba *share* de tiempo asignado.
- Una política de control de admisión admitirá a un cliente que solicite un número particular de *shares* solo si hay suficientes *shares* disponibles.

Planificación de CPU en tiempo real

Planificación *POSIX Real-Time*

- *POSIX* define dos clases de planificación para hilos en tiempo real:
  - `SCHED FIFO`
  - `SCHED RR`
- La *API POSIX* especifica las siguientes dos funciones para obtener y configurar la política de planificación:
  - `pthread attr getschedpolicy(pthread attr t *attr, int *policy)`
  - `pthread attr setschedpolicy(pthread attr t *attr, int policy)`

Ejemplos de sistemas operativos

- Planificación de Linux
- Planificación de Windows
- Planificación de Solaris