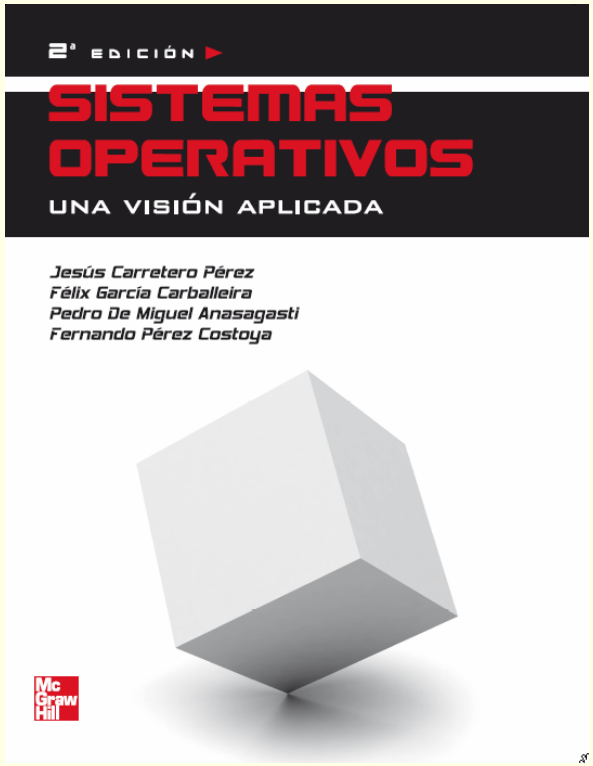
Sistemas operativos
2ª edición

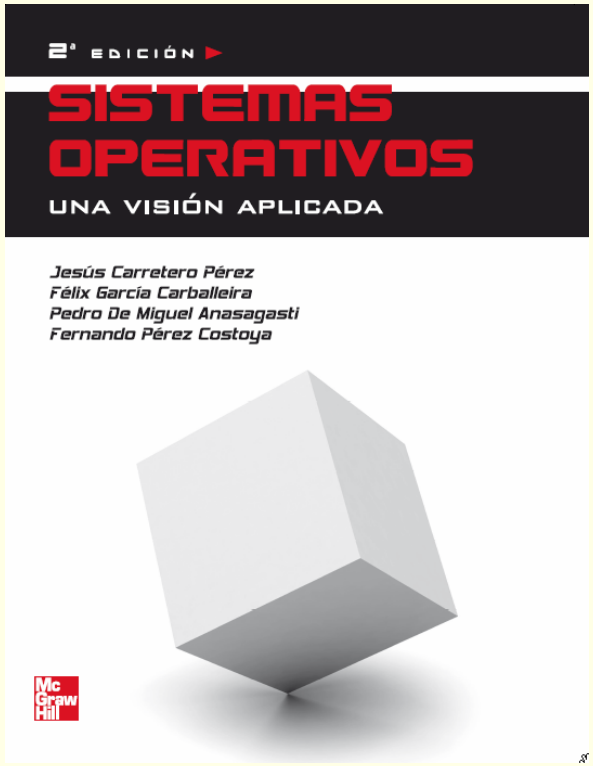


Capítulo 4
Planificación del procesador

Organización del tema

- Primera parte
  - Aspectos generales de la planificación
  - Planificación en sistemas monoprocesador
  - Planificación monoprocesador en Linux
- Segunda parte
  - Planificación en sistemas multiprocesador
  - Planificación multiprocesador en Linux
  - Planificación de aplicaciones paralelas
- Tercera parte
  - Planificación de máquinas virtuales

Sistemas operativos
2ª edición



Capítulo 4
Planificación del procesador
1ª parte: planificación en monoprocesador

Contenido

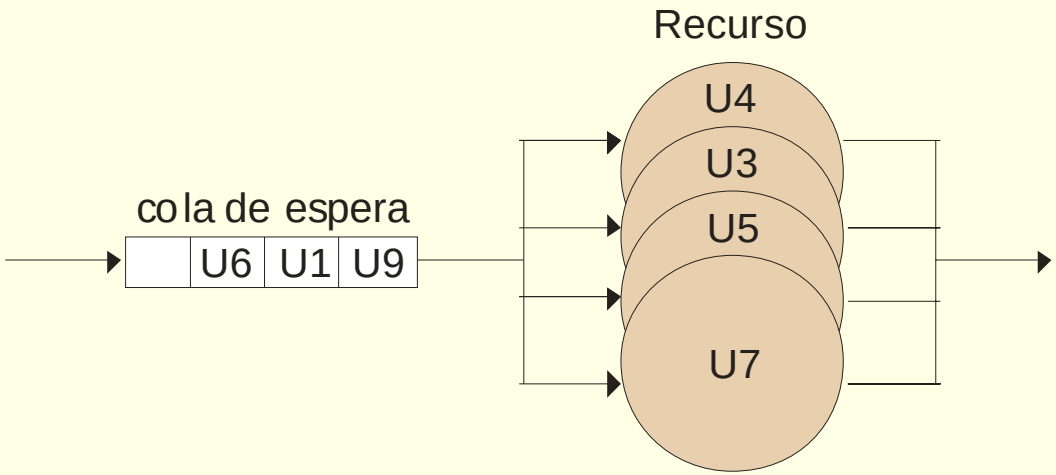
- Introducción
- Caracterización de los procesos
- Objetivos de la planificación
- Niveles de planificación
- Algoritmos de planificación
- Planificación monoprocesador en Linux

Introducción

- SO planifica recursos: UCP recurso más importante
- Planificación del procesador
  - Aparece con multiprogramación
  - Sistemas por lotes: aprovechamiento de UCP
  - Tiempo compartido: reparto equitativo entre usuarios
  - PC: interactividad; buen tiempo de respuesta
  - Multiprocesadores: aprovechar paralelismo
  - Tiempo real (no crítico): cumplir plazos
- Planificación en S.O. de propósito general: reto complejo
  - Variedad de plataformas: de empotrados a NUMA
  - Variedad de cargas: servidor, interactivo, científico, etc.
- NOTA: En S.O. actual la entidad planificable es el *thread*
  - Pero "por tradición" seguimos hablando de procesos

El problema general de la planificación

- Recurso con múltiples ejemplares utilizado por varios usuarios
- Planificación: qué ejemplar se asigna a qué usuario



Aspectos generales de la planificación

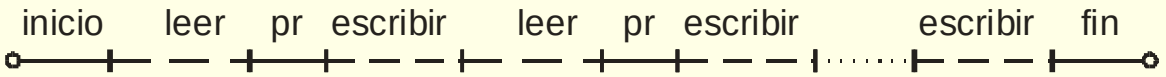
- Objetivos generales:
  - Optimizar uso
  - Minimizar tiempo de espera
  - Ofrecer reparto equitativo
  - Proporcionar grados de urgencia
- Tipos de planificación: expulsiva versus no expulsiva
- Afinidad a subconjunto de ejemplares de recurso
  - Estricta: pedida por el usuario
  - Natural: favorece rendimiento

Caracterización de los procesos

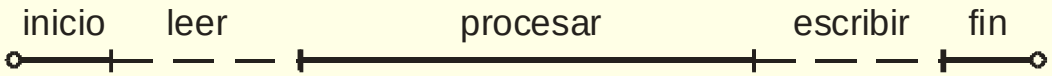
- Perfil de uso del procesador
  - Mejor primero proceso con ráfaga de UCP más corta
- Grado de interactividad
  - Asegurar buen tiempo de respuesta de interactivos
- Nivel de urgencia (o importancia)
  - Especialmente importante en sistemas de tiempo real
- *Threads* (entidad planificable) no son independientes:
  - Incluidos en procesos
  - Que pueden ser parte de aplicaciones
  - Que pertenecen a usuarios
  - Que pueden estar organizados en grupos

Uso intensivo de la E/S versus la UCP

```
Programa

inicio();

Repetir
    leer(fichero_entr, dato);
    /* procesado sencillo */
    res=pr(dato);
    escribir(fichero_sal, res);
hasta EOF(fichero_entr);

fin();
```




```
Programa

inicio();

leer(fichero, matriz);

/* procesado complejo */
procesar(matriz);

escribir(fichero, matriz);

fin();
```



Reparto equitativo: ¿para quién?

- ¿Para grupos, usuarios, aplicaciones, procesos o *threads*?
- Ejemplo con dos usuarios U y V
  - U ejecuta aplicación con 1 proceso que incluye 1 *thread* (t1)
  - V ejecuta dos aplicaciones
    - Una con 2 procesos con un solo *thread* cada uno (t2 y t3)
    - Otra con 1 un único proceso con dos *threads* (t4 y t5)
  - Equitatividad para:
    - *Threads*: cada *thread* 20% de procesador
    - Procesos: t1, t2 y t3 25%; t4 y t5 12,5%
    - Aplicaciones: t1 33,33%; otros *threads* 16,67%
    - Usuarios: t1 50%; otros *threads* 12,5%
- Conveniencia de esquema general configurable
- *Fair-share schedulers*

Objetivos de la planificación del procesador

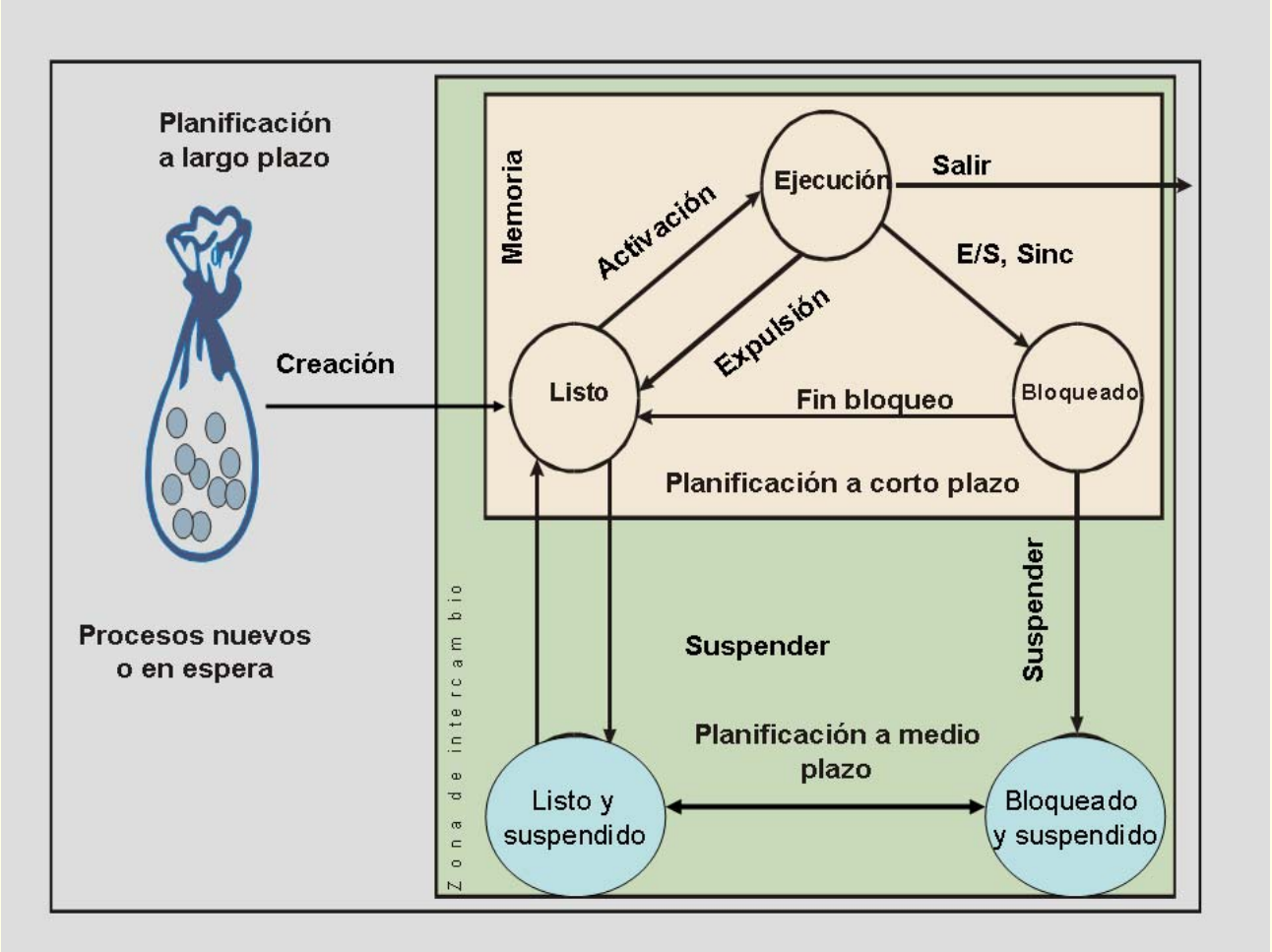
- Optimizar el comportamiento del sistema.
- Diferentes parámetros (a veces contrapuestos)

- Parámetros por proceso:
  - **Tiempo de ejecución**
  - **Tiempo de espera**
  - **Tiempo de respuesta**
- Parámetros globales:
  - **Uso del procesador**
  - **Tasa de trabajos completados**
  - **Equitatividad**

Tipos de planificadores

- A largo plazo (planificador de trabajos)
  - Control de entrada de trabajos al sistema
  - Característico de sistemas por lotes
  - No presente habitualmente en SO de propósito general
- A medio plazo
  - Control carga (suspensión procesos) para evitar hiperpaginación
  - Habitual en SS.OO. de propósito general
    - Aunque Linux ha usado otras opciones (p.e. *swap token*)
- A corto plazo: el planificador propiamente dicho
  - Qué proceso en estado de listo usa el procesador

Tipos de planificadores



Múltiples niveles de planificación

- Hasta 3 niveles: en sistema con hilos de usuario y hipervisor tipo I
  - Planificador de hilos de usuario
    - Hilos usuario del proceso sobre hilos núcleo asignados al mismo
    - Incluido en biblioteca o *runtime* de un lenguaje
  - Planificador de hilos de núcleo
    - Hilos de núcleo del SO sobre UCPs virtuales asignadas al mismo
    - Englobado en el SO
  - Planificador de UCPs virtuales (*vCPU*)
    - UCPs virtuales sobre UCPs físicas asignadas a esa máquina virtual
    - Forma parte del hipervisor

Puntos de activación

- Puntos del SO donde puede invocarse el planificador :
  1. Proceso en ejecución finaliza
  2. Proceso realiza llamada que lo bloquea
  3. Proceso realiza llamada que desbloquea proceso más urgente
  4. Interrupción desbloquea proceso más urgente
  5. Proceso realiza llamada declarándose menos urgente
  6. Interrupción de reloj marca fin de rodaja de ejecución
- Dos tipos de algoritmos:
  - no expulsivos: sólo C.C. voluntarios (1 y 2)
  - expulsivos: además C.C. involuntarios (3, 4, 5 y/o 6)
- No confundir con núcleo expulsivo o no expulsivo

Algoritmos de planificación

- Primero en llegar primero en ejecutar (FCFS)
- Primero el trabajo más corto (SJF/SRTF)
- Planificación basada en prioridades
- *Round robin* (turno rotatorio)
- Lotería
- Colas multinivel

- Algoritmo real es mezcla de teóricos + ajustes empíricos

Primero en llegar primero en ejecutar (FCFS)

- Selección: Proceso que lleva más tiempo en cola de listos
- Algoritmo no expulsivo
- Fácil de implementar (O(1)):
  - Cola de listos gestionada en modo FIFO
- Satisfacción de objetivos generales:
  - ✓ Optimizar uso
  - ↓ Minimizar tiempo de espera (efecto convoy)
  - ↓ Ofrecer reparto equitativo
  - ↓ Proporcionar grados de urgencia

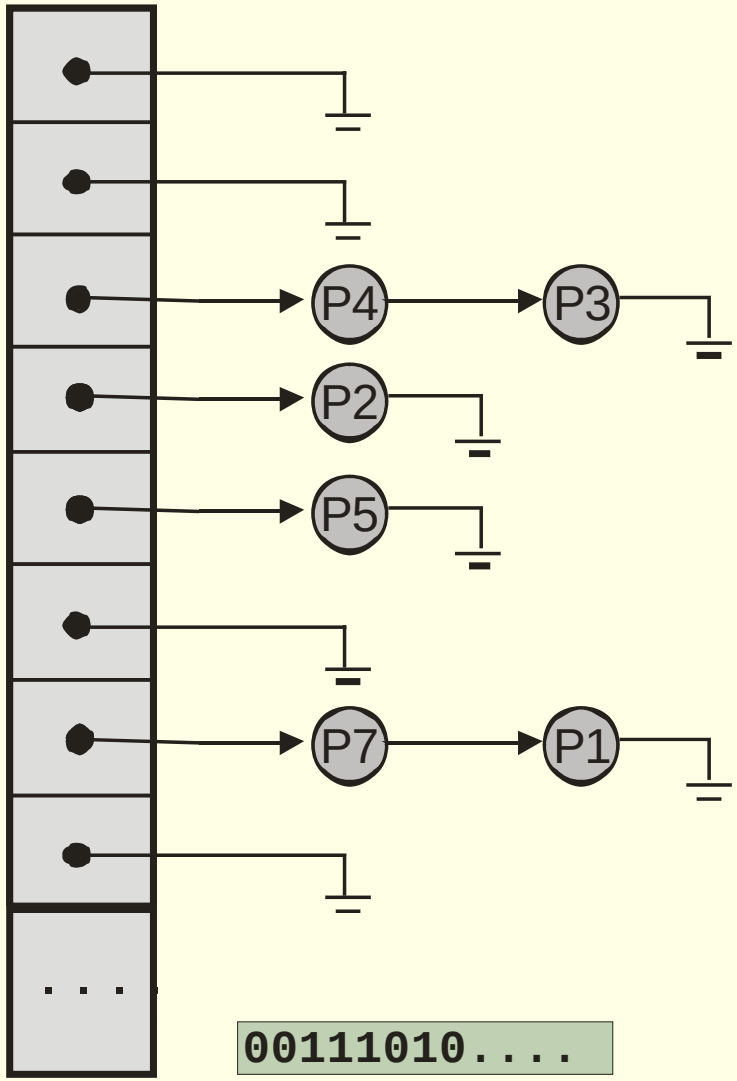
Primero el trabajo más corto (SJF)

- Selección: Proceso listo con próxima ráfaga UCP más corta
- Versión no expulsiva: Sólo si proceso se bloquea o termina
- Versión expulsiva: 1º el de menor tiempo restante (SRTF)
  - También se activa si proceso pasa a listo (puntos 3 y 4)
- ¿Cómo se conoce a priori la duración de la próxima ráfaga?
  - Estimaciones a partir de las anteriores
- Puede producir inanición
- Implementación:
  - O(N) si usa lista de procesos; O(log N): árbol binario
- Punto fuerte:
  - ✓ Minimizar tiempo de espera

Planificación por prioridad

- Cada proceso tiene asignada una prioridad
- Selección: Proceso en cola de listos que tenga mayor prioridad
- Existe versión no expulsiva y expulsiva
  - Si proceso pasa a listo o actual baja su prioridad (3, 4 y 5)
- Las prioridades pueden ser estáticas o dinámicas
- Prioridad puede venir dada por factores externos o internos
- Puede producir inanición
  - "Envejecimiento": Prioridad aumenta con el tiempo
- En general, mal esquema para la equitatividad
- Implementación:
  - Nº de prioridades fijo y reducido: O(1)
  - Sino: O(N) si usa lista de procesos; O(log N): árbol binario
- Punto fuerte:
  - ✓ Proporcionar grados de urgencia

Planificación por prioridad: implementación O(1)



Turno rotatorio (*Round Robin*, RR)

- FCFS + plazo (rodaja o cuanto)
- Algoritmo expulsivo pero sólo con fin del cuanto (punto 6)
- Tiempo de respuesta acotado (nº procesos listos * tam. rodaja)
- Tamaño rodaja: grande (→FIFO) vs. pequeño (sobrecarga)
  - Más pequeño → menor t. respuesta y mejor equitatividad
  - Configurable (Windows cliente 20 ms. servidor 120 ms.)
- Tamaño rodaja de proceso varía durante vida del proceso
  - Windows triplica rodaja de proceso al pasar a primer plano
- Más general: Round-robin con pesos (WRR)
  - Tamaño rodaja proporcional a peso del proceso
  - Implementar importancia de procesos en vez de con prioridad
- Round-robin por niveles para *Fair-share scheduling*
- Punto fuerte:
  - ✓ Ofrecer reparto equitativo

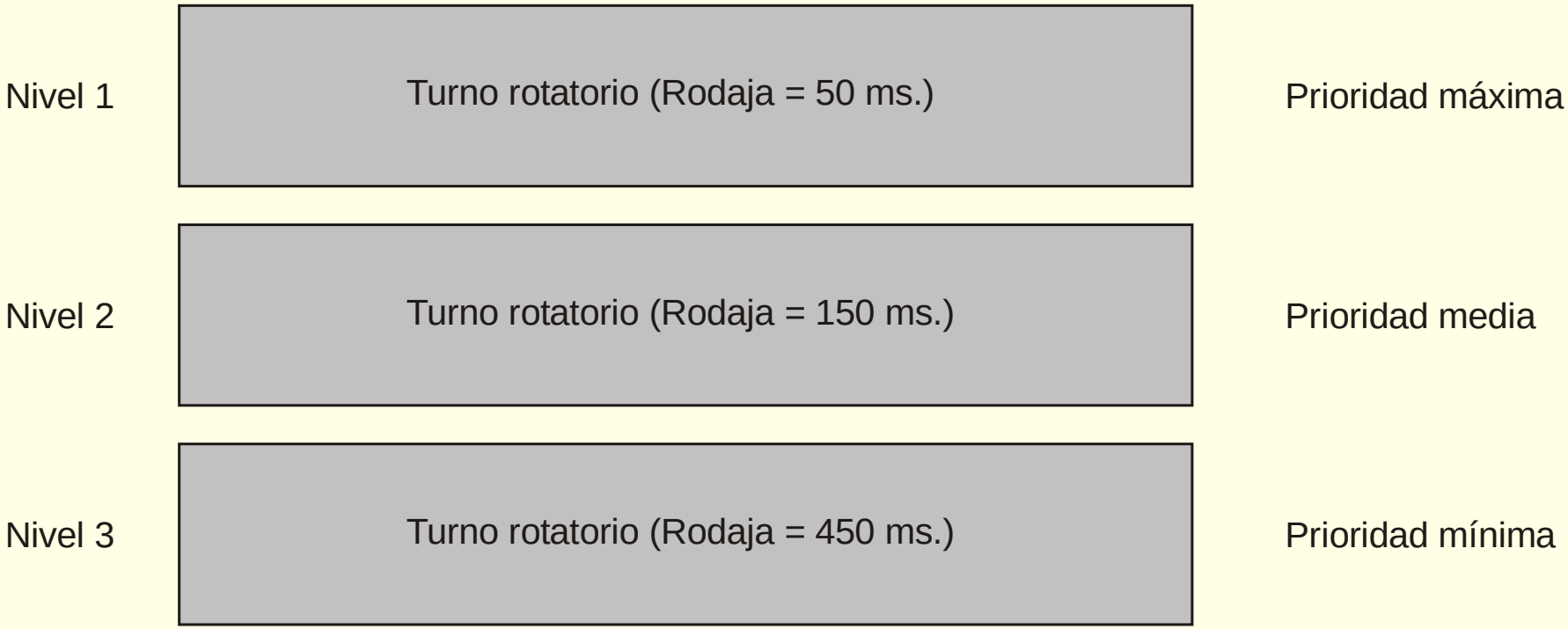
Planificación por lotería

- Cada proceso tiene uno o más billetes de lotería
- Selección: proceso que posee billete aleatoriamente elegido
- Versión no expulsiva o expulsiva (rodaja de tiempo entre sorteos)
- No inanición
- Importancia de los procesos: nº de billetes que poseen
- Favorecer procesos con E/S e interactivos
  - Billetes de compensación si proceso no consume su rodaja
- Transferencia de billetes: p.e. un cliente a un servidor
- Posibilita *Fair-share scheduling* por niveles: *ticket currencies*
- O(N) si usa lista de procesos; O(log N): árbol binario
- Satisfacción de objetivos generales:
  - ✓ Ofrecer reparto equitativo

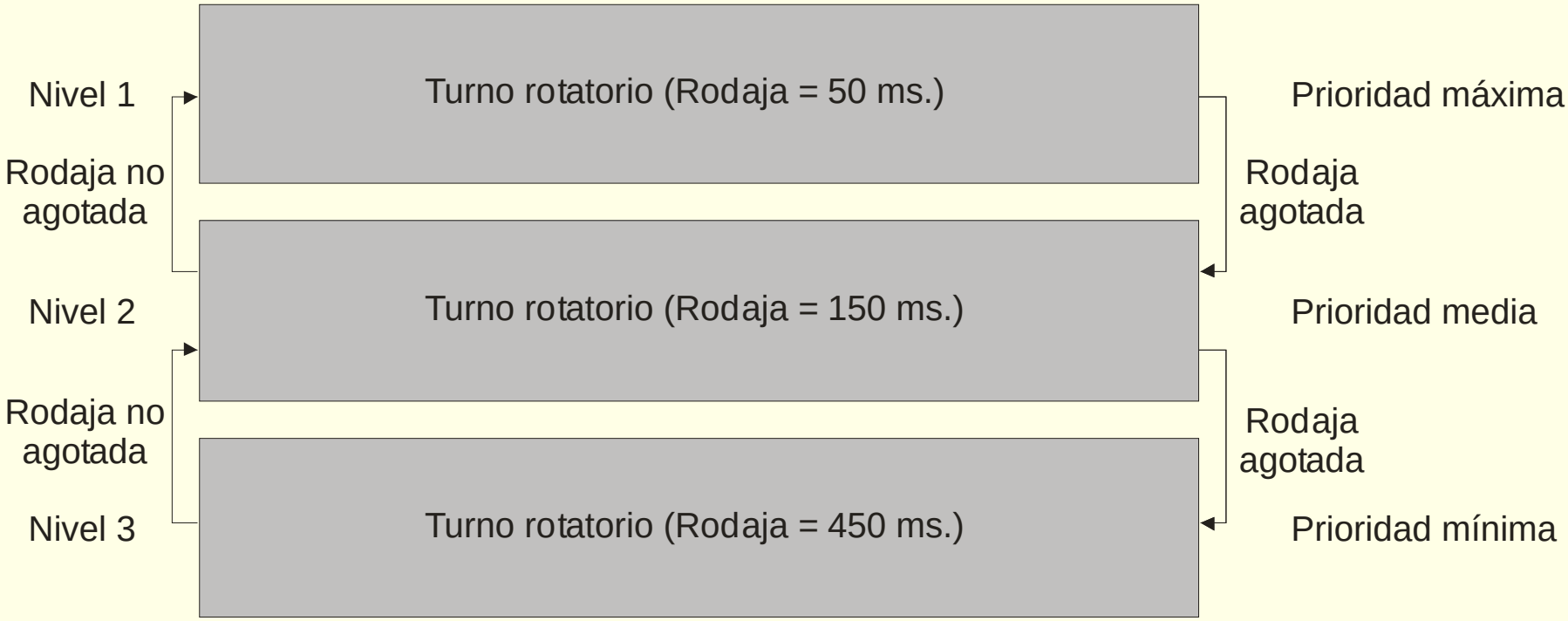
Colas multinivel

- Generalización: Distinguir entre clases de procesos
- Parámetros del modelo:
  - Número de niveles (clases)
  - Algoritmo de planificación de cada nivel
  - Algoritmo de reparto del procesador entre niveles
- Colas con o sin realimentación:
  - Sin: proceso en la misma cola durante toda su vida
  - Con: proceso puede cambiar de nivel

Colas multinivel sin realimentación



Colas multinivel con realimentación



Planificación uniprocesador en Linux

- EMHO planificador no es la pieza más brillante de Linux
  - O(N) hasta versión 2.6
  - Demasiados parámetros configurables
  - No exento de polémica (Kolivas)
- Planificador: módulo relativamente cambiante
  - Es una función de selección; No ofrece API a las aplicaciones
- Estudiaremos dos esquemas de planificación de Linux:
  - Planificador O(1): introducido en la versión 2.6
  - *Completely Fair Scheduler* (CFS) introducido a partir de 2.6.23
- Linux da soporte extensiones t. real POSIX
  - Cola multinivel sin realimentación: t. real (no crítico) y normal
    - Procesos t. real siempre más prioritarios que normales
  - Clase de t. real (SCHED_FIFO/SCHED_RR) ya estudiada
  - Nos centramos en clase normal (SCHED_OTHER)

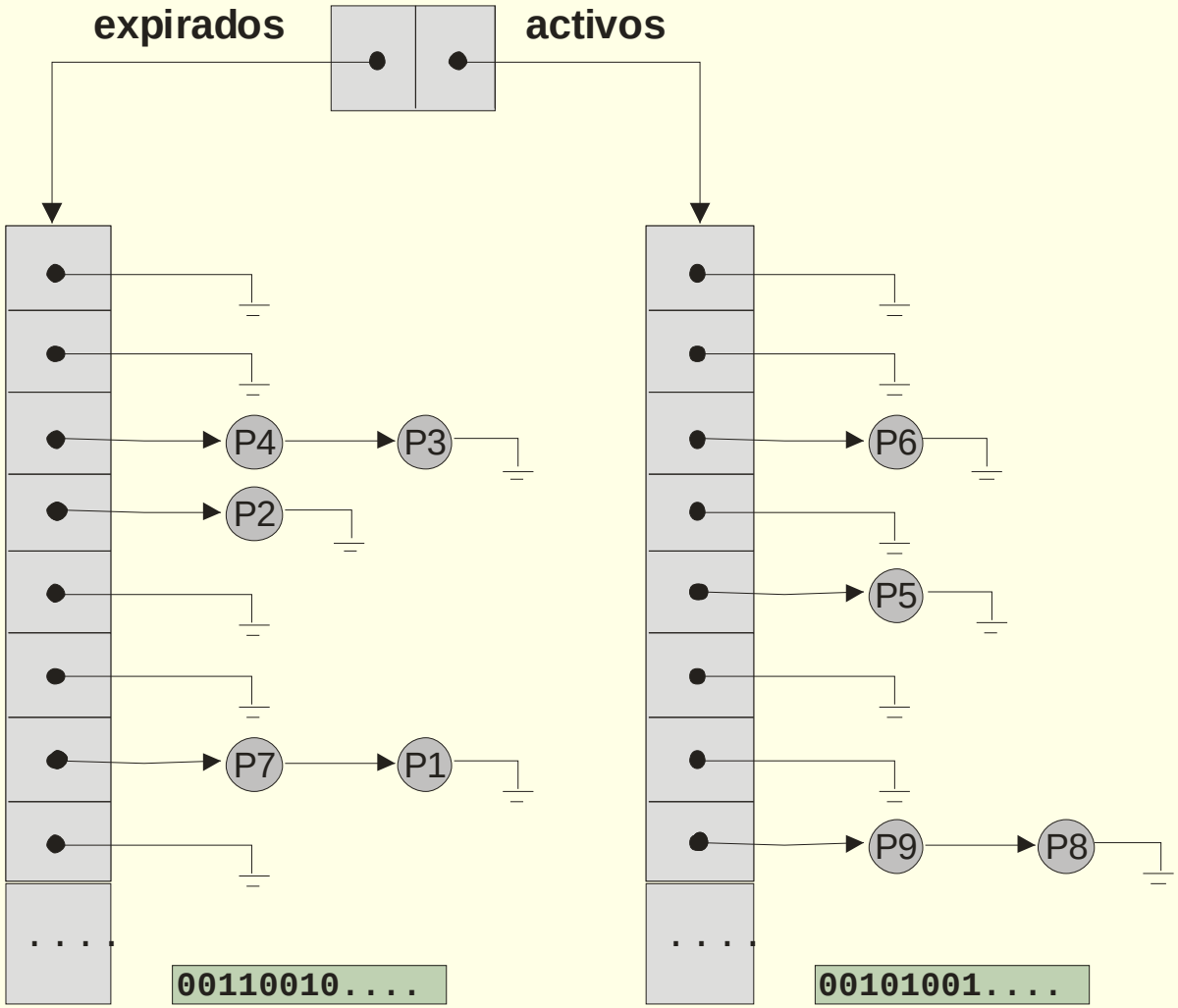
Planificador de Linux O(1)

- Mezcla de prioridades y round-robin
- Esquema expulsivo con prioridad dinámica
  - Prioridad base estática (*nice*): -20 (máx.) a 19 (mín.)
  - Ajuste dinámico de -5 (mejora) a +5 (empeora)
    - Depende de uso de UCP y tiempo de bloqueo
    - Favorece procesos interactivos y con E/S y reduce inanición
- Rodaja con tamaño fijo que depende de la prioridad base
  - 5ms (prio 19); 100ms (prio 0); 800 ms (prio -20)
  - En `fork` padre e hijo se reparten la mitad de rodaja restante
- Proceso más prioritario con doble ventaja sobre menos prioritario
  - Lo expulsa y obtiene mayor porcentaje de procesador
  - ¿Proceso con prio. máx. no deja ejecutar nunca a uno con mín?

Planificador de Linux O(1)

- ¿Cómo concilia prioridad y equitatividad?:
  - 2 listas de procesos listos: activos y expirados
  - Planificador selecciona de lista de activos
  - Proceso que agota su rodaja pasa a lista de expirados
    - Excepto si se considera "interactivo" que vuelve a activos
      - A no ser que haya procesos en lista de expirados que lleven mucho tiempo en ella o más prioritarios que el que agotó la rodaja
  - Cuando se desbloquea proceso pasa a lista de activos
  - Se crean "rondas de ejecución" (épocas)
    - Procesos agotan rodaja y terminan todos en lista de expirados
    - En una ronda proceso va gastando su rodaja por tramos
    - "Fin de ronda": intercambio de listas

Planificador de Linux O(1)



Completely Fair Scheduler (CFS)

- Incorporación a Linux de un *fair-share scheduler*
  - Además arregla ciertas patologías de O(1); por ejemplo
    - En O(1) sólo 2 procesos prio. mínima → RR 5 ms → sobrecarga
      - Sin embargo, sólo 2 de prio. máxima → RR 800 ms
    - Tamaño de rodaja debería depender también de la carga
- Meta: todos los procesos usan "mismo tiempo" de UCP (*vruntime*)
  - Tiempo ponderado por factor que depende de prio. y de carga
  - Cada prio. (*nice*) tiene un peso: -20→88761, 0→1024, 19→15
- Selección: proceso i con menor *vruntime* (tratado más injustamente)
- Rodaja asignada: proporción entre su peso y el total de los listos
  - Rodaja = ($peso_i$ / peso total listos) * periodo
    - En periodo ejecutan todos proc listos con %UCP proporción a peso
    - Normalmente periodo = sched_latency
    - Influye en t. de respuesta: configurable; 6ms. por defecto en v3.13

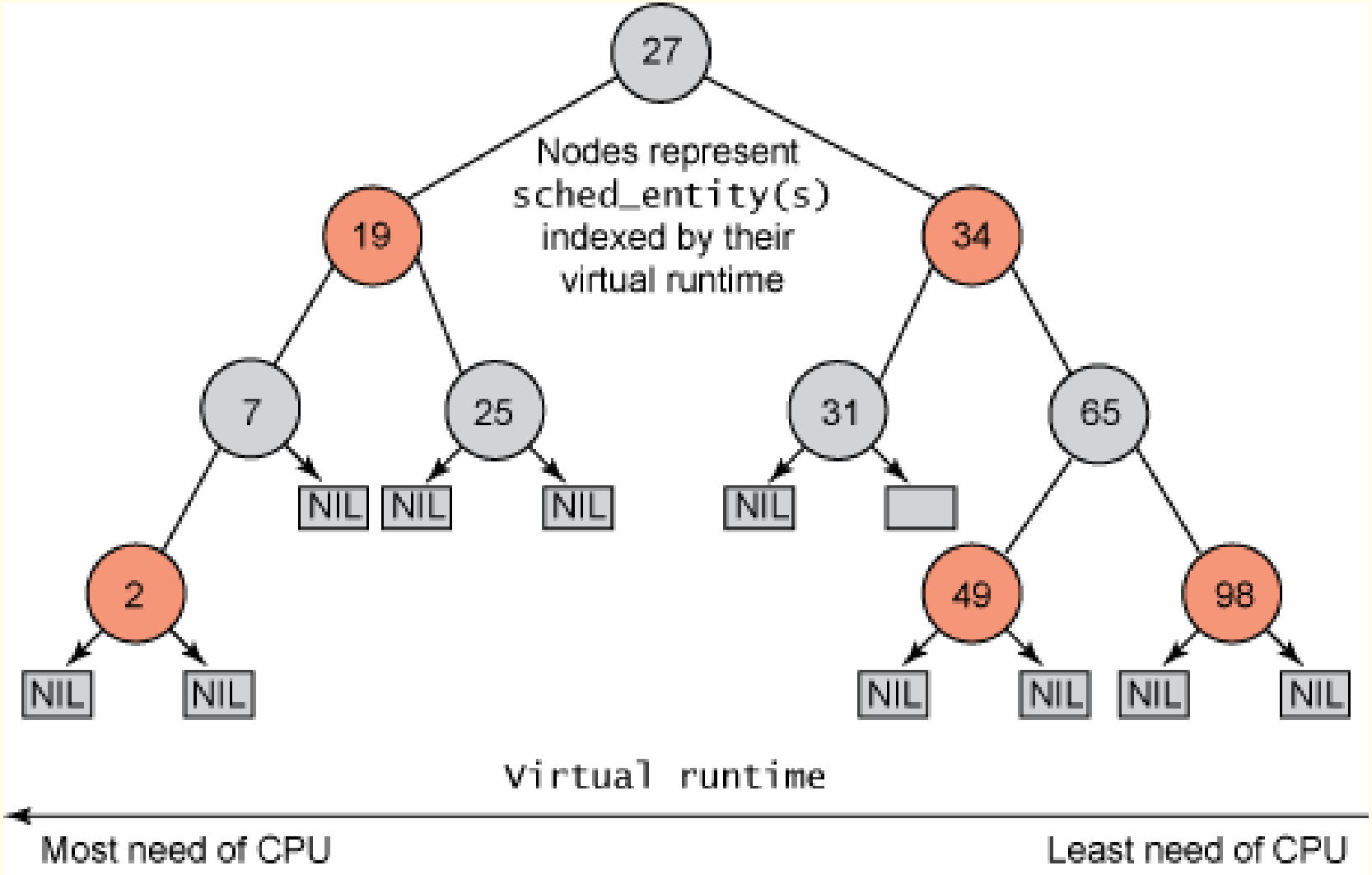
Completely Fair Scheduler (CFS)

- Pesos elegidos para que prio. se traten como valores relativos
  - Ej1. 3 procesos prio/peso: P1 0/1024, P2 1/820 y P3 2/655
    - P1 41% (2,46 ms.); P2 33% (1,97 ms.); P3 26% (1,57 ms.)
  - Ej2. 3 procesos prio/peso: P4 17/23, P5 18/18 y P6 19/15
    - P4 41% (2,46 ms.); P5 32% (1,93 ms.); P6 27% (1,61 ms.)
  - En O(1) no ocurre así (véase última transparencia)
- Cuando proceso i se bloquea o termina rodaja actualiza su vruntime
  - Añadiendo tiempo ejecutado (T) normalizado con peso prio 0
    - $vruntime_i$ += (T / $peso_i$) * peso_prio_0
    - En Ej1 si agotan rodaja todos suman 2,46 ms. a su vruntime
    - En Ej2 si agotan rodaja todos suman 109,71 ms. a su vruntime
    - Si se bloquean después de ejecutar 1 ms. cada proceso suma
      - P1 1; P2 1,24; P3 1,56; P4 44,52; P5 56,89; P6 68,27
      - Proceso con prioridad máxima (-20; peso 88761) sumaría 0,01 ms.
  - + prioridad → + "lento" corre el reloj; con prio 0 reloj real

Completely Fair Scheduler (CFS)

- Si nº listos (NL) alto → rodaja demasiado pequeña (sobrecarga)
  - sched_min_granularity mínima rodaja permitida
    - configurable; 0,75ms. por defecto en v3.13
    - sched_nr_latency = sched_latency / sched_min_granularity
      - Cuántos procesos "caben" en sched_latency (defecto 8)
  - Si NL > sched_nr_latency → periodo = sched_min_granularity * NL
    - Hay que ampliar el periodo → peor tiempo de respuesta
- Proceso nuevo se le asigna min_vruntime (el menor vruntime actual)
  - Pronto podrá usar la UCP
- Procesos con E/S e interactivos automáticamente beneficiados
  - Su vruntime se queda parado
  - Para evitar que proceso con largo bloqueo acapare UCP
    - Desbloqueo: vruntime = max(vruntime, min_vruntime - sched_latency)
- Implementación O(log N): requiere árbol binario *red-black tree*

CFS: *Red-black Tree (RBT)*



http://www.ibm.com/developerworks/library/l-completely-fair-scheduler

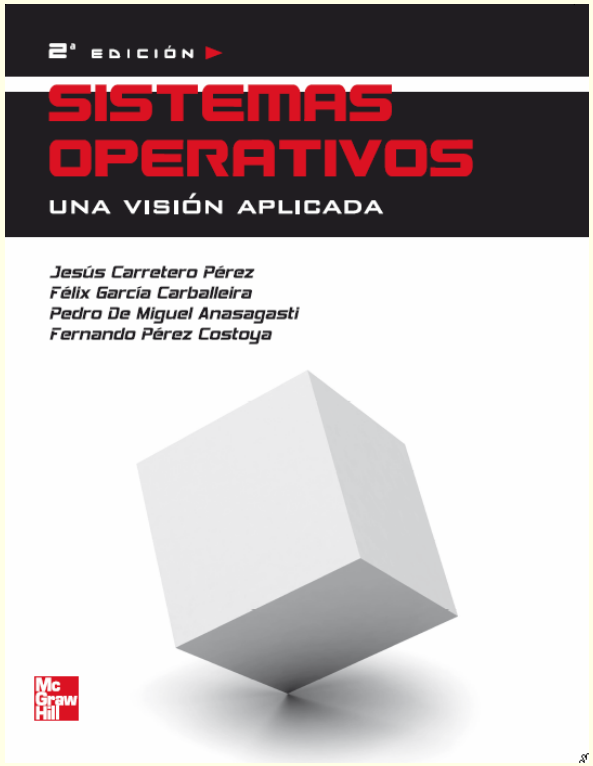
CFS: planificación de grupos

- Por defecto CFS provee *fair-share scheduling* sólo entre procesos
- Posibilidad de crear grupos de procesos formando una jerarquía
- Planificador gestiona entidades planificables: procesos o grupos
- Selección de proceso a ejecutar: empieza en RBT nivel superior
  1. Busca en RBT la entidad con menor vruntime
  2. Si es proceso → termina
  3. Si es un grupo → volver a paso 1 sobre el RBT del grupo
- Actualización se propaga hacia arriba en jerarquía
- Permite cualquier esquema de niveles de *fair-share scheduling*

CFS vs O(1): ejemplo prioridades y % uso de UCP

- CFS
  - Ej1. 3 procesos prio/peso: P1 0/1024, P2 1/820 y P3 2/655
    - P1 41% (2,46 ms.); P2 33% (1,97 ms.); P3 26% (1,57 ms.)
  - Ej2. 3 procesos prio/peso: P4 17/23, P5 18/18 y P6 19/15
    - P4 41% (2,46 ms.); P5 32% (1,93 ms.); P6 27% (1,61 ms.)
- O(1)
  - Ej1. 3 procesos prio/rodaja: P1 0/100ms, P2 1/95 y P3 2/90
    - P1 35% (100 ms.); P2 33% (95 ms.); P3 32% (90 ms.)
  - Ej2. 3 procesos prio/rodaja: P4 17/15ms, P5 18/10 y P6 19/5
    - P4 50% (15 ms.); P5 33% (10 ms.); P6 17% (5 ms.)

Sistemas operativos
2ª edición



Capítulo 4
Planificación del procesador
2ª parte: planificación en multiprocesadores

Contenido

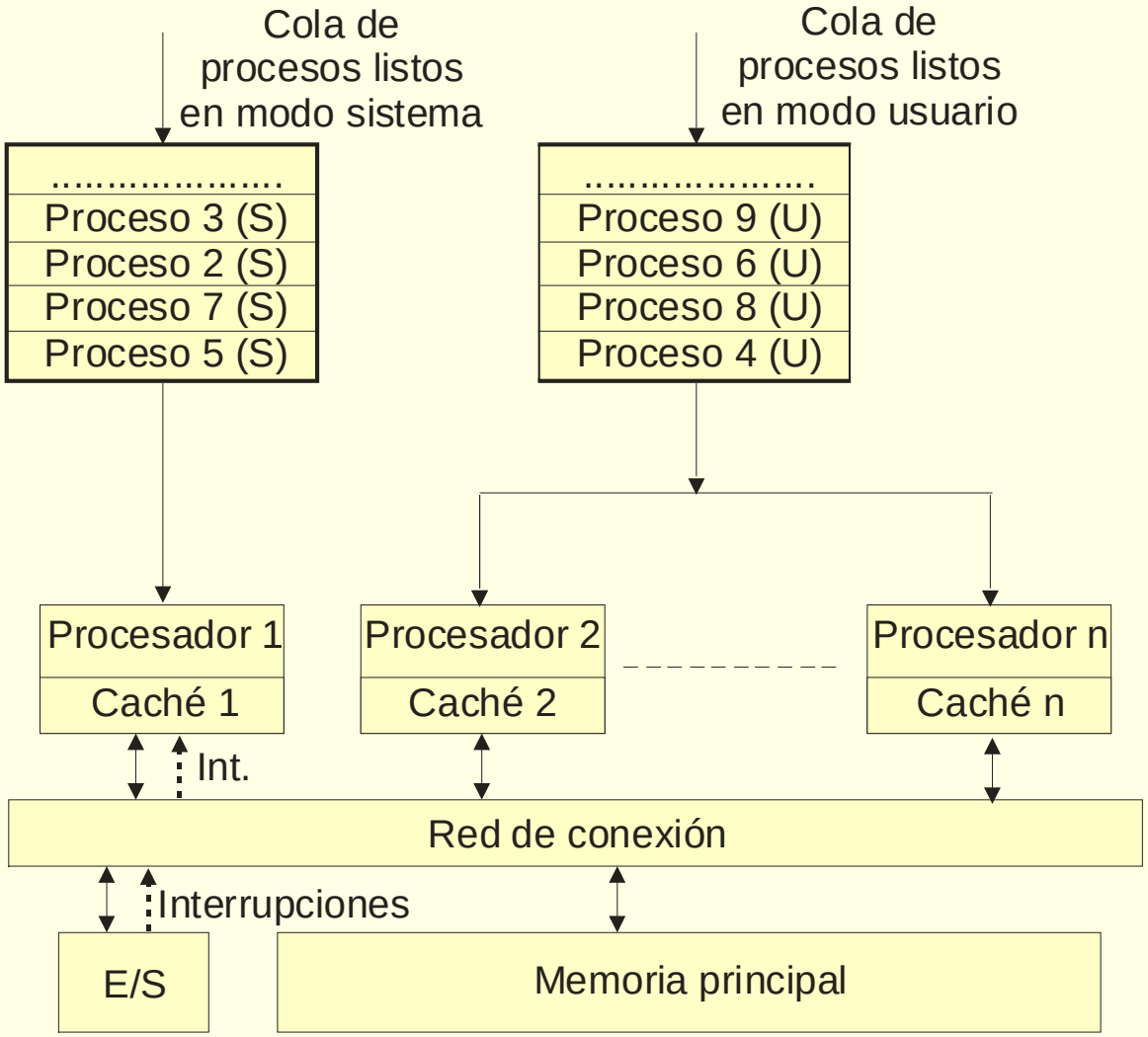
- ASMP *versus* SMP
- Planificación en multiprocesadores
- Planificación con cola única
- Sistema multiprocesador jerárquico
- Planificación con una cola por procesador
- Planificación de multiprocesadores en Linux

- Planificación de aplicaciones paralelas en multiprocesadores
- Planificación en sistemas distribuidos
- Planificación de aplicaciones paralelas en sistemas distribuidos

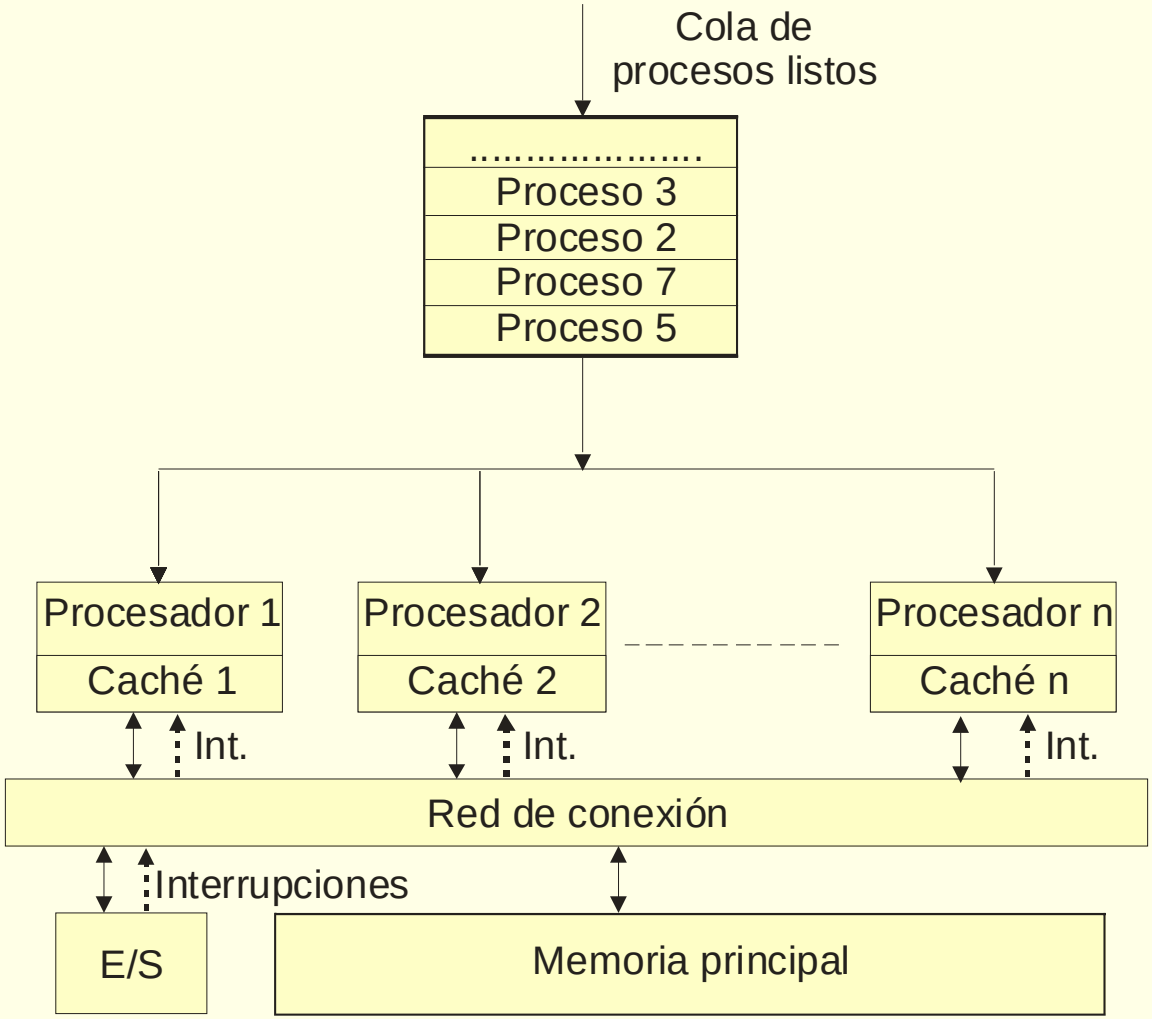
Multiprocesamiento asimétrico vs. simétrico

- Difícil adaptar SO de UP para MP
  - Concurrencia se convierte en paralelismo real
- Solución de compromiso: Multiprocesamiento asimétrico (ASMP)
  - Simetría en hardware pero no en software
  - SO sólo se ejecuta en UCP maestra
    - Llamadas al SO, excepciones e interrupciones en esa UCP
  - Se convierte en "cuello de botella": SO no escalable
  - Beneficioso sólo para programas paralelos que usan poco el SO
- Solución definitiva: Multiprocesamiento simétrico
  - SO se ejecuta en cualquier UCP
    - Llamadas al SO y excepciones en UCP donde se producen
    - Interrupciones en UCP que las recibe

Multiprocesamiento asimétrico (ASMP)



Multiprocesamiento simétrico (SMP)



Planificación en multiprocesadores

- Trivial: *N* UCP ejecutan *N* procesos elegidos por planificador
- Sí, pero hay que tener en cuenta:
  - Afinidad natural y estricta
  - Multiprocesadores jerárquicos (SMT, CMP, NUMA, ...)
    - Compartimiento de recursos entre algunos procesadores
  - Evitar congestión en operación del planificador
    - P.e. debida al uso de cerrojos al acceder a cola de listos
  - Además de rendimiento puede haber otros parámetros
    - P.ej. minimizar consumo (p.e. en un portátil)
- 2 esquemas: Cola única vs. Una cola/procesador
  - Linux a partir de versión 2.6: uso de una cola/UCP
  - Windows cola única para cliente y cola/UCP para servidor

Planificación en MP con cola única

- UCP elige qué proceso de la cola ejecuta (autoplanificación)
- Afinidad natural: mejor ejecutar en misma UCP
  - Aprovecha información en caché
- Planificación:
  - UCP queda libre (CCV o CCI):
    - Planificador elige proceso (llamémosle "el más prioritario")
      - Prioridad matizada por la afinidad natural
      - Asigna UCP a proceso afín aunque con prioridad un poco menor
      - Si afín se le asigna un *bonus* extra a la prioridad
  - Proceso *P* pasa a listo (desbloqueo o nuevo): se le asigna
    1. UCP afín libre
    2. Cualquier UCP libre
    3. UCP con proceso *Q* tal que prio(*P*)>prio(*Q*)
       - Prioridad matizada por afinidad natural si desbloqueo (*bonus* extra)
  - Uso de **int. SW de planificación** con IPI para forzar CCI

Afinidad estricta

- Planificación debe respetar afinidad estricta
- Proceso informa de qué UCPs desea usar
- Cambios en el esquema de planificación
  - Proceso pasa a listo: Sólo UCPs en su afinidad estricta
  - UCP queda libre: Sólo procesos que incluyan a esa UCP
  - Cambio de afinidad de proceso puede causar planificación

- Servicio POSIX para el control de afinidad estricta

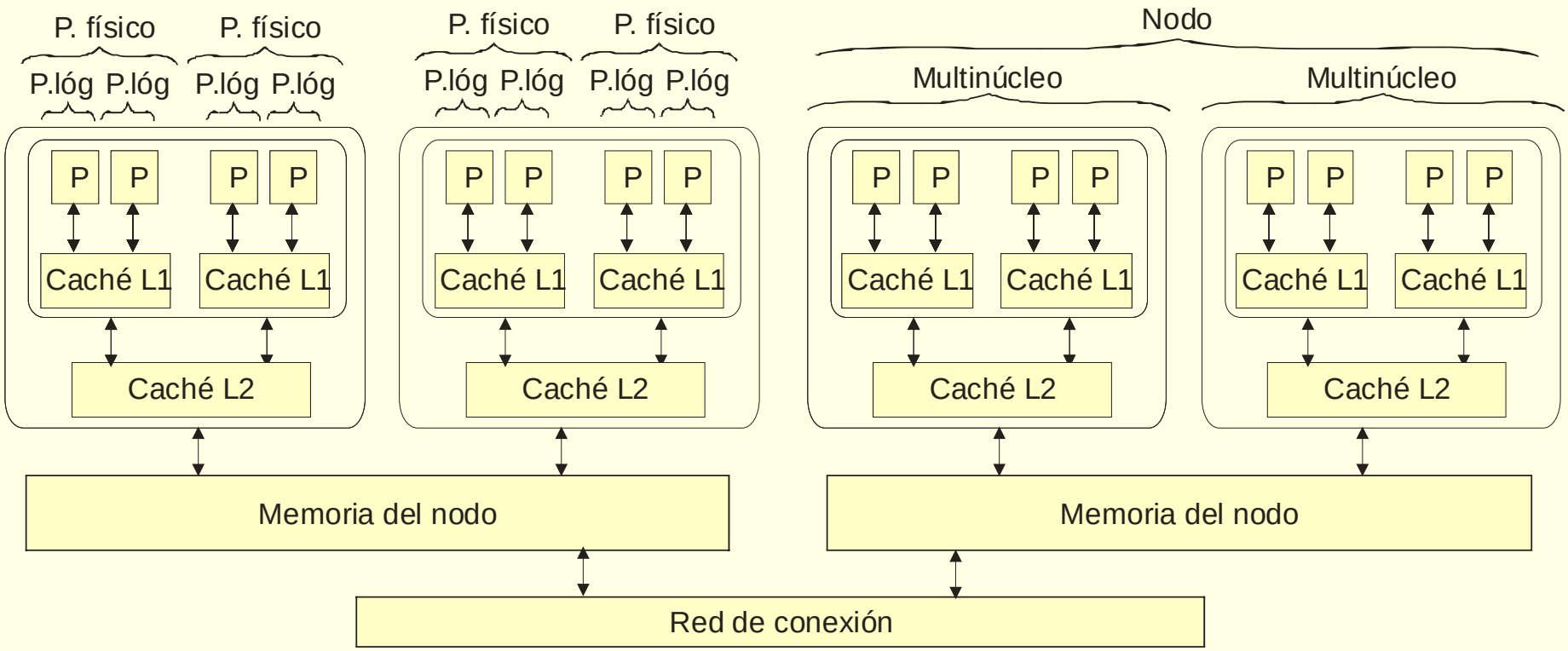
  int **sched_setaffinity**(pid_t pid, unsigned int longit, cpu_set_t *máscara)
  - *máscara* define en qué UCPs puede ejecutar *pid*
  - Usada por mandato `taskset` de Linux
- Servicio Windows para el control de afinidad estricta

  BOOL **SetProcessAffinityMask**(HANDLE hpr, DWORD_PTR máscara)
  - *máscara* define en qué UCPs puede ejecutar proceso

Sistema multiprocesador jerárquico

- Multiprocesador no es un conjunto de UCPs al mismo nivel
  - Multiprocesador tiene carácter jerárquico
- Sistema NUMA con múltiples nodos
- UCP multinúcleo (CMP: *Chip MultiProcessing*)
- Cada núcleo con SMT (*Simultaneous Multithreading*)
- Algunas UCP no independientes: pueden compartir recursos
  - UCP lógicas $\in$ mismo núcleo $\rightarrow$ caché L1
  - núcleos $\in$ mismo multinúcleo $\rightarrow$ caché L2
- Incluso relacionados con consumo de energía
  - núcleos $\in$ mismo multinúcleo $\rightarrow$ consumo de energía común
- ¿Afecta esta jerarquía al SO? ¿Y a la planificación?

Sistema multiprocesador jerárquico



NUMA con 2 nodos: 2 multinúcleo/nodo con 2 núcleos y 1 p. lógico/núcleo

Ejemplos de configuraciones MP para Linux

CPUs	Visible CPUs	Memory	Description
2 cores	2	2GB	Low end x86 desktop system 2008
2 cores x 2 threads x 2 sockets	8	4-8GB	Middle-end x86 desktop system 2009
4 cores x 2 threads x 2 sockets	16	8-32GB	Standard low end x86 server 2009
6 cores x 4 sockets	24	32-128GB	Standard 4 socket x86 server 2009
8 cores x 2 threads x 4 sockets	64	128-512GB	Standard 4 socket x86 server 2010
8 cores x 2 threads x 8 sockets	128	128GB-1TB	8 socket x86 server 2010
2 cores x 32 sockets	64	512GB-2TB	High end commercial server 2008
2 cores x 512 sockets	1024	>1TB	Super computer 2007

A. Kleen. "Linux multi-core scalability". *In Proceedings of Linux Kongress*, Octubre 2009.

Sistema operativo para multiprocesador jerárquico

- SO actual debe ser consciente de jerarquía de MP
- CONFIG_SMP

  http://lxr.free-electrons.com/source/kernel/sched/core.c?v=3.19#L411
- CONFIG_NUMA

  http://lxr.free-electrons.com/source/kernel/sched/core.c?v=3.19#L6187
- CONFIG_SCHED_MC

  http://lxr.free-electrons.com/source/kernel/sched/core.c?v=3.19#L6318
- CONFIG_SCHED_SMT

  http://lxr.free-electrons.com/source/kernel/sched/core.c?v=3.19#L6315

Planificación en multiprocesador jerárquico (1/2)

- Compartimiento de recursos entre algunos procesadores
  - Afecta a afinidad natural: Extensión de afinidad a la jerarquía
  - Afecta a asignación de UCPs libres a proc nuevos (sin afinidad)
    - En MP no jerárquico: vale cualquier UCP libre
- Jerarquía de afinidades
  - SMT: Afinidad a núcleo
    - Mejor ejecutar en UCP lógica $\in$ mismo núcleo
  - CMP: Afinidad a multinúcleo
    - Mejor ejecutar en núcleo $\in$ mismo multinúcleo
  - NUMA: Afinidad a nodo
    - Mejor ejecutar en mismo nodo
- Prioridad matizada por la afinidad natural
  - *Bonus* dependiendo de nivel de afinidad (SMT>CMP>NUMA)

Planificación en multiprocesador jerárquico (2/2)

- Asignación de procesadores libres a nuevos procesos
  - Si 2 UCP comparten: potencia total < 2*potencia/UCP
    - 2 UCP lógicas ∈ mismo núcleo; 2 núcleos ∈ mismo multinúcleo
  - Reparto teniendo en cuenta grado de independencia
    - Mejor ir ocupando UCPs con mayor grado de independencia
    - Creación de nuevo proceso
      - Buscar UCP lógica libre en núcleo libre de multinúcleo libre
  - Aunque para minimizar consumo puede ser mejor lo contrario:
    - Mejor usar núcleo libre de multinúcleo ocupado que de libre
      - Permite mantener multinúcleo libre en bajo consumo
  - Linux: administrador decide qué prima (consumo vs. eficiencia)

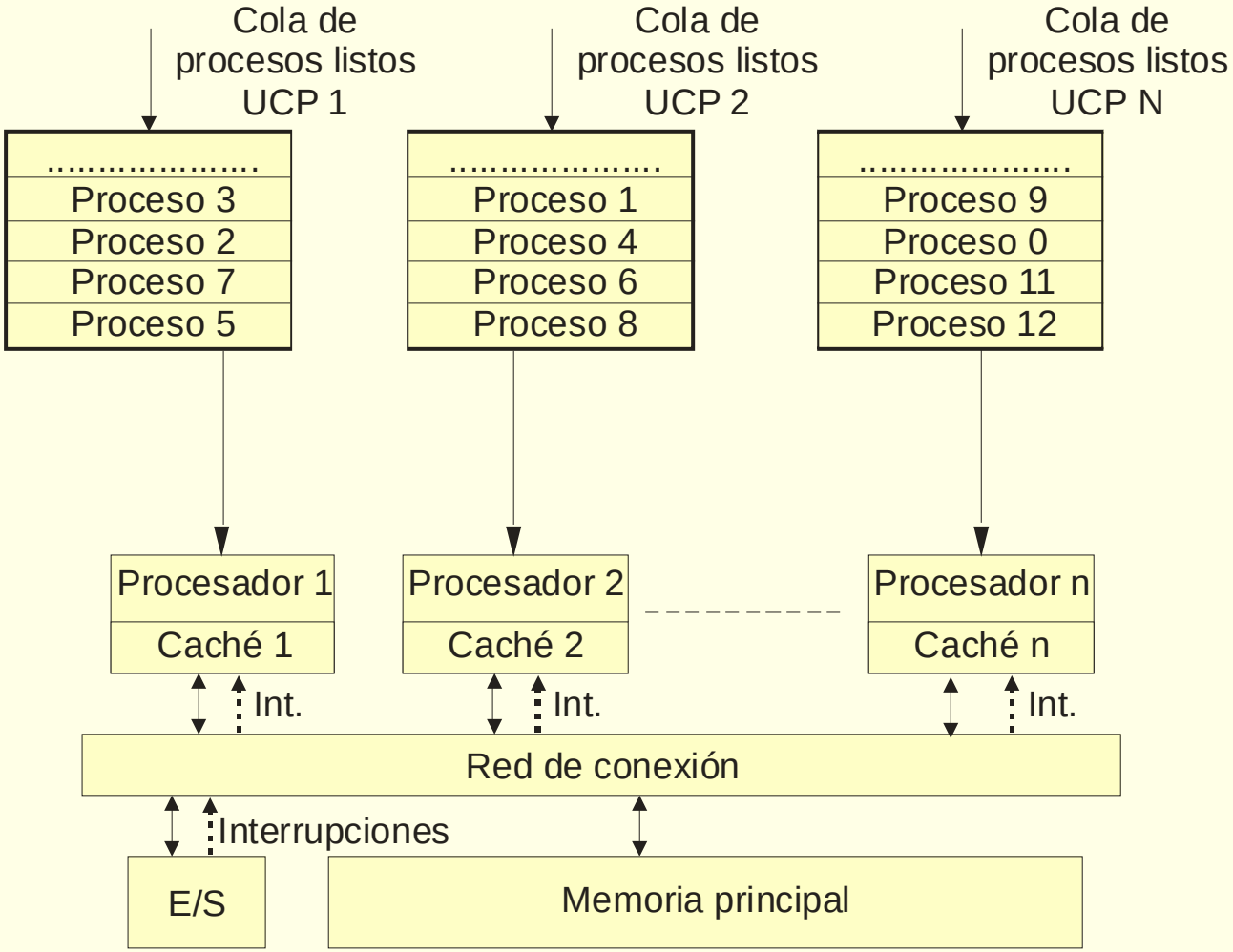
    echo 1 > /sys/devices/system/cpu/sched_mc_power_savings

    http://lxr.free-electrons.com/source/kernel/sched/core.c?v=3.19#L6315

Planificación en MP con una cola por UCP

- Cola única:
  - Accesos a cola requieren cerrojo: mala escalabilidad
  - Limitado aprovechamiento de la afinidad natural
    - Procesos cambian de UCP → "*cache line bouncing*"
- Cola por UCP: UCP se planifica de forma independiente
  - No hay congestión por cerrojo y se aprovecha mejor afinidad
- ¿En qué UCP inicia ejecución nuevo proceso?
  - En MP procesos compiten por espacio en las diversas cachés
  - Meta: carga equilibrada en el sistema → UCP menos cargada
  - Aplicando jerarquía: Procesador seleccionado corresponde a
    - Nodo menos cargado (N)
    - Multinúcleo (M) menos cargado de N
    - Procesador físico (F) menos cargado de M
    - Procesador lógico (L) menos cargado de F

Multiprocesamiento simétrico (SMP) cola/UCP



Asignación de procesador en UNIX

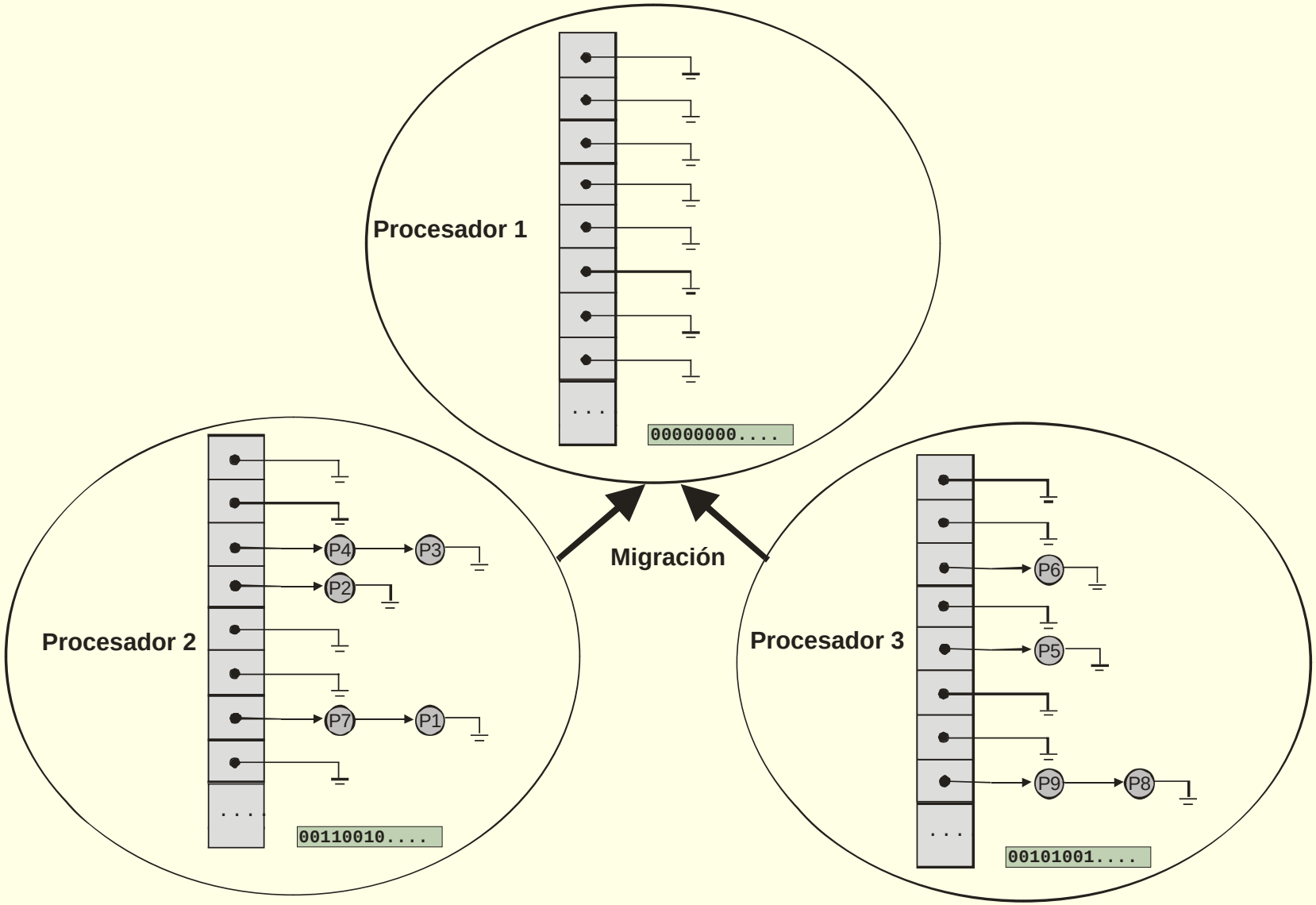
- Modelo de procesos de UNIX conlleva 3 puntos de decisión:
  - exec. Pérdida total de afinidad
    - Puede ser buen momento para migrar
    - Buscar procesador menos cargado aplicando jerarquía
  - pthread_create.
    - Mantiene afinidad
      - Razonable asignar mismo procesador
      - Aunque pierde paralelismo
      - Puede aplicarse jerarquía de afinidades
  - fork.
    - Situación intermedia: Afinidad pero hasta COW
      - ¿Mismo procesador o no?
    - En Linux configurable (SD_BALANCE_FORK):

    http://lxr.linux.no/linux+v3.13.5/include/linux/sched.h#L781

Migración de procesos

- Mecanismo de equilibrado de carga debe ser explícito
  - Migración de procesos ante desequilibrios
- Periódicamente o si cola de UCP queda vacía
- Carácter jerárquico del multiprocesador
  - Por afinidad, mejor migrar entre UCPs que pertenezcan al:
    1. Mismo núcleo
    2. Mismo multinúcleo
    3. Mismo nodo
  - Equilibrado teniendo en cuenta jerarquía
    - Equilibrar nodos
    - Equilibrar multinúcleos de cada nodo
    - Equilibrar núcleos de cada multinúcleo
    - Equilibrar procesadores lógicos de cada núcleo

Planificación con una cola por procesador

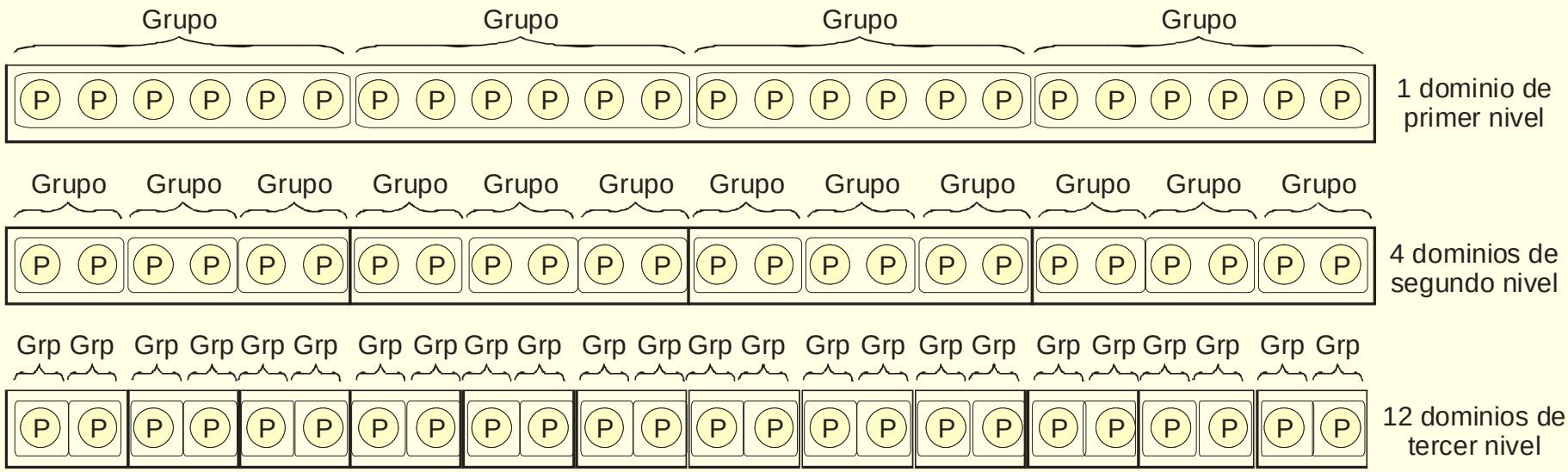


Planificación multiprocesador en Linux

- Mejoras en versión 2.6: uso de una cola por UCP
- Gestión de carácter jerárquico: **dominios de planificación**
  - Dominio=conjunto de grupos
  - Dominio intenta mantener carga equilibrada en sus grupos
    - Cada grupo tiene un "poder de cómputo" (*CPU Power*)
    - Tiene en cuenta grado de independencia de UCP
      - 2 UCP lógica $\in$ mismo núcleo $\rightarrow$ *CPU Power del grupo* = 1,1
- Equilibrado de carga basado en info. específica de cada nivel
  - Frecuencia de comprobación de desequilibrios
    - Mayor en niveles más bajos
  - Nivel de desequilibrio que provoca migración
    - Menor en niveles más bajos
  - Coste de la migración (mayor cuanto más afinidad se pierda)
    - Entre UCPs lógicas del mismo núcleo $\rightarrow$ Coste 0

Dominios de planificación

NUMA con 4 nodos de 3 UCPs físicas (2 lógicas/física)



Planificación de aplicaciones paralelas en MP

- Aspecto vinculado con el campo del paralelismo
  - Básicamente ajeno al SO
  - Implementado normalmente por entorno de ejecución paralelo
  - Presentación sólo da una visión general
- Aplicación paralela AP formada por múltiples procesos/*threads*
  - No finaliza hasta que no termine el último
- Alto grado de paralelismo e interacción
  - Procesos/*threads* de una AP deberían ejecutar simultáneamente
- En el sistema hay un conjunto de APs con ejecución *batch*
  - Uso de un planificador a largo plazo
- 2 técnicas más frecuentes
  - *Space sharing*
  - *Gang scheduling*
- Temas abiertos: asignación de UCPs a AP en MP jerárquico

Space sharing

- Multiplexación en el espacio de las APs
- Se asigna un conjunto de procesadores a cada AP
  - Una UCP dedicada a cada proceso/*thread*
    - Uso de primitivas de afinidad estricta del SO para la reserva
  - Ejecución simultánea: interacción muy rápida
  - Sin sobrecarga por cambios de contexto
- Planificador a largo plazo controla la entrada de trabajos
  - AP declara cuántas UCPs requiere y espera entrada al sistema
  - Planificador asigna UCPs a APs siguiendo una política dada
    - FCFS, SJF (requiere estimación de tiempo), prioridad, EDF,…
  - *backfilling*: UCPs disponibles no satisfacen a AP 1º en cola
    - Se "cuela" otra AP pero garantizando no inanición de la 1ª
- Extensión para servidor paralelo: asignación dinámica
  - Servidor puede ajustar grado de paralelismo a UCPs disponibles

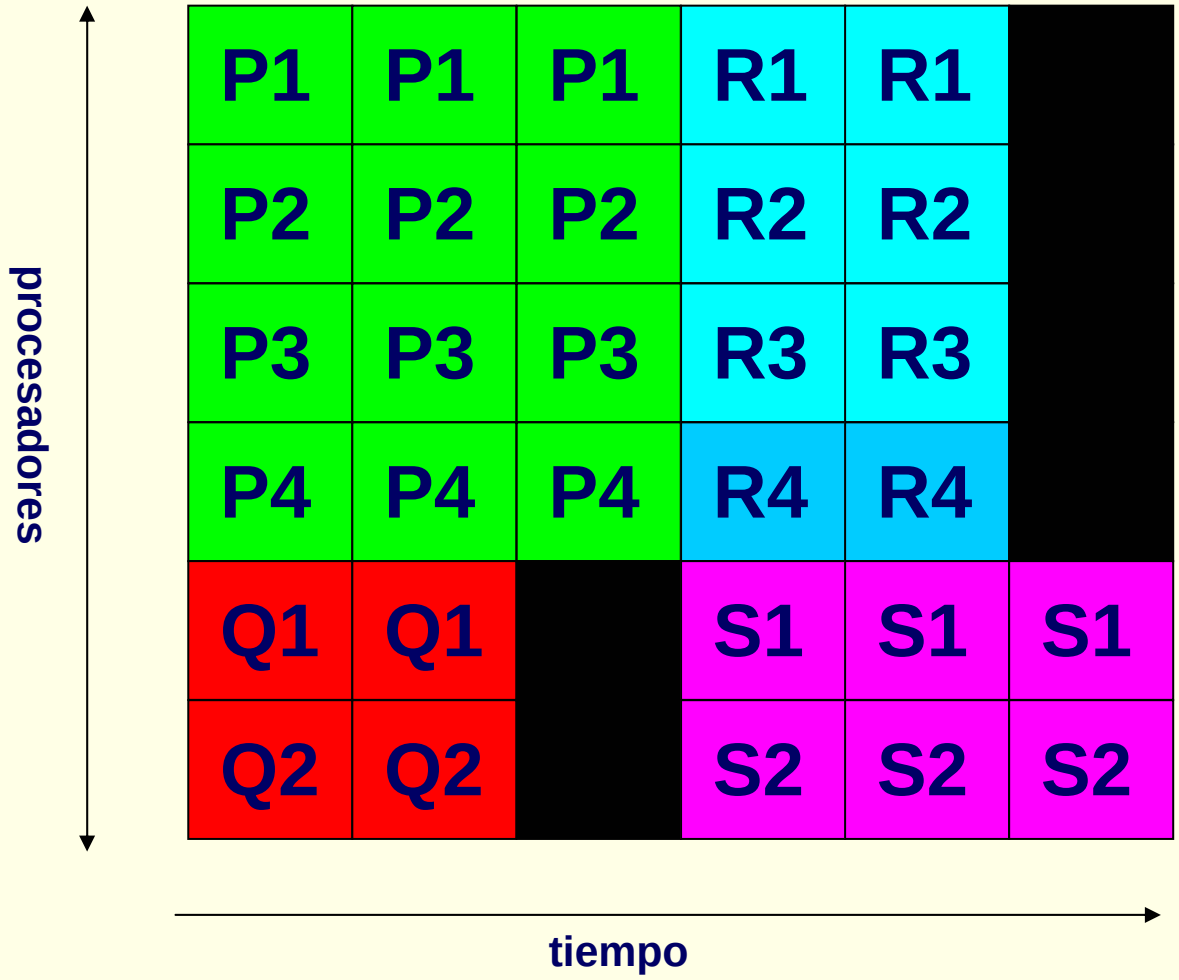
Gang scheduling

- Multiplexación en el espacio y en el tiempo de las APs
  - Procesos/*threads* de una AP ejecutan simultáneamente
  - Pero no tienen procesadores dedicados
  - En cada rodaja se ejecutan todos los proc/*threads* de varias APs
    - Dificultad/ineficiencia implementar una planificación sincronizada
  - Sobrecarga por cambios de contexto involuntarios
  - Reduce tiempo de espera de las APs para entrar al sistema
    - Aunque alarga su tiempo de ejecución
- Planificador controla la asignación de UCPs a APs
  - AP declara cuántas UCPs requiere
  - Planificador se basa en una matriz de Ousterhout
    - Rodajas X Procesadores
    - O[i,j]: qué proceso/*thread* ejecuta en UCP i durante rodaja j

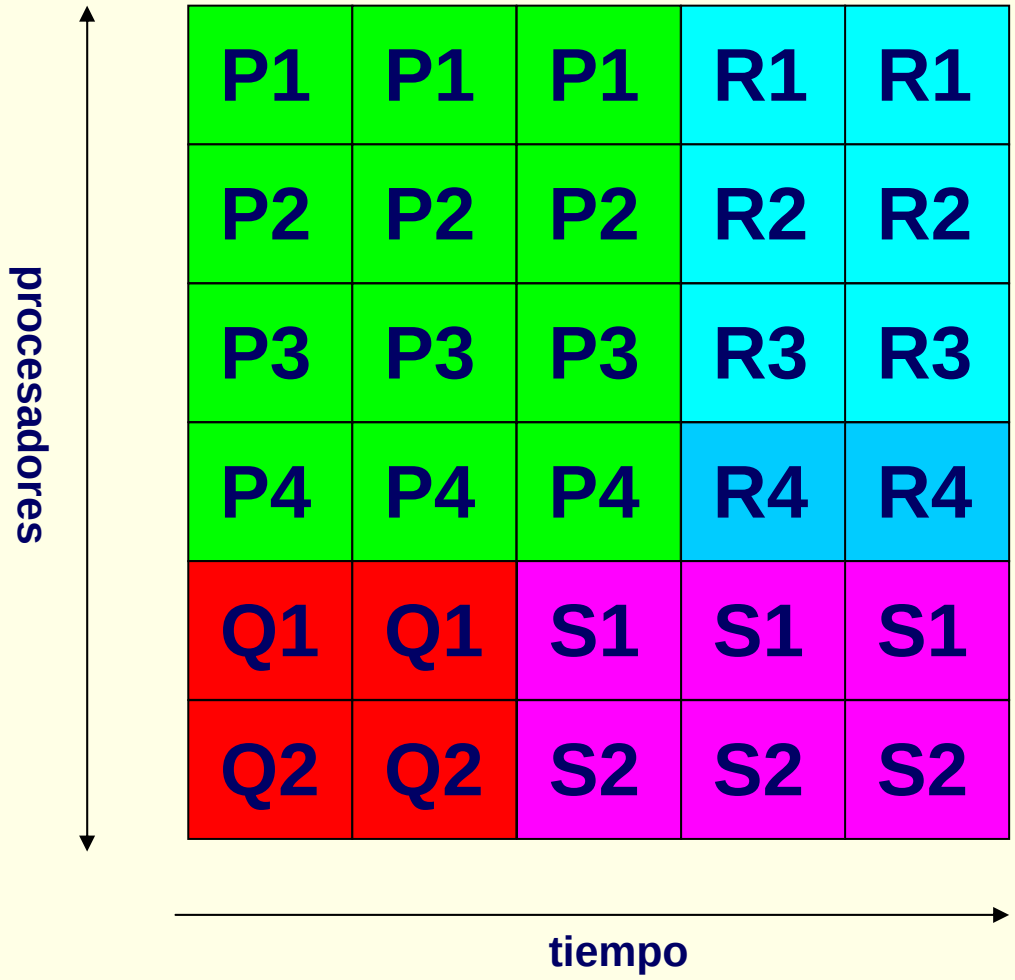
Ejemplo de planificación de APs en MP

- Sistema MP con 6 procesadores
- 4 APs por orden de prioridad (o llegada)
  - P requiere 4 UCPs y dura 3 unidades
  - Q requiere 2 UCPs y dura 2 unidades
  - R requiere 4 UCPs y dura 2 unidades
  - S requiere 2 UCPs y dura 3 unidades
- 3 estrategias:
  - *Space sharing* sin *backfilling*
  - *Space sharing* con *backfilling*
  - *Gang scheduling*

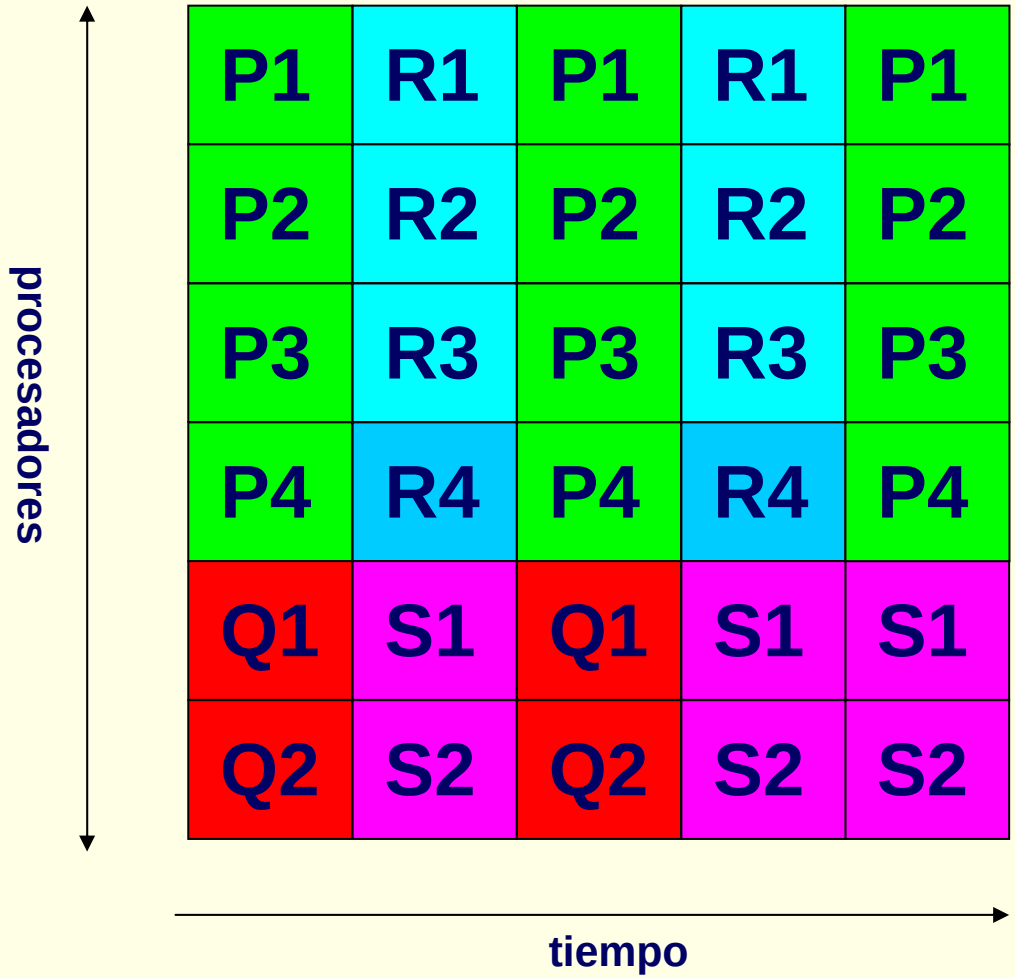
Ejemplo: *Space sharing* sin *backfilling*



Ejemplo: *Space sharing* con *backfilling*



Ejemplo: *Gang Scheduling*



Planificación en sistemas distribuidos

- Aspecto vinculado con el campo de los sistemas distribuidos
  - Básicamente ajeno al SO
  - Implementado normalmente por *middleware*
  - Presentación sólo da una visión general
- Falta de memoria compartida condiciona planificación en SS.DD.
  - Equilibrio de carga → migración (como con MP y cola/UCP)
    - Pero sin memoria compartida: costosa y técnicamente compleja
      - Incluso no factible en algunos sistemas
    - Necesario migrar mapa memoria y recursos asociados al proceso
- Asignación del procesador inicial a un proceso
  - Ejecución en la UCP donde se crea
  - Ejecución remota para reparto de carga
    - Más sencilla que migración
    - Problemas en sistemas heterogéneos

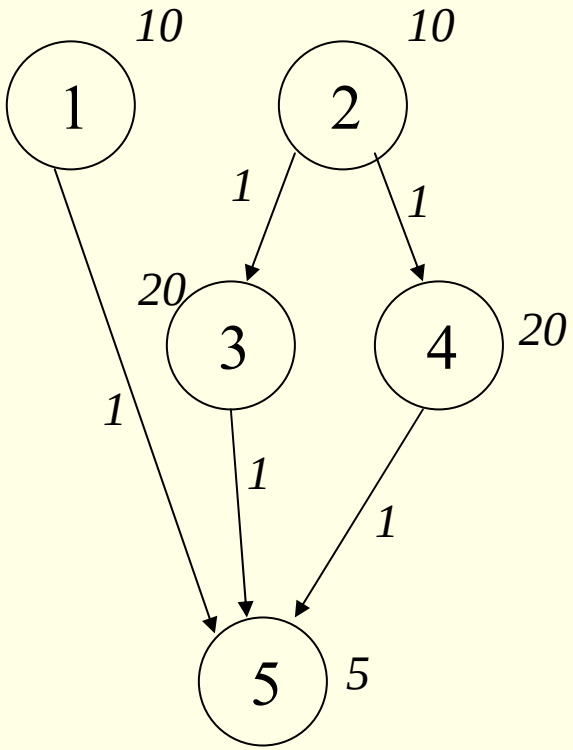
Estrategias de equilibrado de carga

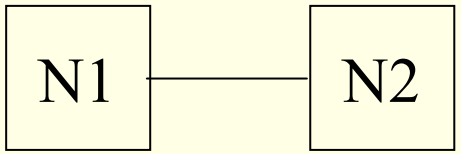
- Iniciada por el emisor: emisor busca receptores
  - Nodo sobrecargado pide ayuda
    - P.e. a un conjunto aleatorio de nodos
  - Envía un proceso al nodo seleccionado (p.e. menos cargado)
    - Mejor nuevo: Ejecución remota, no requiere migración
  - Sobrecarga peticiones ayuda inútiles si sistema muy cargado
- Iniciada por el receptor: receptor solicita procesos
  - Nodo descargado ofrece ayuda
    - P.e. a un conjunto aleatorio de nodos
  - Pide un proceso al nodo seleccionado (p.e. más cargado)
    - Requiere migración
  - Sobrecarga ofertas inútiles si sistema poco cargado (- grave)
- Simétrico: combinación de las anteriores

Planificación de aplicaciones paralelas en SS.DD.

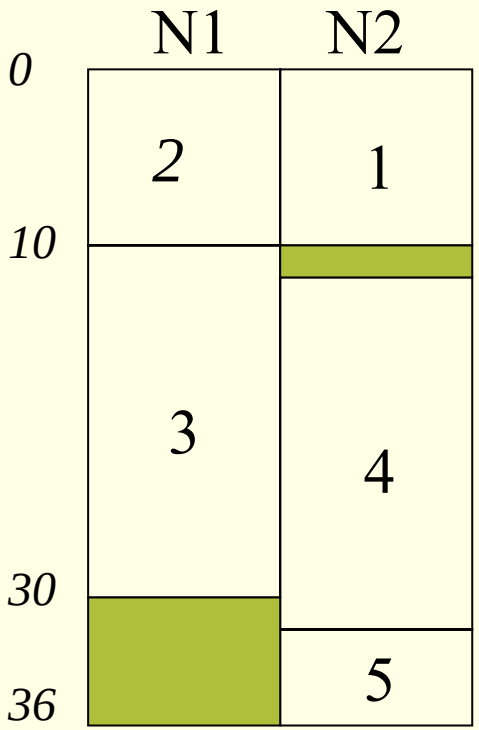
- Maximizar paralelismo: Esquema con 1 proceso de la AP/nodo
  - Gestión similar a *Space sharing* en MP
    - Planificador a largo plazo controla la entrada de trabajos
    - AP declara cuántas UCPs requiere y espera entrada al sistema
    - Planificador asigna UCPs a APs siguiendo una política dada
      - FCFS, SJF (requiere estimación de tiempo), prioridad, EDF,…
    - *backfilling*: UCPs disponibles no satisfacen a AP 1º en cola
      - Se "cuela" otra AP pero garantizando no inanición de la 1ª
- Esquema con múltiples procesos de la AP/nodo
  - Asignación estática de conjunto de procesos a nodos
    - Maximizando paralelismo y minimizando comunicación
  - Depende de la arquitectura de la AP. Ejemplos:
    - Modelo basado en precedencias: grafo acíclico dirigido (DAG)
    - Modelo basado en comunicaciones: grafo no dirigido

Ejemplo modelo precedencia de tareas (DAG)

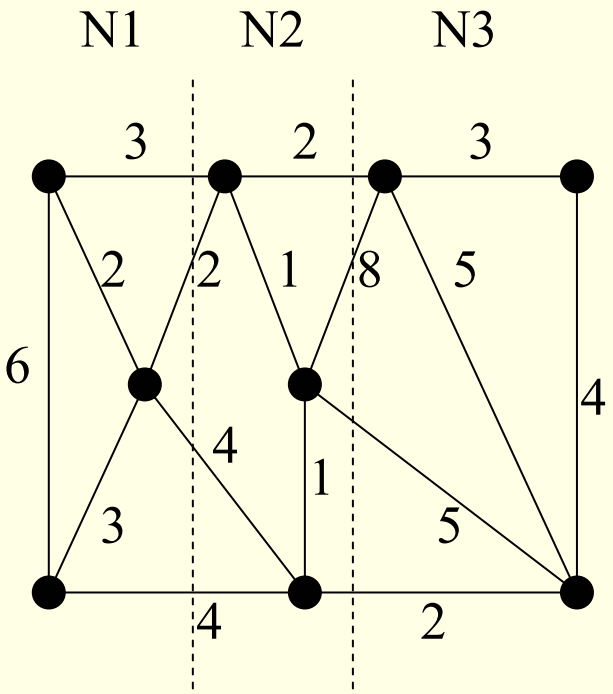




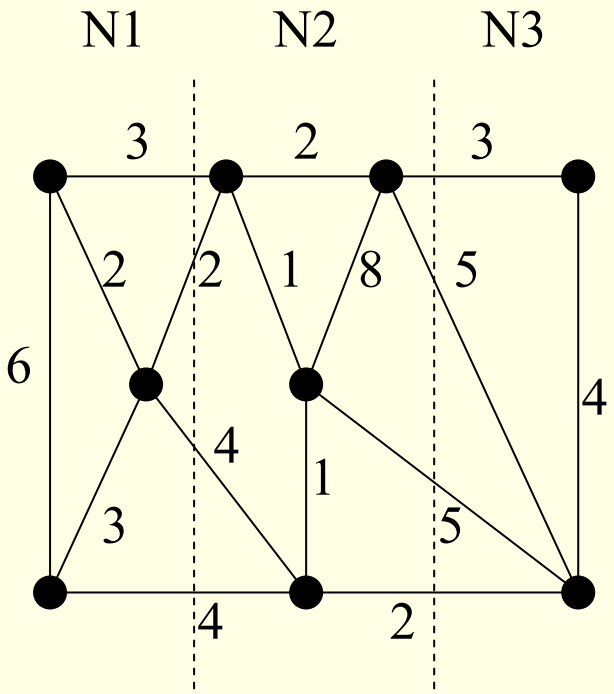
Planificador



Ejemplo modelo basado en comunicaciones



Tráfico entre nodos:
13+17=30



Tráfico entre nodos:
13+15=28

Tanenbaum. "Sistemas Operativos Distribuidos"