Sistemas Operativos

Universidad Complutense de Madrid

2020-2021

Módulo 3.2: Planificación

Juan Carlos Sáez

Contenido

Contenido

Planificación

Objetivos

- Optimizar uso de las CPUs
- Minimizar tiempo de espera
- Ofrecer reparto equitativo (justicia)
- Proporcionar grados de urgencia (prioridades)

Tipos de algoritmos de planificación

- **No expropiativo**: el proceso conserva la CPU hasta que (1) se bloquea, (2) la cede expresamente o (3) termina su ejecución.
- **Expropiativo**: el SO puede expulsar al proceso de la CPU
  - Exige un reloj que interrumpe periódicamente

Estructuras de datos del planificador

Estructuras de datos

- El planificador mantiene los procesos/hilos en una cola (*run queue*)
  - Típicamente se implementa como lista doblemente enlazada
- La *run queue* está formada por los BCPs de los procesos listos para ejecutar
  - El proceso que está actualmente en ejecución en la CPU no se mantiene en la *run queue*
  - El planificador no gestiona procesos en estado "bloqueado"
- Algunos algoritmos de planificación mantienen varias colas de procesos
  - Por prioridad, por tipo, ...

Activación del planificador

Puntos de activación

- Periódicamente (interrupción del temporizador de la CPU)
- Como resultado del procesamiento de alguna interrupción generada por otros dispositivos de E/S
- El proceso en ejecución causa una excepción que lo bloquea (fallo de página) o fuerza su terminación (violación de segmento)
- Cuando el proceso en ejecución termina
- El proceso realiza una llamada bloqueante
- Cesión voluntaria del procesador
  - `sched_yield()`
- Se desbloquea un proceso más "importante" que el actual
  - *expropiación de usuario*

Métricas del planificador

Métricas por entidad (proceso o thread)

- Tiempo de ejecución o de retorno
  - $T_{ejecución} = T_{fin} - T_{creación}$
- Tiempo de espera: tiempo total que el proceso pasa esperando en la cola del planificador (listo para ejecutar)
- Tiempo de respuesta:
  - $T_{respuesta} = T_{primerUsoDeCPU} - T_{creación}$

Métricas globales

- Porcentaje de utilización del procesador
- Productividad: número de trabajos completados por unidad de tiempo

Contenido

Algoritmos no expropiativos

- El planificador no quita la CPU al proceso una vez que está en ejecución, a no ser esta la ceda voluntariamente, termine o se bloquee por E/S

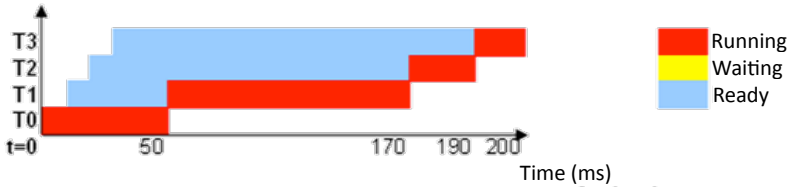
Algoritmos

- Primero en llegar primero en ejecutar o FCFS (*First-Come First-Served*)
- Primero el trabajo más corto o SJF (*Shortest Job First*)
  - También conocido como SPN (*Shortest Process Next*)
- Planificación basada en prioridades

Primero en llegar primero en ejecutar (FCFS)

- *Run queue* gestionada como cola FIFO
- Algoritmo simple que optimiza el uso de CPU

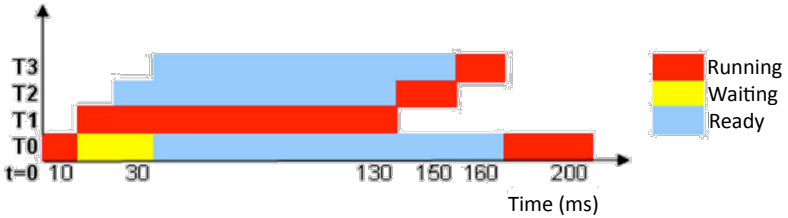
Proceso o thread	Tiempo de llegada (ms)	Tiempo de CPU (ms)
T0	0	50
T1	10	120
T2	20	20
T3	30	10



Primero en llegar primero en ejecutar (FCFS)

- Programas con E/S son encolados al final
- Programas largos afectan al sistema

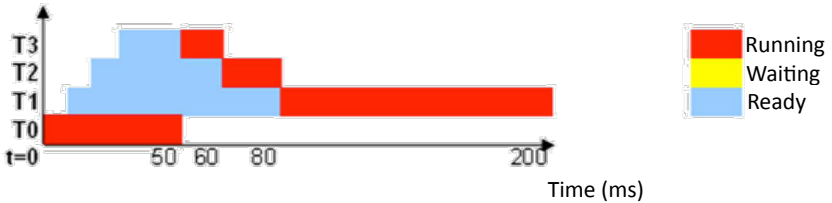
Proceso o thread	Tiempo de llegada (ms)	Tiempo de CPU (ms)
T0	0	50
T1	10	120
T2	20	20
T3	30	10



Primero el trabajo más corto (SJF)

- Bueno para programas interactivos
- Necesita conocer el perfil de las tareas
- Problemas de inanición

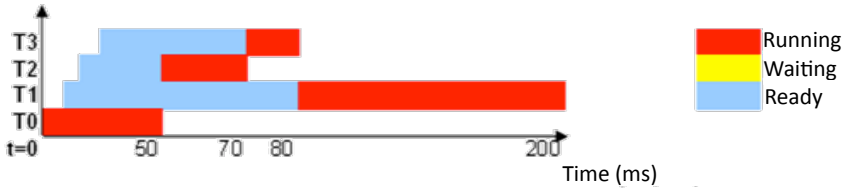
Proceso o thread	Tiempo de llegada (ms)	Tiempo de CPU (ms)
T0	0	50
T1	10	120
T2	20	20
T3	30	10



Planificación basada en prioridades

- El usuario especifica el nivel de urgencia de cada proceso
- Problema de inanición:
  - Solución: Aumento de la prioridad con la edad

Proceso o thread	Llegada (ms)	Tiempo de CPU (ms)	Prioridad
T0	0	50	4
T1	10	120	3
T2	20	20	1
T3	30	10	2



Algoritmos expropriativos

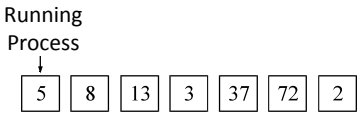
- No expropiativos no son adecuados para SSOO de propósito general
  - Mezcla de trabajos interactivos y trabajos intensivos en CPU
- Los algoritmos expropiativos se activan periódicamente
  - El temporizador del sistema se configura para generar interrupciones periódicas por cada CPU (~ms)
    - Cada interrupción se denomina *tick*
    - Config. por defecto en Linux/x86: 250 *ticks* por segundo (4ms)

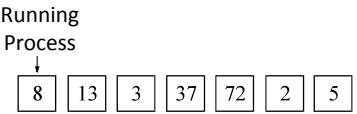
Algoritmos

- *Round Robin* - RR (turno rotatorio)
- Primero el de menor tiempo restante - SRTF
  - *Shortest Remaining Time First*
- Prioridad expropiativa
- Colas multinivel

Round Robin - RR (I)

- La planificación se realiza dividiendo el tiempo de CPU en rodajas llamadas *quanto* o *time slice* (expresado en ticks)
- RR: $\rightarrow$ FCFS + time slice
  - El planificador expropia al proceso en ejecución cuando consume su *time slice*
  - Cuando proceso es expropiado, RR lo inserta al final de la cola
  - Implementación: cada proceso tiene un contador de *ticks* asociado
    - Inicialmente contador=_ticks time slice_
    - Cada tick, el planificador decrementa el contador del proceso/hilo en ejecución
    - Expropiación $\Longleftrightarrow$ contador = 0

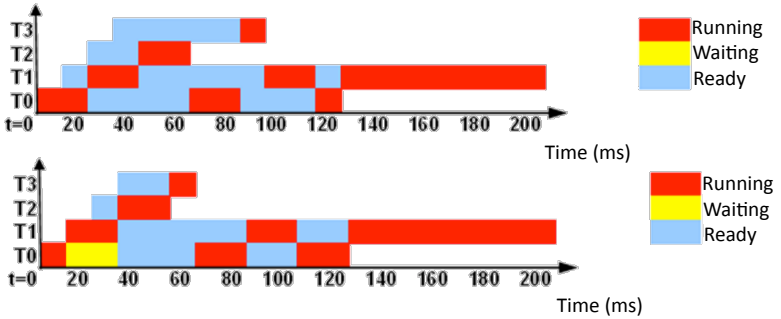




Round Robin - RR (I)

Proceso o thread	Tiempo de llegada (ms)	Tiempo de CPU (ms)
T0	0	50
T1	10	120
T2	20	20
T3	30	10

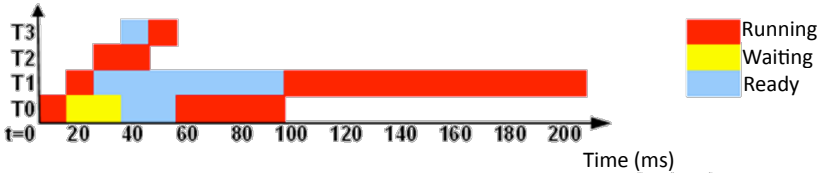
Timeslice=20ms



Primero el de menor tiempo restante (SRTF)

- SRTF: SJF + expropiación
  - Bueno para programas interactivos
  - Necesita conocer el perfil de las tareas
  - Problemas de inanición

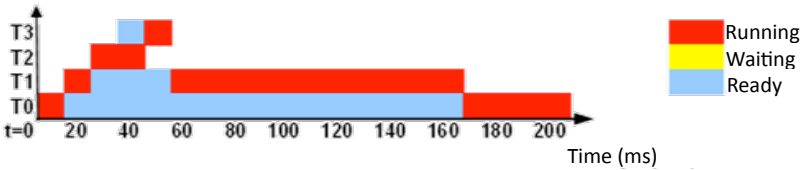
Proceso o thread	Tiempo de llegada (ms)	Tiempo de CPU (ms)
T0	0	50
T1	10	120
T2	20	20
T3	30	10



Expropiativo basado en prioridades

- El usuario especifica el nivel de urgencia de cada proceso
- Problema de inanición:
  - Solución: Aumento de la prioridad con la edad

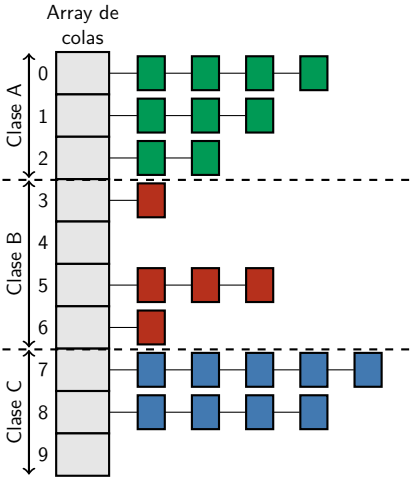
Proceso o thread	Llegada (ms)	Tiempo de CPU (ms)	Prioridad
T0	0	50	4
T1	10	120	3
T2	20	20	1
T3	30	10	2



Planificación con colas multinivel (I)

- Objetivo: dar soporte a distintas clases de procesos
- En el sistema existen k niveles de prioridad
  - Se mantiene una cola de procesos para cada nivel (array de colas)
  - En cada nivel de prioridad puede haber un *time slice* diferente
- Los niveles de prioridad se agrupan en rangos para dar servicio a distintos tipos de procesos/hilos
  - Tiempo real
  - Hilos de sistema
  - Interactivos
  - Batch
  - ...

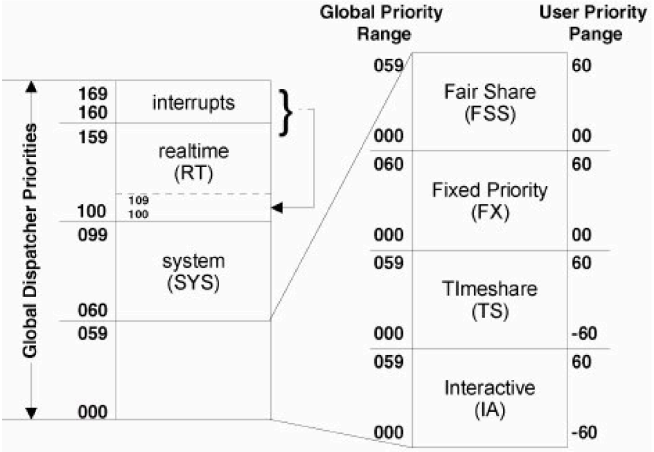
Planificación con colas multinivel (II)



Planificación con colas multinivel (III)

- La planificación se realiza a dos niveles:
  - Global (*dispatcher*): Elección del siguiente proceso a ejecutar y realización de cambios de contexto
    - Siempre se escoge el proceso más prioritario del sistema que está listo para ejecutar
  - Local (*scheduling class*): Gestión las colas asociadas a un determinado rango de prioridades (tipo particular de procesos)
    - Gestión de timeslices y procesamiento de tick
    - La clase de planificación *decide cuándo se ha de expropiar al proceso actual*
    - Invocación al dispatcher para efectuar expropiaciones de usuario

Ejemplo: Planificador de Solaris



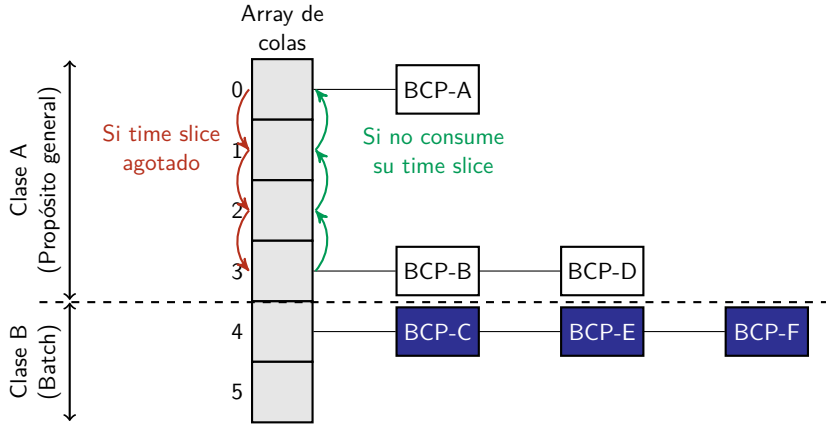
Planificación con colas multinivel (IV)

Dos alternativas de gestión de las colas de procesos:

1. Sin realimentación (procesos con prioridad fija)
   - Proceso en la misma cola (cuando está listo para ejecutar)
2. Con realimentación (procesos con prioridad dinámica)
   - Los procesos pueden cambiar de nivel
     - El cambio de nivel sólo se produce dentro del rango de prioridades gestionado por el "planificador local"
   - Necesario definir política de cambio de nivel
     - Ejemplo: Política para favorecer a procesos interactivos

```
Si proceso agota su timeslice, baja de nivel
Si proceso no agota su timeslice (p. ej., bloqueo E/S),
   sube de nivel
```

Ejemplo: Multinivel con realimentación

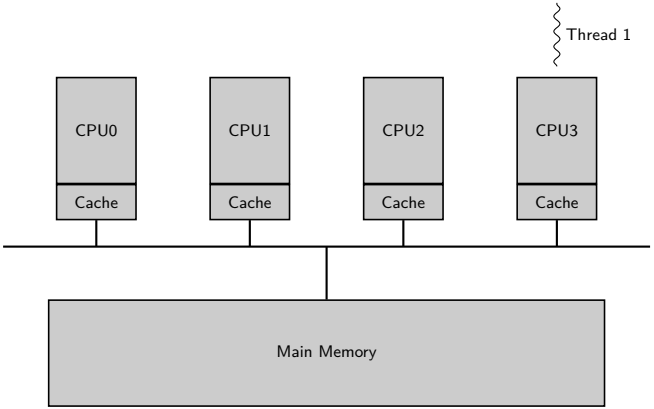


Contenido

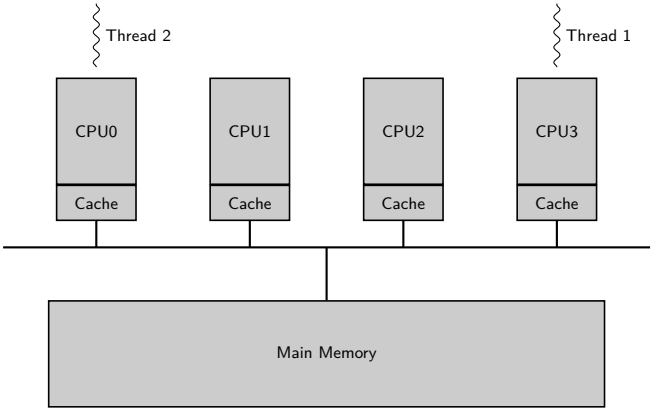
Planificación SMP

- SMP (*Symmetric Multi-Processing*)
- Garantizar un equilibrio de carga (load balancing)
  - Que no haya un procesador ocioso y otros con mucha carga de trabajo
- Tener en cuenta la afinidad de procesos y procesadores
  - Importante al replanificar un proceso
  - Evitar realizar migraciones de hilos
- Tener en cuenta la compartición de datos entre procesos/hilos si hay varios nodos de memoria (NUMA)
  - Si dos hilos comparten memoria, probablemente sea bueno que compartan todo lo posible su nivel de jerarquía

Concepto de afinidad

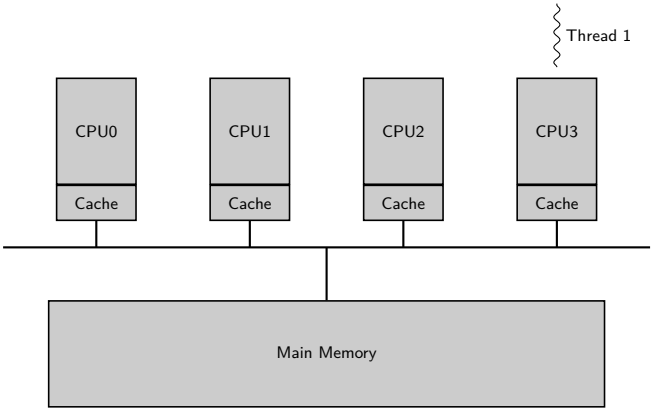


Concepto de afinidad



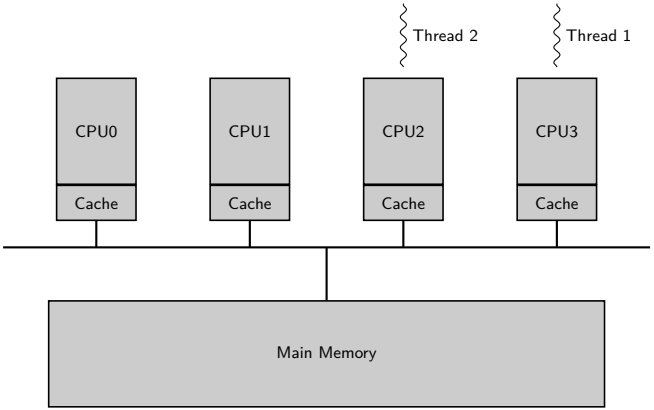
Un nuevo hilo entra al sistema (Thread 2). Al ejecutarse, carga parte de sus datos en la cache (CPU0). El hilo desarrolla *afinidad* a la CPU0 (*cache hot*).

Concepto de afinidad



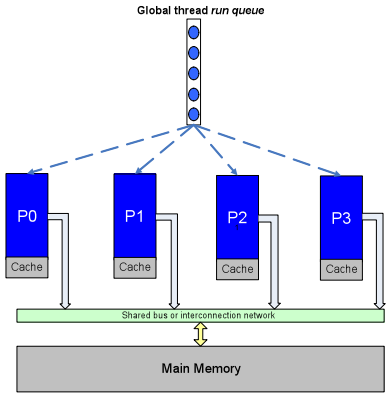
El hilo 2 se bloquea por E/S.

Concepto de afinidad



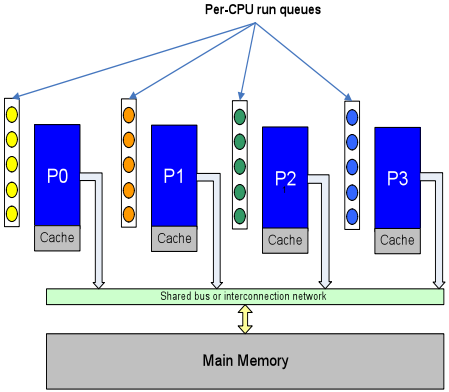
El hilo 2 se despierta (fin E/S) y el planificador lo asigna a un procesador diferente (CPU 2).
Migración de hilo → degradación del rendimiento.

Planificación SMP (Linux v2.4.x)



- Una única run queue para todos los procesadores
- Equilibrio en la carga
  - Todos los procesadores tienen potencialmente el mismo trabajo
- Malo para la afinidad
  - El proceso A se ejecutó en la CPU1 y luego se envía a CPU2 (migración)
  - Migración → reconstruir estado de cache
- Problemas de escalabilidad

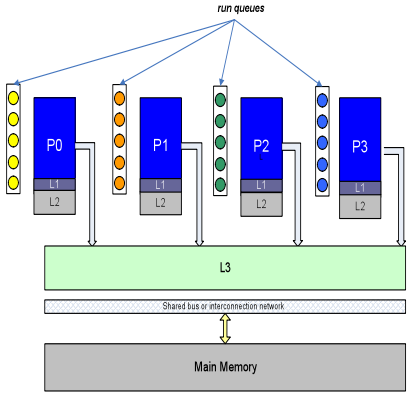
Planificación SMP (Linux v2.6.x+)



- Una cola de ejecución por procesador
- Mayor escalabilidad
- Periódicamente (o bajo demanda) se ejecuta el equilibrador de carga
  - Considera qué procesos pueden/deben migrarse
  - Tiene en cuenta la afinidad

Este modelo es el que utilizan la mayor parte de SSOO actuales de propósito general (Linux, Solaris, FreeBSD o MS Windows)

Planificación en multicore

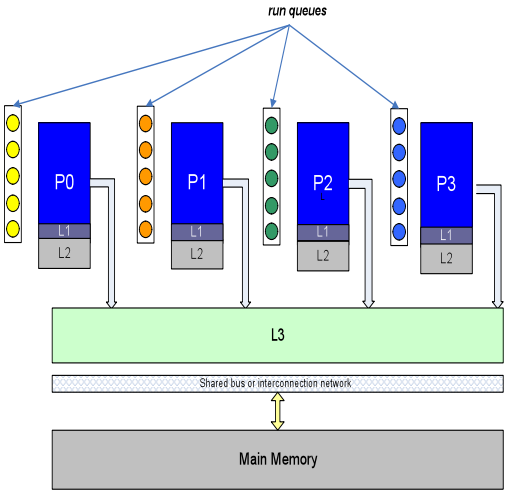


- El SO ve cada core como un procesador independiente, pero no lo es
  - Algún nivel de cache compartido entre cores
- Potencial degradación del rendimiento por contención en recursos compartidos
- Problemas de justicia

Planificación en multicore: área de investigación activa

Contenido

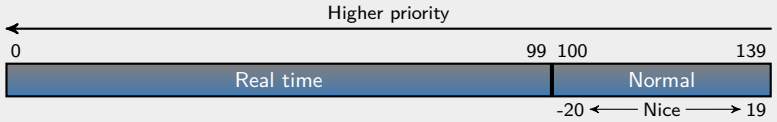
Planificación en Linux



Planificación en Linux (v3.14+)

140 niveles de prioridad

- 100 para procesos real-time
  - 3 políticas de planificación: *deadline*, RR y FIFO
- 40 para procesos normales (CFS)
  - Prioridad puede cambiarse con el comando `nice`
  - `$ nice -n <valor_nice> <comando_aplicación>`
    - `<valor_nice>` $\in$ [-20,19]



Completely Fair Scheduler (CFS)

Objetivos CFS (*Completely Fair Scheduler*)

- Intenta garantizar una distribución justa del tiempo de CPU considerando la prioridad de los procesos
  - CFS no usa *time slices*
- Proporcionar buenos tiempos de respuesta
  - Adecuado para entornos interactivos (GUIs)

Idea general

- Si 4 hilos de la misma prioridad estuvieran en el sistema durante 40 ms, cada hilo debería ejecutarse durante 10 ms para asegurar una distribución uniforme (*justicia*)
  - *¿Qué deberíamos hacer si los hilos tuviesen distintas prioridades?*

CFS: distribución de tiempo de CPU

- Tiempo de ejecución se divide en intervalos de longitud variable llamados *sched period*
  - En cada sched period cada proceso activo debe planificarse al menos una vez
- En cada sched period, para cada proceso P:
  - $T_{CPU}(P) = sched\_period\_ms \cdot \frac{peso(P)}{\sum_{i=1}^{n} peso(i)}$

Ejemplo

- 3 procesos A, B y C, con pesos 2,2 y 1, respectivamente
- sched_period_ms=20ms
- $T_{CPU}(A) = T_{CPU}(B) = 8ms$ y $T_{CPU}(C) = 4ms$

CFS: distribución de tiempo de CPU

- Si sched_period fijo (p.ej., 20ms) y número de procesos muy elevado $\rightarrow$ $T_{CPU}(P_i) \approx 0$
  - Cambios de contexto muy frecuentes
  - Planificador sólo puede reaccionar cada *tick* (p.ej., 4ms)

`min_granularity`

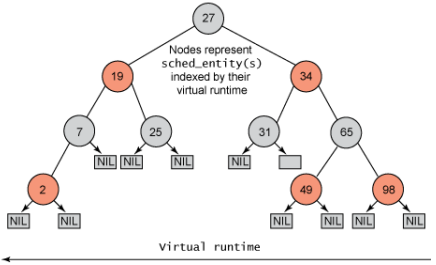
- Longitud de cada *sched period* se fija en base al número de procesos activos y otros parámetros
- Todo proceso puede ejecutarse durante un cierto tiempo (`min_granularity`) sin ser expulsado
  - Después de este tiempo, el planificador comprueba si el proceso debe ser expropiado o no
- Un proceso puede abandonar la CPU antes de tiempo por otras razones (E/S, ceder la CPU voluntariamente,...)

CFS: Siguiente proceso a ejecutar

- CFS lleva la cuenta del tiempo de CPU virtual (vruntime) que cada proceso ha recibido
  - El *vruntime* de un proceso se incrementa cada vez que éste consume un tick de CPU o fracción (p.ej. se bloquea antes)
  - El tiempo de CPU virtual transcurre más rápidamente para procesos de menor prioridad y más lentamente para los de mayor prioridad
    - $\text{Unidad\_tiempo\_virtual}(P) = \text{Unidad\_tiempo\_real} \cdot \frac{Peso_{nice=0}}{Peso_P}$
    - $Peso_{nice=0}$: Peso de un proceso con prioridad por defecto
- El planificador intenta que todos los procesos reciban el mismo *vruntime*
  - Se ejecuta el proceso que lleva más tiempo esperando (mínimo *vruntime*)

CFS: Siguiente proceso a ejecutar

- CFS mantiene una "lista" de procesos (por cada CPU) ordenada ascendentemente por *vruntime*
  - Una *run queue* por CPU, formada por los BCPs de procesos asignados a esa CPU (posiblemente con distinta prioridad)
- Por motivos de eficiencia, se usa un *Red-black tree* para implementar la lista ordenada:
  - Árbol equilibrado: operaciones $O(logN)$



CFS: Visión global

Resumen del algoritmo

- A medida que un proceso se ejecuta, su *vruntime* se incrementa (en base a su prioridad)
  - El *vruntime* permanece constante mientras el proceso espera en la *run queue*
- Cuando un proceso P se ha ejecutado durante *min_granularity* ms sin ser expropiado, CFS comprueba periódicamente si el proceso *merece* seguir en la CPU o no:
  - Si $vruntime(P) > min\_vruntime\_en\_run\_queue \rightarrow$ expropiación
- Cuando un proceso es expropiado, CFS selecciona para ejecutar el proceso con el mínimo vruntime en la *run queue*