

# Bloqueos Mutuos

Thursday, 12 March 2020 5:50 PM

## ¿Cuándo se presenta un bloqueo mutuo?

### Condiciones de Coffman

**Exclusión mutua** Los procesos reclaman acceso exclusivo de los recursos

**Espera por** Los procesos mantienen los recursos que ya les habían sido asignados mientras esperan recursos adicionales

**No apropiatividad** Los recursos no pueden ser extraídos de los procesos que los tienen hasta su completa utilización

**Espera circular** Existe una cadena circular de procesos en que cada uno mantiene a uno o más recursos que son requeridos por el siguiente en la cadena



Gunnar Wolf

Administración de procesos: Bloqueos mutuos y politi

El SO y los Bloqueos mutuos  
Prevención  
Evasión  
Detección y recuperación  
La triste realidad...

## Evaluando en base a las condiciones de Coffman

- Cada una de las condiciones presentadas son *necesarias, pero no suficientes* para que haya un bloqueo
- Pero pueden alertarnos hacia una situación de riesgo
- Cuando se presentan las cuatro, tenemos un bloqueo mutuo que sólo puede resolverse terminando a uno de los procesos involucrados
  - Pérdida de datos / estado

## Esquematizando el ejemplo clásico

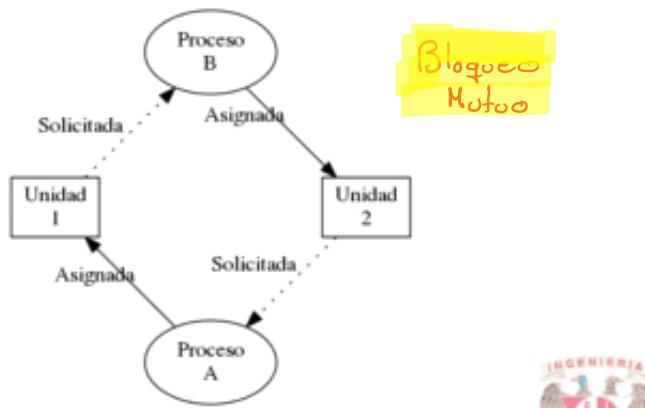


Figura: Esquema clásico de un bloqueo mutuo simple: Los procesos A y B esperan mutuamente para el acceso a las unidades 1 y 2.



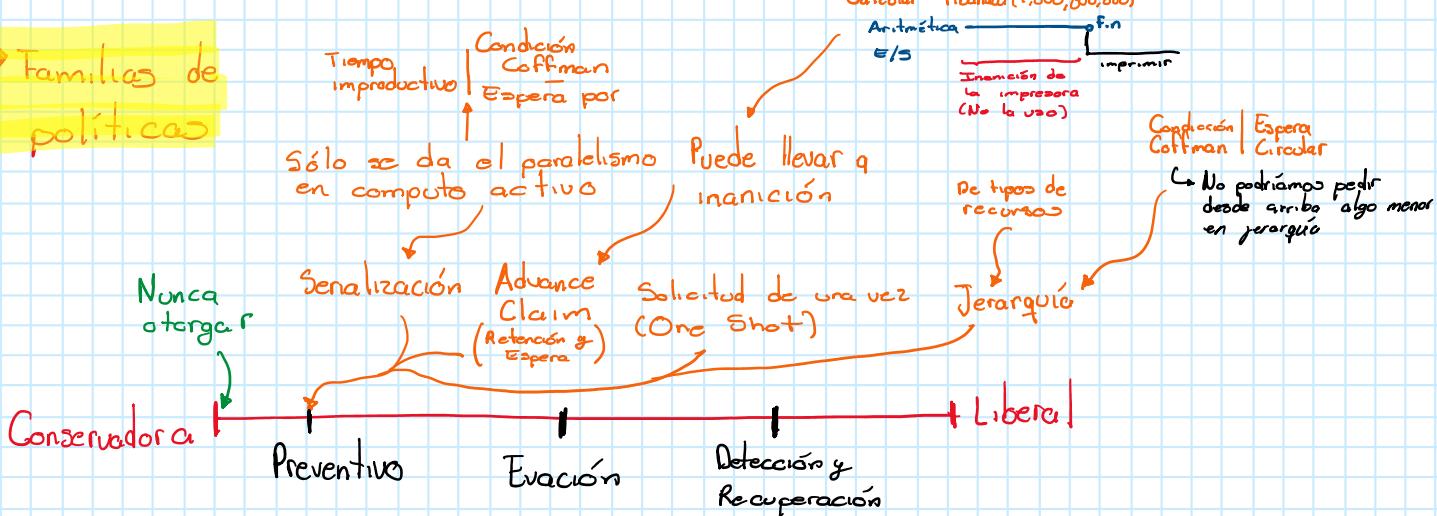
Gunnar Wolf Administración de procesos: Bloqueos mutuos y polí

El SO y los Bloqueos mutuos  
Prevención Evasión  
Detección y recuperación  
La triste realidad...

## El punto de vista del sistema operativo

- El rol del sistema operativo va más allá de lo presentado en las láminas anteriores (*Exclusión mutua*)
- No podemos asumir que los procesos cooperarán entre sí
  - Ni siquiera que sabrán por anticipado de la existencia mutua
- Un rol primario del sistema operativo es gestionar los recursos del equipo

### Familias de políticas



## Categorías de estrategias ante bloqueos mutuos

**Prevención** Modela el comportamiento del sistema para *eliminar toda posibilidad* de un bloqueo.

Resulta en una utilización subóptima de recursos.

**Evasión** Impone condiciones menos estrictas. No puede evitar *todas las posibilidades* de un bloqueo; cuando éste se produce busca evitar sus consecuencias.

**Detección y recuperación** Permite que ocurran los bloqueos, pero busca *determinar si ha ocurrido* y actuar para eliminarlos.



### ► Prevención

Impone condiciones menos estrictas. No puede evitar *todas las posibilidades* de un bloqueo; cuando éste se produce busca *evitar sus consecuencias*.



Gunnar Wolf

Administración de procesos: Bloqueos mutuos y polit

El SO y los Bloqueos mutuos  
Prevención  
Evasión  
Detección y recuperación  
La triste realidad...

## Enfoque 1: Flujos seguros e inseguros

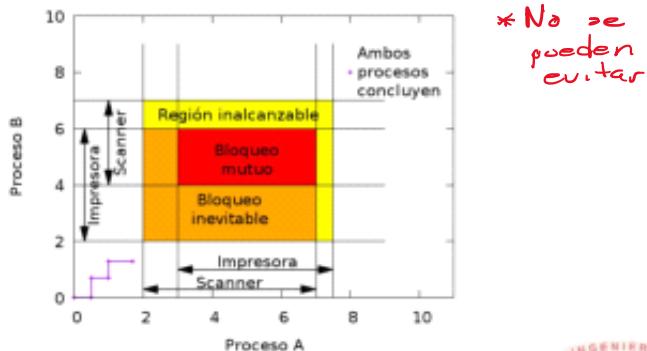
- Realizado por el planificador
- Requiere saber *por anticipado* qué procesos utilizarán qué recursos
  - Un poco menos detallado que la prevención (número de recursos por categoría)
- Saber *cuándo* se va a usar cada recurso
- Análisis de la interacción, marcando *áreas de riesgo*



### Flujos seguros e inseguros



\* No se oceden



**Figura:** Evasión de bloqueos: Los procesos A (horizontal) y B (vertical) requieren del acceso exclusivo a dos recursos rivales, exponiéndose a bloqueo mutuo.



Gunnar Wolf      Administración de procesos: Bloqueos mutuos y polí

El SO y los Bloqueos mutuos  
Prevención  
Evasión  
Detección y recuperación  
La triste realidad...

## Áreas de riesgo

- Mientras avancemos por el *área segura* no hay riesgo de bloqueos
- Sistemas uniprocesador, sólo avance vertical/horizontal
  - Sistemas multiprocesador, avance diagonal
- Áreas de riesgo: Cuando *alguno de los recursos rivales* es otorgado
- El bloqueo mutuo se produce cuando  $3 \leq t_A \leq 7$ , y  $4 \leq t_B \leq 6$
- Pero, *sabiendo esto*, el SO debe mantener a  $t_B < 2$  siempre que  $2 \leq t_A \leq 3$ 
  - O a  $t_A < 2$  siempre que  $2 \leq t_B \leq 4$ .



## Enfoque 2: Algoritmo del banquero

- **Edsger Dijkstra**, para el sistema operativo **THE**, 1965-1968
- El sistema procede cuidando de la *líquidez* para siempre poder satisfacer los *préstamos* (recursos) de sus clientes
- Permite que el *conjunto de recursos* solicitados por los procesos sean mayores a los disponibles



Gunnar Wolf      Administración de procesos: Bloqueos mutuos y polí

El SO y los Bloqueos mutuos  
Prevención  
Evasión  
Detección y recuperación  
La triste realidad...

## Requisitos para el Algoritmo del banquero

## Requisitos para el Algoritmo del banquero

- Debe ejecutarse cada vez que un proceso solicita recursos
- Todo proceso debe declarar su **reclamo máximo (claim)** de **recursos al iniciar su ejecución**
  - Si el reclamo en cualquier categoría es superior al máximo existente, el sistema niega la ejecución



Sistema (Recurso)  
Total = 5  
Libreos = 3

Programa reclama  
(Pide) = 4  
asignado = 0  
solicitado = 0  
*No se ocupará directamente o al mismo tiempo*

*//Sólo será aceptado si pide menos del total*

## Estados

**Estado** Matrices de recursos disponibles, reclamos máximos y asignación de recursos a los procesos en un momento dado

**Estado seguro** Un estado en el cual todos los procesos pueden ejecutar hasta el final sin encontrar un bloqueo mutuo.

**Estado inseguro** Todo estado que no garantice que todos los procesos puedan ejecutar hasta el final sin encontrar un bloqueo mutuo.



## Lógica del algoritmo del banquero

Cada vez que un proceso solicita recursos, se calcula cuál sería el estado resultante de otorgar dicha solicitud, y se otorga siempre que:

estado resultante de otorgar dicha solicitud, y se otorga siempre que:

- No haya reclamo por más recursos que los disponibles
- Ningún proceso solicite (o tenga asignados) recursos por encima de su reclamo
- La suma de los recursos *asignados* por cada categoría no sea mayor a la cantidad de recursos *disponibles* en el sistema para dicha categoría



## Definiendo estados seguros

Un estado es *seguro* cuando hay una secuencia de procesos (denominada *secuencia segura*) tal que:

- ① Un proceso  $j$  puede necesariamente terminar su ejecución
  - Incluso si solicitara todos los recursos que reservó en su reclamo
  - Siempre debe haber suficientes recursos libres para satisfacerlo
- ② Un segundo proceso  $k$  de la secuencia puede terminar:
  - Si  $j$  termina y libera todos los recursos que tiene
  - Siempre que sumado a los recursos disponibles ahora, con aquellos que liberaría  $j$ , hay suficientes recursos libres
- ③ El  $i$ -ésimo proceso puede terminar si todos los procesos anteriores terminan y liberan sus recursos.



Gunnar Wolf

Administración de procesos: Bloqueos mutuos y politi

El SO y los Bloqueos mutuos  
Prevención  
Evasión  
Detección y recuperación  
La triste realidad...

## Peor caso con el algoritmo del banquero

En el peor de los casos, esta secuencia segura nos llevaría a bloquear todas las solicitudes excepto las del proceso único en el orden presentado.



► Ejemplos

## Ejemplo

Asumiendo que tenemos *sólo una clase de recursos* y nos quedan dos instancias libres:

Proceso	Asignado	Reclamando
A	4	6
B	4	11
C	2	7

- ① A puede terminar porque sólo requiere dos instancias adicionales
- ② Terminado A, C puede recibir las 5 restantes que requiere
- ③ Terminados A y C, B puede satisfacer las 7
- ④ La secuencia A-C-B es segura.



Gunnar Wolf

Administración de procesos: Bloqueos mutuos y politi

El SO y los Bloqueos mutuos  
Prevención  
Evasión  
Detección y recuperación  
La triste realidad...

## Ejemplo 2

Sólo una clase de recursos, 2 instancias libres

Proceso	Asignado	Reclamado
A	4	6
B	4	11
C	2	9

- ① A puede satisfacer su demanda
- ② Pero una vez terminado A, no podemos asegurar las necesidades ni de B ni de C
- ③ Este es un *estado inseguro*, por lo cual el algoritmo del banquero *no debe permitir llegar a él*.



## Ejemplo 3: Paso por paso

¿Hasta dónde nos dejaría avanzar el algoritmo del banquero?

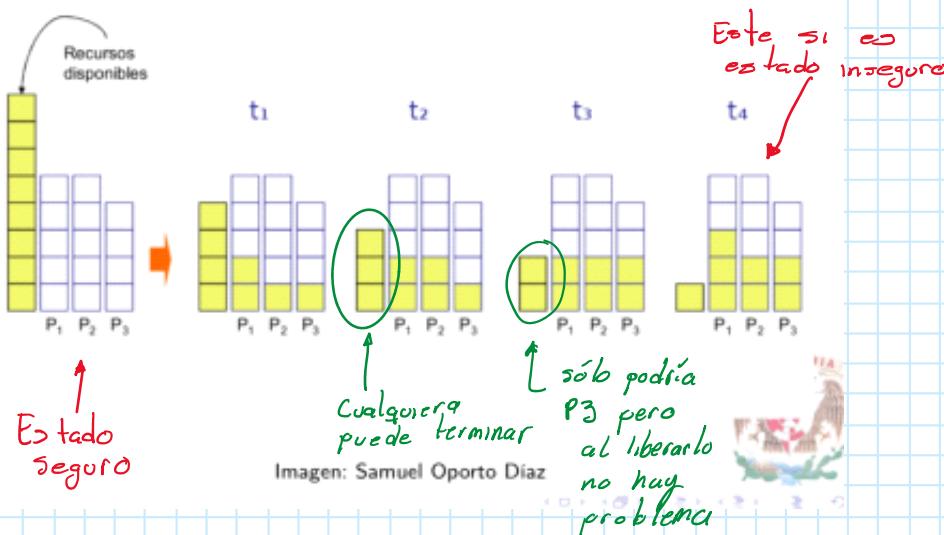
- Estado inicial:
  - 8 recursos disponibles
  - Reclamos:  $P_1: 5, P_2: 5, P_3: 4$
- $t_1: P_1$  con 2,  $P_2$  con 1,  $P_3$  con 1
- $t_2: P_2$  solicita 1
- $t_3: P_3$  solicita 1
- $t_4: P_1$  solicita 1



Gunnar Wolf      Administración de procesos: Bloqueos mutuos y políticas de asignación

El SO y los Bloqueos mutuos  
Prevención  
Evasión  
Detección y recuperación  
La triste realidad...

### Ejemplo 3: Paso por paso



## Implementación ejemplo (para una sola categoría)

```
l = ['A', 'B', 'C', 'D', 'E']; # Todos los procesos del sistema
s = []; # Secuencia segura
while ! l.empty? do
  p = l.select{|id| reclamado[id] - asignado[id] <=
    libres}.first
  raise Exception, 'Estado inseguro' if p.nil?
  libres += asignado[p]
  l.delete(p)
  s.push(p)
end
puts "La secuencia segura encontrada es: " + s
```



Gunnar Wolf      Administración de procesos: Bloqueos mutuos y políticas de asignación

El SO y los Bloqueos mutuos

## Precisiones

- En el ejemplo 2, es posible que ni  $B$  ni  $C$  requirieran ya todos sus recursos reclamados
- Pero el estado es inseguro.
- El algoritmo del banquero, en el peor caso, puede tomar  $O(n!)$ 
  - Típicamente toma  $O(n^2)$
- Hay refinamientos a este algoritmo que reducen su costo de ejecución
  - Puede ser llamado con muy alta frecuencia



# Ya es un algoritmo vivo y sobre todo conservador

► Parte Liberal

← Se presenta el bloqueo Mutuo pero se resuelve

## Característica básica

Permite que ocurran los bloqueos, pero busca **determinar si ha ocurrido y actuar para eliminarlos.**

A diferencia de la **prevención** y la **evasión**, las cuatro condiciones de **Coffman** pueden presentarse.

The screenshot shows a presentation slide with the following elements:

- Title:** Funcionamiento básico
- Navigation:** Back, Forward, Home, etc.
- Author:** Gunnar Wolf
- Subject:** Administración de procesos: Bloqueos mutuos y políticas
- Content:** El SO y los Bloqueos mutuos  
Prevención  
Evasión  
Detección y recuperación  
La triste realidad...

- **Se ejecuta como una tarea periódica**
  - Busca *limitar* el impacto de los bloqueos existentes en el sistema
  - (Discutiremos al respecto más adelante...)
- **Mantenemos una lista de recursos existentes, asignados y solicitados**
- Eliminando lo *obviamente resuelto*, nos quedamos con lo *obviamente bloqueado*



## Estrategia general

- Tenemos la representación completa de los recursos y procesos en el sistema
- Buscamos *reducir* la gráfica, retirando los elementos que no aportan *información relevante*:
  - Los procesos que no estén solicitando ni tienen asignado un recurso
  - Para los procesos restantes: Si todos los recursos que solicitan *pueden ser concedidos* (no están concedidos a otro), eliminamos al proceso y a todas sus flechas.
    - Repetimos cuantas veces sea posible
  - Si no quedan procesos en el grafo, no hay interbloqueos y podemos continuar
  - Los procesos "irreductibles" están en bloqueo mutuo.
    - Proceder según la política del sistema



Gunnar Wolf Administración de procesos: Bloqueos mutuos y políticas

El SO y los Bloqueos mutuos  
Prevención  
Evasión  
Detección y recuperación  
La triste realidad...

### Ejemplo 1

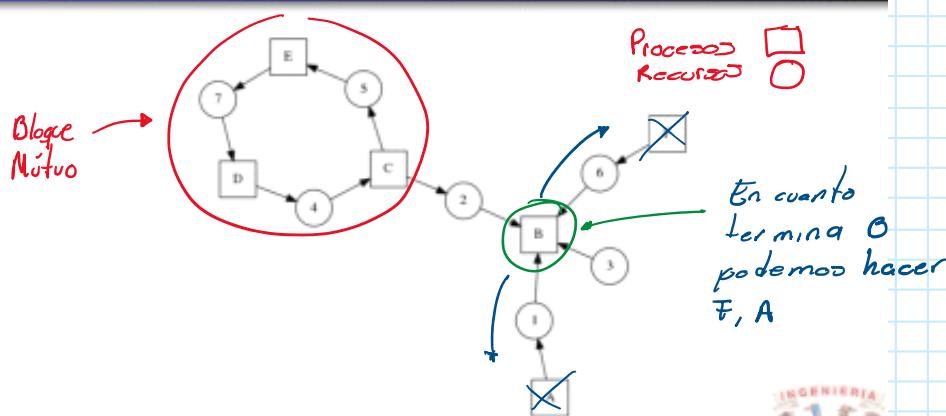


Figura: Detección de ciclos denotando bloques: Grafo de procesos y recursos en un momento dado

	A	B	C	D	E	F	Solicitado [S]	Asignado [A]
1	S	A						
2	A		S					
3	A			S				
4			A	S				
5	A			A				
6				A	S			
7								

↑  
No puedes solicitar 2

→ Buscar cual puede terminar

terminable = columnas.select (sólo A)

```

for proc in terminable:
    termina(proc)
    for recurso in proc.asig:

```

termina(proc)

for recurso in proc.assigned:

libera(recurso)

if terminable.empty and !columnas.empty:

//Primero B es terminable  
después [A y F]

Una vez esto:

Mátelos, luego 'virigüa'

- Terminar a todos los procesos bloqueados
- Técnica más sencilla y más justa...
  - Para cierta definición de justicia
- Maximiza la pérdida de información



Gunnar Wolf

Administración de procesos: Bloqueos mutuos y polít.

El SO y los Bloqueos mutuos  
Prevención  
Evasión  
Detección y recuperación  
La triste realidad...

Retroceder al último punto de control (checkpoint)

- Sólo cuando el sistema implementa esta funcionalidad
- Sólo cuando *ninguno* de los procesos depende de factores externos
- Retroceder en el tiempo, ¿no llevaría a un nuevo bloqueo?

Retroceder al último punto de control (checkpoint)

- Sólo cuando el sistema implementa esta funcionalidad
- Sólo cuando *ninguno* de los procesos depende de factores externos

## Retroceder al último punto de control (checkpoint)

- Sólo cuando el sistema implementa esta funcionalidad
- Sólo cuando *ninguno* de los procesos depende de factores externos
- Retroceder en el tiempo, ¿no llevaría a un nuevo bloqueo?
- Los bloqueos están ligados al orden *específico* de ejecución
- Probablemente otro orden específico se salve del bloqueo
  - Y si no... Se vuelve una vez más.
  - Tal vez, agregar un contador de *intentos* que cambie el planificador

\*Modelo de ANDROID



Gunnar Wolf      Administración de procesos: Bloqueos mutuos y políticas

El SO y los Bloqueos mutuos  
Previsión  
Evasión  
Detección y recuperación  
La triste realidad...

Terminación selectiva / limitada

- Terminar forzosamente, uno a uno (y no en bloque), a los procesos bloqueados
  - Terminar a uno y re-evaluar el estado
  - Los restantes pueden continuar operando
- **¿Cómo elijo al desafortunado? Varias estrategias...**

## Estrategias para elegir el proceso a terminar (1)

- Los procesos más sensibles para detener/relanzar: Los que demandan *garantías de tiempo real*. Evitar penalizarlos.
- Causar una menor pérdida de trabajo. El que haya consumido menor cantidad de procesador hasta el momento.
- Mayor tiempo restante estimado (si puedo estimar cuánto procesamiento *queda pendiente*)



Gunnar Wolf      Administración de procesos: Bloqueos mutuos y políticas

El SO y los Bloqueos mutuos  
Previsión  
Evasión  
Detección y recuperación

## Estrategias para elegir el proceso a terminar (2)

- Menor *número* de recursos asignados hasta ahora (como criterio de *justicia*: ¿qué proceso está haciendo un uso *más juicioso* del sistema?)
- Prioridad más baja (cuando la hay)
- Procesos con naturaleza repetible sin pérdida de información (aunque sí de tiempo)
  - p.ej. es mejor interrumpir una compilación que la actualización de una BD



► Triste Realidad

## El Algoritmo del avestruz

- Mecanismo más frecuente utilizado por los sistemas operativos de propósito general
- Ignorar las situaciones de bloqueo
- Esconder la cabeza bajo tierra y esperar a que pasen
- Esperar que su ocurrencia sea suficientemente poco frecuente
- Esperar que el usuario detecte y resuelva el problema por sí sólo



Gunnar Wolf

Administración de procesos: Bloqueos mutuos y polit

El SO y los Bloqueos mutuos  
Prevención  
Evasión  
Detección y recuperación  
La triste realidad...

## El por qué del avestruz

- Las condiciones impuestas por las diversas estrategias son casi imposibles de alcanzar
- Conocimiento previo de requisitos insuficiente
- Bloqueos originados fuera de nuestra esfera de acción
  - Recursos externos
  - Procesos *representante* (*proxy*)
  - Estructuras a niveles superiores
- Modelo de computación tendiente al interactivo
  - En servidores: Programas (¡y personas!) dedicados al monitoreo



## ¿Qué es un recurso?

- Para que estos mecanismos funcionen, tenemos que definir qué es un recurso — A qué esperamos cubrir (y, por tanto, a qué no)
  - No sólo cintas, impresoras, tarjetas de audio...
  - ¡También segmentos de memoria, tiempo de procesamiento!
  - ¿Y las estructuras lógicas creadas por el SO? Archivos, semáforos, monitores, ...
  - ¿Y elementos que vienen de fuera de nuestro sistema? (cómputo distribuido, SOAP/servicios Web, etc.)

cualquier cosa



Gunnar Wolf

Administración de procesos: Bloqueos mutuos y polit

El SO y los Bloqueos mutuos  
Prevención  
Evasión  
Detección y recuperación  
La triste realidad...

## Cuestión de compromisos

## Cuestión de compromisos

- Un sistema operativo de propósito general tiene que funcionar para muy distintas situaciones
  - Es muy difícil plantear esquemas aptos para toda necesidad
- Debe tomarse una decisión entre lo correcto y lo conveniente
  - Un SO no debería permitir que hubiera bloqueos
  - Pero la inconveniencia a usuarios y programadores sería demasiada



### ► Puntos de vista

## Posiciones disciplinares: Matemáticos contra ingenieros

- ¿Dónde cabe mejor la computación? ¿Y dónde cabemos mejor cada uno de nosotros?
- La asignación de recursos puede verse como un problema formal, matemático
  - Los mecanismos descritos no son perfectos
  - Pero el problema *no ha demostrado ser intratatable*
  - Un bloqueo es claramente un error
- Los matemáticos en nuestro árbol genealógico académico nos llaman a no ignorar el problema
  - A resolverlo sin importar la complejidad computacional



Gunnar Wolf

Administración de procesos: Bloqueos mutuos y politi

El SO y los Bloqueos mutuos  
Previsión  
Evasión  
Detención y recuperación  
La triste realidad...

## El punto de vista del ingeniero

- La ingeniería es la ciencia *aplicada al mundo real*
- Debe, sí, evitar efectos nocivos
  - Pero también, calcula costos, probabilidades de impacto, umbrales de tolerancia...
- Un sistema típico corre riesgo de caer en un bloqueo con una probabilidad  $p > 0$ 
  - Impacto: Perder *dos procesos* en un sistema
  - Otros factores de fallo: Hardware, otros subsistemas del SO, errores en cada uno de los programas...
- Prevenir el bloqueo conlleva complejidad en desarrollo o disminución en rendimiento



# Planificador Procesos

Tuesday, 17 March 2020 5:43 PM

5:46 PM Tue 17 Mar Principal decisión en un sistema multitareas

- ¿Qué proceso es el siguiente a ejecutar?
  - ¿Qué procesos han ido terminando?
- ¿Qué eventos ocurrieron que hacen que *cambien de estado*?
  - Solicitudes (y respuestas) de E/S
  - Swap de/a disco
- ¿Cuál es el siguiente proceso al que le toca atención del CPU?
  - ¿Y por cuánto tiempo?

Vemos que hay tres tipos muy distintos de planificación.



Gunnar Wolf Planificación de procesos: Algoritmos de planificación

Introducción  
Métricas  
Algoritmos de planificación  
Esquemas híbridos y prioridades externas

## Planificador a largo plazo

- Cual es el siguiente proceso a ser iniciado
- Principalmente orientado a la operación *en lotes*
  - Principalmente a los sistemas con *spool*
  - También presente en la multiprogramación temprana
- Decide en base a los requisitos *pre-declarados* de los procesos, y a los recursos disponibles al ejecutarse
- Periodicidad: segundos a horas
- Hoy en día no se emplean
  - El usuario indica expresamente qué procesos iniciar
  - Podría verse a los programas como cron, at, o en Windows al Planificador de procesos como cubriendo este rol
    - Aunque son procesos *plenamente en espacio de usuario*

## Planificador a largo plazo

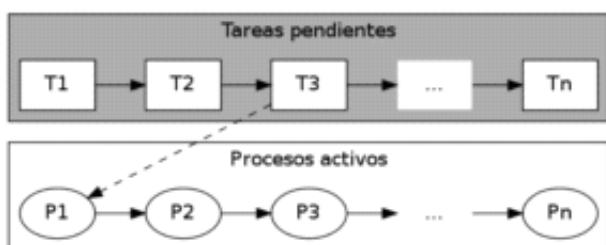


Figure: Planificador a largo plazo



## Planificador a mediano plazo

- **Cuáles procesos hay que *bloquear***

- Por escasez/saturación de algún recurso (p.ej. almacenamiento primario)
- Por haber iniciado una operación que no puede satisfacerse aún

- **Cuáles procesos hay que *desbloquear***

- A la espera de algún dispositivo
- Fueron enviados a *swap*, pero ya requieren o merecen ejecutarse

- Frecuentemente llamado *agendador (scheduler)*



## Planificador a mediano plazo

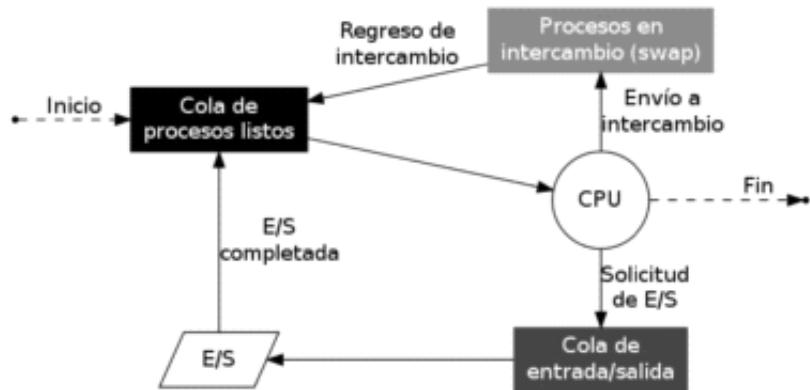


Figure: Planificador a mediano plazo, o agendador



Gunnar Wolf

Planificación de procesos: Algoritmos de planificación

Introducción

Métricas

Algoritmos de planificación  
Esquemas híbridos y prioridades externas

## Planificador a corto plazo

- Cómo compartir *momento a momento* al CPU entre todos los procesos
- Se efectúa decenas de veces por segundo
  - Debe ser simple, eficiente y rápido
- Se encarga de planificar los procesos *listos para ejecución*
  - Estados *listo* y *ejecutando*
- Frecuentemente llamado *despachador* (*dispatcher*)



## Planificador a mediano plazo

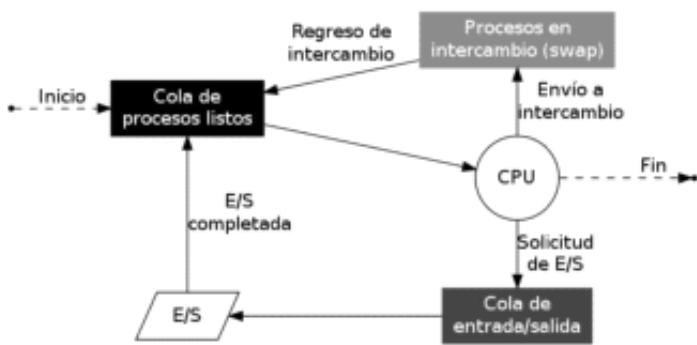


Figure: Planificador a mediano plazo, o agendador



Figure: Planificador a mediano plazo, o agendador

The slide has a header with the title 'Planificador a mediano plazo, o agendador'. Below the title is a navigation bar with icons for back, forward, search, and other presentation controls. The main content area contains a list of bullet points describing the short-term scheduler.

- Cómo compartir *momento a momento* al CPU entre todos los procesos
- Se efectúa decenas de veces por segundo
  - Debe ser simple, eficiente y rápido
- Se encarga de planificar los procesos *listos para ejecución*
  - Estados *listo* y *ejecutando*
- Frecuentemente llamado *despachador* (*dispatcher*)

## Planificador a corto plazo

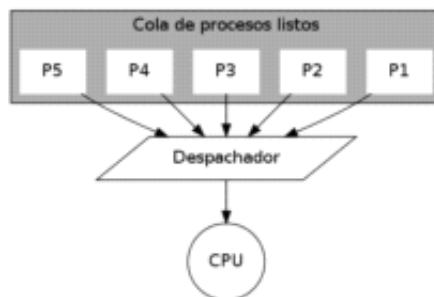
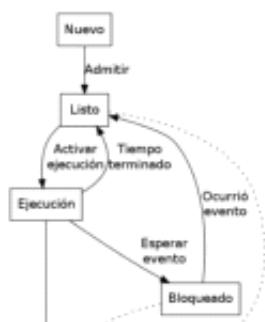


Figure: Planificador a corto plazo, o despachador

The slide has a header with the title 'Tipo de planificador según transición'. Below the title is a navigation bar with icons for back, forward, search, and other presentation controls. The main content area contains a state transition diagram and a list of bullet points defining long-term and medium-term planners based on transitions.



- **Largo plazo:**  
Admitir
- **Mediano plazo:**  
Ocurrió evento,  
Esperar evento



Ocurrió evento,  
Esperar evento

- **Corto plazo:**  
Activar ejecución FRIA  
Tiempo terminado



Figure: Diagrama de transición entre los estados de un proceso

## Tipos de proceso

- Diversos procesos tienen distintas características
- Alternan entre *ráfagas* (*bursts*)

**CPU** Trabajando con datos ya existentes en el sistema

**E/S** Mayormente esperando a eventos de E/S

- Un *programa* dado puede ser *mayormente* de un tipo u otro — Dicho programa está *limitado por CPU* o *limitado por E/S*
- Cuando termina una ráfaga de CPU y se suspende esperando E/S, deja de estar *listo* y sale de la vista del *despachador*
- Esto nos lleva a separar los procesos en...



Gunnar Wolf

Planificación de procesos: Algoritmos de planificación

Introducción

Métricas

Algoritmos de planificación  
Esquemas híbridos y prioridades externas

## Tipos de proceso

Ojo: ¡Un poco contraintuitivo!

**Largos** Han estado *listos* o *en ejecución* por mucho tiempo

- Esto es, están en una ráfaga de CPU

**Cortos** En este momento están en una ráfaga de E/S

- Requieren atención meramente ocasional del procesador
- Tienden a estar bloqueados, esperando a eventos
  - Como buena parte de los procesos



# Unidades a manejar

Para hablar de planificación del procesador, *no* vamos a manejar tiempos *estándar* (s, ms, ns), sino que:

**Tick** Un tiempo mínimo dado durante el cual se puede realizar trabajo útil. Medida caprichosa y arbitraria.

- En Windows, un *tick* dura entre 10 y 15 ms.
- En Linux (2.6.8 en adelante), dura 1 ms.

**Quantum** Tiempo mínimo, expresado en *ticks*, que se permitirá a un proceso el uso del procesador.

- En Windows, 2 a 12 *ticks* (esto es, 20 a 180ms).
- En Linux, 10 a 200 *ticks* (10 a 200ms)



Gunnar Wolf

Planificación de procesos: Algoritmos de planificación

Introducción

Métricas

Algoritmos de planificación

Esquemas híbridos y prioridades externas

## ¿Qué es mejor?

- No hay un sólo criterio para definir qué es una *mejor* respuesta
- El patrón correcto varía según el propósito del sistema
- Un proceso interactivo *sufre* si el tiempo de respuesta incrementa, aunque pueda procesar por más tiempo corrido
  - En caso de sufrir demoras, debemos intentar que sean *consistentes*, aunque el *tiempo promedio* resulte deteriorado
  - Es mejor saber que el sistema *siempre* tardará 0.5s en responder a mis necesidades a que unas veces responda de inmediato y otras tarde 3s.

## ¿Qué métricas compararemos?

Para un proceso  $p$  que requiere ejecutarse por tiempo  $t$ ,

**Tiempo de respuesta ( $T$ )** Tiempo total que toma el trabajo.  
Incluye el tiempo que pasó inactivo (pero listo).

**Tiempo en espera ( $E$ )** De  $T$ , cuánto tiempo está esperando ejecutar. (*Tiempo perdido*)

- $E = T - t$ ; Idealmente, para  $p$ ,  $E_p \rightarrow 0$

**Proporción de penalización ( $P$ )** Fracción del tiempo de respuesta durante la cual  $p$  estuvo en espera.

$$\bullet P = \frac{T}{t}$$

**Proporción de respuesta ( $R$ )** Fracción del tiempo de respuesta durante la cual  $p$  pudo ejecutarse.

$$\bullet R = \frac{t}{T} ; R = \frac{1}{P}$$



durante la cual  $p$  pudo ejecutarse.

- $R = \frac{t}{T} ; R = \frac{1}{P}$



Además de los anteriores, para el sistema...

**Tiempo núcleo o kernel** Tiempo que pasa el sistema en espacio de núcleo

**Tiempo desocupado (idle)** Tiempo en que la cola de procesos listos está vacía y no puede realizarse ningún trabajo.

- El sistema operativo aprovecha este tiempo para realizar *tareas de mantenimiento*

**Utilización del CPU** Porcentaje del tiempo en que el CPU está realizando *trabajo útil*.

- Conceptualmente, entre 0 y 100%
- En realidad, en un rango entre 40 y el 90%.



## Frecuencias

Respecto al patrón de llegadas y salidas de procesos a la cola de procesos listos:

$\alpha$  Frecuencia de llegada promedio

$\beta$  Tiempo de servicio requerido promedio

$\rho$  Valor de saturación,  $\rho = \frac{\alpha}{\beta}$

Esto significa:

$\rho = 0$  Nunca llegan procesos nuevos; el sistema estará desocupado

$\rho = 1$  Los procesos salen al mismo ritmo al que entran

$\rho > 1$  Los procesos llegan más rápido de lo que puede ser atendidos. La cola de procesos listos tiende a crecer.  $R$  disminuye para todos.



## ► Algoritmos de planificación

### Primero llegado, primero servido (FCFS)

- *First Come, First Serve.*

- También referido como *FIFO (First In, First Out)*

- El esquema más simple de planificación

## Primero llegado, primero servido (FCFS)

- *First Come, First Serve.*
  - También referido como *FIFO (First In, First Out)*
- El esquema más simple de planificación
- Apto para multitarea cooperativa
- Cada proceso se ejecuta en orden de llegada
- *Hasta que suelta el control*



Gunnar Wolf

Planificación de procesos: Algoritmos de planificación

Introducción  
Métricas  
Algoritmos de planificación  
Esquemas híbridos y prioridades externas

## Primero llegado, primero servido (FCFS)

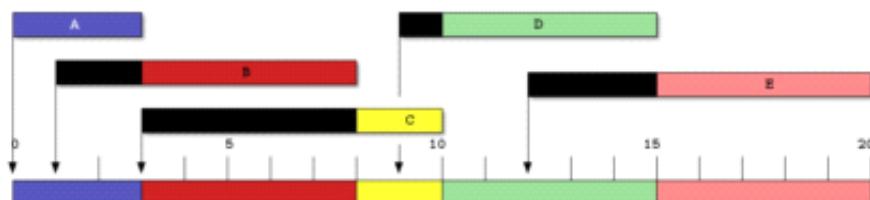


Figure: Primero llegado, primero servido (FCFS)



## Primero llegado, primero servido (FCFS)

- La *sobrecarga administrativa* es mínima.
  - El algoritmo es extremadamente simple: una cola FIFO
  - Efectúa el mínimo posible de cambios de contexto
  - No requiere hardware de apoyo (temporizador / interrupciones)
  - → *Principio de histéresis* (Finkel): "Hay que resistirse al cambio"
- El rendimiento percibido por los últimos procesos disminuye
  - Los procesos cortos pueden esperar desproporcionadamente mucho tiempo



- Los procesos cortos pueden esperar desproporcionadamente mucho tiempo
- La demora aumenta fuertemente conforme crece  $\rho$
- Tendencia a la inanición cuando  $\rho \geq 1$



Gunnar Wolf

Planificación de procesos: Algoritmos de planificación

Introducción  
Métricas

Algoritmos de planificación  
Esquemas híbridos y prioridades externas

## Ronda (Round Robin)

- Busca dar buena respuesta tanto a procesos cortos como largos
- Requiere multitarea preventiva
- Ejecutamos cada proceso por un *quantum*
  - Si no terminó su ejecución, se interrumpe y coloca de vuelta al final de la cola
  - Los procesos nuevos se *forman* también al final de esta misma cola



• □ ▢ ▤ ▦ ▨ ▪ ▫ ▭ ▮ ▯ ▯ ▯

## Ronda (Round Robin) con $q = 4$

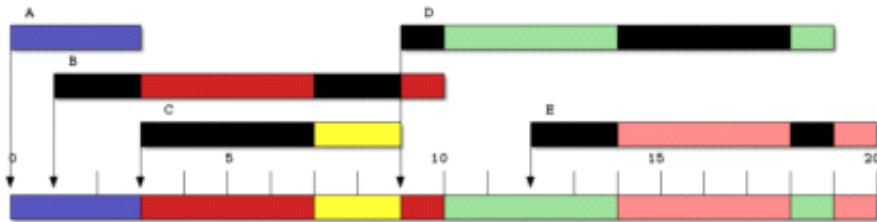


Figure: Ronda (Round Robin), con  $q = 4$



Gunnar Wolf

Planificación de procesos: Algoritmos de planificación

Introducción

Métricas

Algoritmos de planificación

Esquemas híbridos y prioridades externas

## Ronda (Round Robin) con $q = 4$

Proceso	Inicio	Fin	T	E	P
A	0	3	3	0	1.0
B	3	10	9	4	1.8
C	7	9	6	4	3.0
D	10	19	10	5	2.0
E	14	20	8	3	1.6
Promedio			7.2	3.2	1.88



## El proceso más corto a continuación (SPN)

- **Shortest Process Next**
- **Multitarea cooperativa**
- Pero requerimos un algoritmo más  *justo*  que FCFS
- Sabemos cuánto tiempo va a requerir cada proceso
  - No por  *magia* : Podemos estimar / predecir basados en su historia
    - Recuerden: Un proceso puede entrar y salir del ámbito del despachador
  - SPN puede mantener la contabilidad de los procesos



- Recuerden: Un proceso puede entrar y salir del ámbito del despachador

- SPN puede mantener la contabilidad de los procesos incluso tras entregarlos de vuelta al agendador



Gunnar Wolf Planificación de procesos: Algoritmos de planificación

Introducción  
Métricas  
Algoritmos de planificación  
Esquemas híbridos y prioridades externas

## El proceso más corto a continuación (SPN)

- *Shortest Process Next*
- Multitarea cooperativa
- Pero requerimos un algoritmo más  *justo*  que FCFS
- Sabemos cuánto tiempo va a requerir cada proceso
  - No por magia: Podemos estimar / predecir basados en su historia
    - Recuerden: Un proceso puede entrar y salir del ámbito del despachador
  - SPN puede mantener la contabilidad de los procesos incluso tras entregarlos de vuelta al agendador



### Estimando para SPN: Promedio exponencial

Es común emplear un  *promedio exponencial*  para estimar la siguiente demanda de tiempo de  $p$ : Si en su última invocación empleó  $q$  quantums,

$$e'_p = fe_p + (1 - f)q$$

Donde  $0 \leq f \leq 1$  es el  *factor atenuante* , determinando qué tan  *reactivo*  será el promedio a cada cambio.

Es común que  $f \approx 0.9$

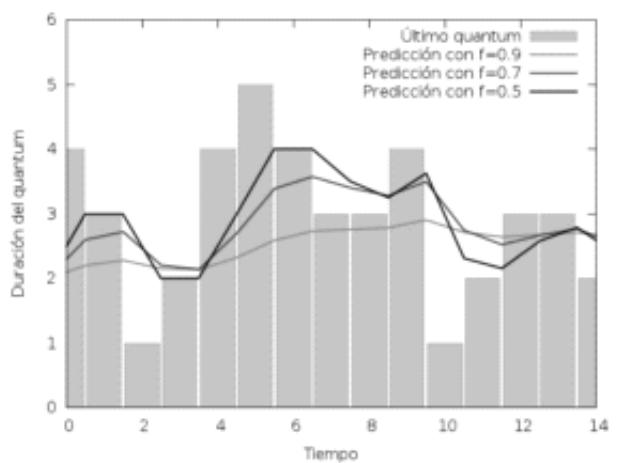


Gunnar Wolf Planificación de procesos: Algoritmos de planificación

Introducción  
Métricas  
Algoritmos de planificación  
Esquemas híbridos y prioridades externas

## Estimando para SPN: Promedio exponencial





*SPN → Sirve sobretodo para procesos cortos*

## Variaciones sobre SPN: El más penalizado a continuación (HPRN)

- *Highest Penalty Ratio Next*
- *Multitarea cooperativa*
  - Las alternativas (FCFS y SPN) parecen injustas para muchos procesos
  - Busca otorgar un mejor balance
- Todos los procesos incian con un valor de penalización  $P = 1$
- Cada vez que un proceso es obligado a esperar un tiempo  $w$  por otro,  $P = \frac{w+t}{t}$  (acumulando  $w$ )
- Se elige el proceso cuyo valor de  $P$  sea mayor



Gunnar Wolf

Planificación de procesos: Algoritmos de planificación

Introducción  
Métricas  
Algoritmos de planificación  
Esquemas híbridos y prioridades externas

## El más penalizado a continuación (HPRN)

- Mientras  $\rho < 1$ , HPRN evita inanición incluso en procesos largos
- Finkel apunta que, ante la experimentación, HPRN se ubica siempre entre FCFS y SPN
- Principal desventaja: Es un algoritmo caro
  - Cuando hay muchos procesos en la cola,  $P$  tiene que calcularse para todos ellos a cada invocación del despachador



- Cuando hay muchos procesos en la cola,  $P$  tiene que calcularse para todos ellos a cada invocación del despachador



# Algoritmos de planificación

Thursday, 19 March 2020 6:08 PM

- FIFO es relativamente similar a Round Robin pero con tiempo infinito

## Mecanismos con múltiples colas

- Hasta ahora, se evalúa **cómo ordenar los procesos en la cola única de procesos listos**
- Dar trato diferenciado a procesos con perfiles distintos es complicado
- ... ¿Y si montamos *distintas colas* de procesos listos?
  - Asignando determinado *patrón de comportamiento* a la migración de una cola a otra
  - Dando un *trato diferenciado* a los procesos de distintas colas



Gunnar Wolf Planificación de procesos: Algoritmos de planificación

Introducción  
Métricas  
**Algoritmos de planificación**  
Esquemas híbridos y prioridades externas

## Mecanismos con múltiples colas

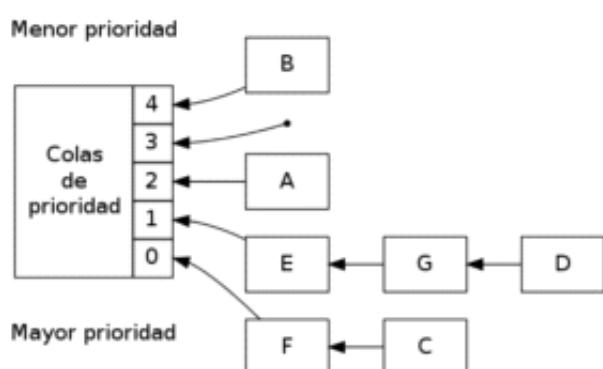


Figure: Representación de un sistema con cinco colas de prioridad y siete procesos listos



## Retroalimentación multinivel (FB)

- Multilevel Feedback
- Multitarea preventiva
- Se crea no una, sino varias colas de procesos listos
  - Cada cola con un distinto nivel de prioridad  $C_n$

## Retroalimentación multinivel (FB)

- Multilevel Feedback
- Multitarea preventiva
  - Se crea no una, sino varias colas de procesos listos
    - Cada cola con un distinto nivel de prioridad,  $C_P$
- El despachador toma el proceso al frente de la cola de más prioridad
- Tras  $n$  ejecuciones, el proceso es degradado a  $C_{P+1}$
- Favorece a los procesos cortos
  - Terminan su trabajo sin ser marcados como de prioridad inferior
- El algoritmo es barato
  - Sólo hay que actualizar a un proceso a cada ejecución, y evaluar un número limitado de colas



Gunnar Wolf

Planificación de procesos: Algoritmos de planificación

Introducción  
Métricas  
Algoritmos de planificación  
Esquemas híbridos y prioridades externas

## Retroalimentación multinivel (FB)

↳ Lo utilizan Windows y Mac OS\*

\* Obvio con variantes

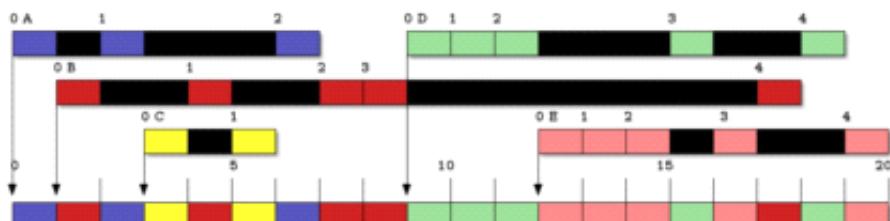


Figure: Retroalimentación multinivel (FB) básica. En la línea superior al proceso se muestra la cola antes del quantum en que se ejecuta.



- En una implementación ingenua provoca cambios de contexto
- En Windows uno de los procesos de mayor importancia es el ratón debido a la importancia que se tiene para el usuario.

## Retroalimentación multinivel (FB)

- ¡Pero todos los números apuntan a que es una peor estrategia que las anteriores!

## Retroalimentación multinivel (FB)

- ¡Pero todos los números apuntan a que es una peor estrategia que las anteriores!
- Los únicos beneficiados son los recién llegados
  - Entran a la cola de mayor prioridad
  - Un proceso largo, a mayor  $\rho$ , enfrenta inanición
- El rendimiento del algoritmo puede ajustarse con dos variables básicas:
  - $n$  Cuántas ejecuciones para ser *degradado* a  $C_{P+1}$
  - $Q$  Duración del *quantum* de las siguientes colas
- Veamos cómo se comporta cuando:
  - Mantenemos  $n = 1$
  - $Q = 2^{nq}$  (donde  $q$  es la duración del *quantum* base)

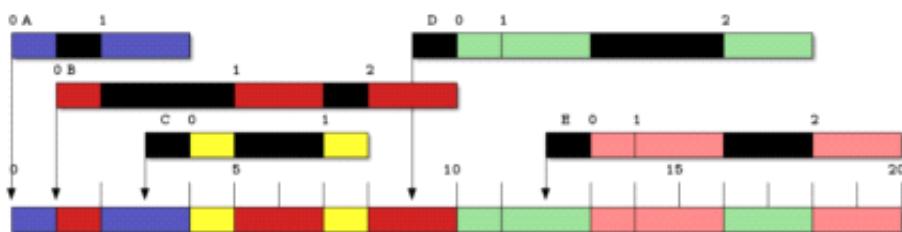
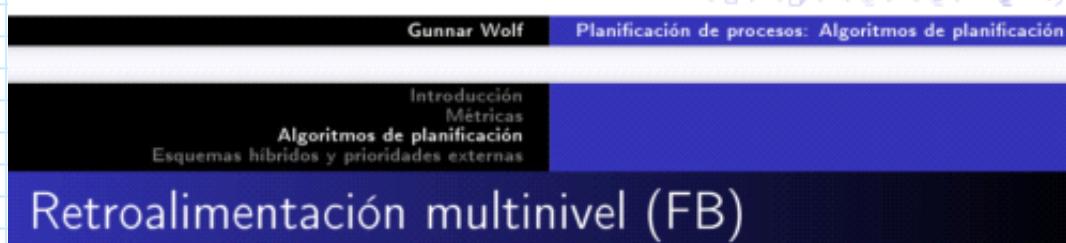


Figure: Retroalimentación multinivel (FB) con  $Q$  exponencial

## Ronda egoísta (SRR)

- *Selfish Round Robin*
- Multitarea preventiva
- Favorece a los procesos que *ya llevan tiempo ejecutando* sobre los recién llegados
  - Un proceso nuevo se forma en la cola de *procesos nuevos*, el despachador avanza sólo sobre los *procesos aceptados*
- Parámetros ajustables:
  - a Ritmo de incremento de prioridad de procesos aceptados
  - b Ritmo de incremento de prioridad de procesos nuevos

b Ritmo de incremento de prioridad de procesos **nuevos**

- Cuando la prioridad de un proceso nuevo alcanza a la de uno aceptado, éste se acepta.



Gunnar Wolf

Planificación de procesos: Algoritmos de planificación

Introducción

Métricas

Algoritmos de planificación

Esquemas híbridos y prioridades externas

## Ronda egoísta (SRR)

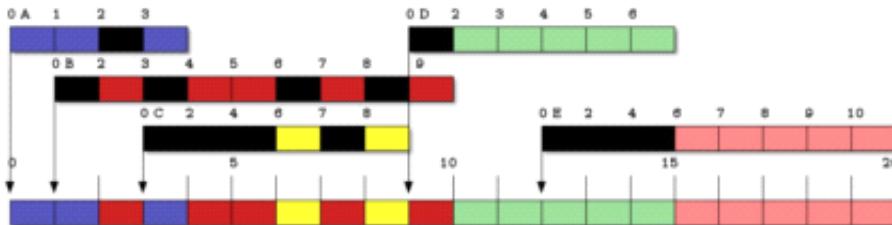


Figure: Ronda egoísta (SRR) con  $a = 2$  y  $b = 1$



## Clasificando a los distintos esquemas

Los siete algoritmos presentados pueden caracterizarse sobre dos descriptores primarios

Tipo de multitarea si el esquema está planteado para operar bajo multitarea *preventiva* o *cooperativa*

Emplea información *intrínseca* Si, para tomar cada decisión de planificación, emplean información propia (intrínseca) a los procesos evaluados, o no — Esto es, si el historial de ejecución de un proceso tiene impacto en cómo será planificado a futuro



Gunnar Wolf

Planificación de procesos: Algoritmos de planificación

Introducción

Métricas

Algoritmos de planificación

Esquemas híbridos y prioridades externas

## Clasificando a los distintos esquemas

Table: Caracterización de los mecanismos de planificación a corto plazo

**Table:** Caracterización de los mecanismos de planificación a corto plazo

	No considera intrínseca	Considera intrínseca
<b>Cooperativa</b>	Primero llegado primero servido (FCFS)	Proceso más corto (SPN), Proceso más penalizado (HPRN)
<b>Preventiva</b>	Ronda (RR)	Proceso más corto preventivo (PSPN), Retroalimentación (FB), Ronda egoísta (SRR)



## Esquemas híbridos

- Los esquemas de planificación empleados normalmente usan *mezclas* de los algoritmos presentados
- Permite emplear el algoritmo que más ventajas presente *ante una situación dada*
  - Y evitar algunas de sus deficiencias



Gunnar Wolf Planificación de procesos: Algoritmos de planificación

Introducción  
Métricas  
Algoritmos de planificación  
Esquemas híbridos y prioridades externas

## Esquemas híbridos: Algoritmo por cola en FB

- Manejamos varias colas en un esquema FB
- Cada cola usa internamente un algoritmo distinto para elegir el proceso que está *a la cabeza*. Algunas ideas como ejemplo:
  - Una cola bajo PSPN: *Empuja* a los procesos más largos hacia colas que sean interrumpidas con menor frecuencia
  - Emplear SRR para las colas de menor prioridad
    - Sus procesos ya esperaron mucho para tener respuesta; cuando obtienen el procesador, avanzan lo más ágilmente posible
    - Pero no obstaculizan a los procesos cortos que van llegando



# Esquemas híbridos: Dependientes del estado del sistema

- Podemos considerar también información *extrínseca* para despachar
  - Información *externa* al estado y ejecución de cada uno de los procesos
  - Información dependiente del estado del sistema, del tipo de usuario, etc.
- A continuación, algunos ejemplos



Gunnar Wolf Planificación de procesos: Algoritmos de planificación

Introducción
Métricas
Algoritmos de planificación
Esquemas híbridos y prioridades externas

## Preventiva o cooperativa, dependiendo de $\rho$

- Si los procesos son *en promedio* cortos y  $\rho < 1$ 
  - Métodos con la mínima sobrecarga administrativa (FCFS o SPN)
  - O un RR con *quantum* muy largo (evitando los problemas de la multitarea cooperativa)
- Si los procesos tienden a ser más largos o si sube  $\rho$ 
  - Cambiamos a RR con un *quantum* más bajo o a PSPN



## Ronda con *quantum* dependiente de procesos pendientes

- Esquema simple de ronda
- La duración de un *quantum* es ajustada periódicamente
- Cada *quantum* depende de la cantidad de procesos en el total de procesos listos, siguiendo  $Q = \frac{q}{n}$
- Pocos procesos esperando
  - Mayor  $Q \rightarrow$  Menos cambios de contexto
- Muchos procesos esperando
  - Menor  $Q$
  - Nunca más allá de un mínimo, para evitar sobrecarga burocrática



Gunnar Wolf

Planificación de procesos: Algoritmos de planificación

Introducción  
Métricas  
Algoritmos de planificación  
Esquemas híbridos y prioridades externas

## Ronda + Prioridad externa

- Usamos un esquema simple de ronda, con una sola cola
- La duración del *quantum* dependerá de la prioridad externa
  - Fijada por el usuario o por el sistema por factores *ajenos* al despachador
- Un proceso de mayor prioridad ejecutará por mayor tiempo



## Peor servicio a continuacion (WSN)

- Generalización sobre HPRN
- No sólo se considera penalización el tiempo esperado en la cola de procesos listos
  - Veces que ha sido interrumpido por el temporizador
  - Prioridad externa
  - Espera por E/S u otros recursos
- El proceso que ha sufrido peor servicio es seleccionado para su ejecución
- Desventaja: Considerar demasiados factores (con distintos pesos) impacta en el tiempo de ejecución del algoritmo
  - Puede llamarse a WSN periódicamente para formar colas



• Desventaja. Considerar demasiados factores (con distintos pesos) impacta en el tiempo de ejecución del algoritmo

- Puede llamarse a WSN periódicamente para formar colas
- Proceder con esquemas más simples
- ... Aunque esto reduce la velocidad de reacción



Gunnar Wolf

Planificación de procesos: Algoritmos de planificación

Introducción  
Métricas

Algoritmos de planificación  
Esquemas híbridos y prioridades externas

## Lindura (niceness)

- Empleado por varios Unixes históricos
- El usuario inicia (nice) o modifica (renice) la prioridad de su proceso
  - Típicamente sólo *hacia arriba* — Se porta *más lindo*.
- Esta prioridad *externa* y el tiempo consumido recientemente por el proceso constituyen una prioridad *interna*
- La prioridad interna aumenta cuando el proceso espera
  - Por el despachador, por E/S, o cualquier otra causa
- La prioridad interna es matizada por el tamaño de la cola de procesos listos
  - Entre más procesos pendientes, mayor el peso que modifique a la prioridad



- Linux Kernel 2.6...
  - Medidas de tiempo para los procesos, en específico por CFS
  - Descriptor de Hilo o de procesos es una estructura hecha en C
  - Árbol Rojo-Negro → Varante de Árbol Binario

# Temas relacionados

Tuesday, 24 March 2020 5:47 PM

## Comparando los distintos algoritmos

- Los ejemplos que presentamos son sólo con *una distribución arbitraria* de procesos
  - Diferentes distribuciones llevarán necesariamente a distintos resultados
- Para comparar *de forma significativa* los distintos mecanismos, habría que analizar con muy distintas cargas
- Raphael Finkel compara en *An operating systems vade mecum* (1988) lo observado en distintos conjuntos de procesos, bajo los diferentes mecanismos



Gunnar Wolf

Planificación de procesos: Temas relacionados

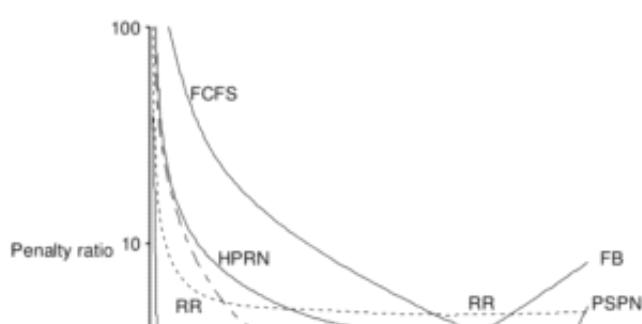
Afinando al despachador  
El despachador y los hilos  
El despachador y el multiprocесamiento  
Planificación de tiempo real

## Midiendo la respuesta de distintos algoritmos

- Parámetros de los procesos:
  - Generados de forma aleatoria, obedeciendo a distribución exponencial
  - Simulación de 50,000 procesos
  - $\alpha = 0,8$ ,  $\beta = 1,0 \rightarrow \rho = 0,8$
- Finkel apunta a que una *verdadera* comparación debería ser tridimensional (variando  $\rho$ )
  - Pero presentan ya un acercamiento útil acerca de su comportamiento general



## Proporción de penalización (P)



## Proporción de penalización (P)

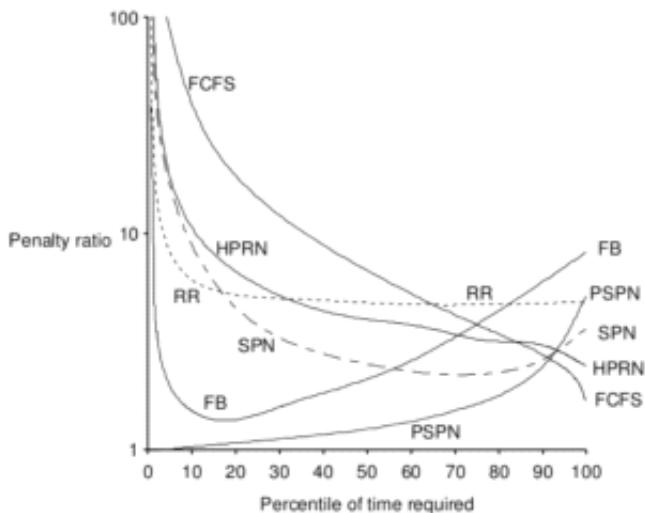


Figura: Proporción de penalización contra porcentaje de tiempo requerido en despachadores (Finkel, p.33)

Gunnar Wolf

Planificación de procesos: Temas relacionados

Afinando al despachador  
El despachador y los hilos  
El despachador y el multiprocesamiento  
Planificación de tiempo real

## Tiempo perdido (E)

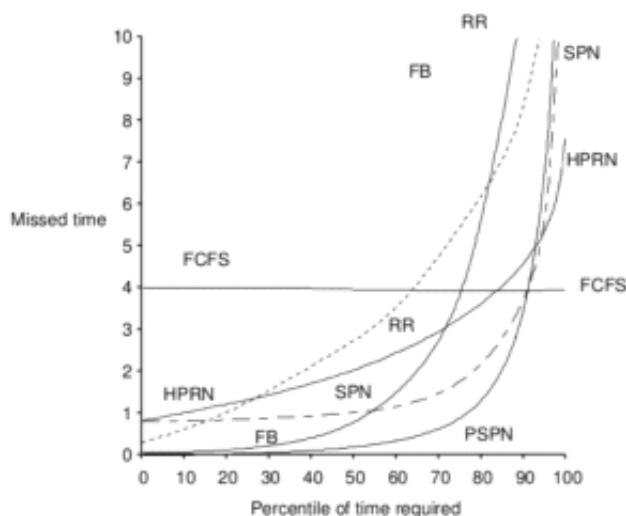


Figura: Tiempo perdido contra porcentaje de tiempo requerido en despachadores (Finkel, p.34)

## Duración mínima del quantum

- Vimos que la penalización en la ronda puede evitarse con *quantums* mayores
  - Pero puede degenerar en multiprocesamiento cooperativo
- ¿Qué tan corto debe ser un *quantum*?

## Duración mínima del quantum

- Vimos que la penalización en la ronda puede evitarse con *quantums* mayores
  - Pero puede degenerar en multiprocesamiento cooperativo
- ¿Qué tan corto debe ser un *quantum*?



Gunnar Wolf

Planificación de procesos: Temas relacionados

Afinando al despachador  
El despachador y los hilos  
El despachador y el multiprocesamiento  
Planificación de tiempo real

## Duración mínima del quantum

- Duración de cambio de contexto *hoy en día*:  $\approx 0,01ms$  (Silberschatz p.187)
  - Con un *quantum* corto, 10ms, es  $\frac{1}{1000}$  la duración del tiempo efectivo de procesamiento
  - Con un *quantum* largo, 200ms,  $\frac{1}{20000}$
- No es realmente significativo — ¡Ni debe serlo!
  - Perder 1 % del tiempo de cómputo en *burocracia* sería en general visto como excesivo
  - *Ojo*: ¡El tiempo núcleo *no sólo* es el tiempo del cambio de proceso!



→ Relación entre hilos y procesos

## Muchos a uno

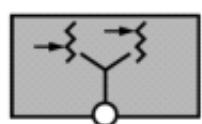


Figura: Mapeo de hilos *muchos a uno* (imagen: Beth Plale)

- Hilos verdes (o de usuario)
- El SO ve un sólo proceso
  - El tiempo se reparte dentro del proceso por mecanismos

## Muchos a uno

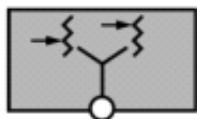


Figura: Mapeo de hilos *muchos a uno* (imagen: Beth Plale)

- **Hilos verdes (o de usuario)**
- El SO ve un sólo proceso
  - El tiempo se reparte dentro del proceso por mecanismos internos
- **Código más portable**
- **No se aprovecha realmente el paralelismo**
- Llamadas bloqueantes → Todos esperan



Gunnar Wolf

Planificación de procesos: Temas relacionados

Afinando al despachador  
El despachador y los hilos  
El despachador y el multiprocesamiento  
Planificación de tiempo real

## Uno a uno

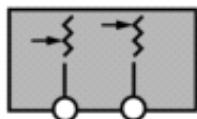


Figura: Mapeo de hilos *uno a uno* (imagen: Beth Plale)

- Cada hilo es un *proceso ligero* (*lightweight process, LWP*)
  - Un LWP es mucho más ligero de levantar que un proceso
  - Comparte memoria, descriptores, estructuras
- **Aprovecha tanto el paralelismo como lo permite el hardware**
  - Cada hilo puede correr en un procesador distinto, si los hay
- El SO debe poder manejar LWP



## Muchos a muchos

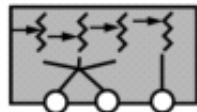


Figura: Mapeo de hilos *muchos a muchos* (imagen: Beth Plale)

- Existen *hilos unidos (bound threads)*, que corresponden cada uno a un LWP
- También *hilos no unidos (unbound threads)*, donde *uno o más* corresponden a cada LWP
- Brindan la flexibilidad de ambos modelos
  - Si el SO no maneja LWP, pueden caer en el modelo *uno a muchos* como modo degradado



Gunnar Wolf

Planificación de procesos: Temas relacionados

Afinando al despachador  
El despachador y los hilos  
El despachador y el multiprocesamiento  
Planificación de tiempo real

## El ámbito de contención

## El ámbito de contención

¿Recibe cada uno de los hilos *la misma atención* que recibiría un proceso? Hay dos enfoques (categorización de hilos POSIX, pthread).

- Ámbito de contención de sistema (System Contention Scope, SCS)
- Ámbito de contención de proceso (Process Contention Scope, PCS)

El *ámbito de contención* se refiere a cuál será la estructura dentro de la cual coexisten los hilos, y dentro de la cual cada hilo contendrá (competirá) por el procesador.



## Ámbitos de contención

### PTHREAD\_SCOPE\_SYSTEM

- Todos los hilos son atendidos en el tiempo que sería asignado a *un sólo proceso*
- Modelo *muchos a uno*, así como los *hilos no unidos* multiplexados en *muchos a muchos*

... Pero una implementación pthreads puede ofrecer sólo *uno de los modelos* → Tanto Linux como Windows manejan sólo PTHREAD\_SCOPE\_SYSTEM (SCS).

### PTHREAD\_SCOPE\_PROCESS

- Cada hilo es visto por el planificador como un proceso independiente
- Modelo *uno a uno* y los *hilos unidos en muchos a muchos*



Gunnar Wolf

Planificación de procesos: Temas relacionados

Afinando al despachador  
El despachador y los hilos  
El despachador y el multiprocesamiento  
Planificación de tiempo real

## Otras características en pthreads

- pthreads incluyen varios de los otros aspectos mencionados en esta unidad
- El programador puede solicitar al núcleo la prioridad de cada uno de los hilos por separado (pthread\_setschedprio)
- Incluso solicitar el empleo de un algoritmo de planificación en específico (sched\_setscheduler)
- ... Pero recordemos que pthreads permite, en muchos casos, responder al proceso «*Disculpa, no puedo hacer eso*»





## Compartir el procesador

- Estrategia empleada por Control Data (CDC6600, 1964, diseñada por Seymour Cray): Multitarea gestionada en hardware
- Un sólo procesador, pero con 10 juegos de registros
  - Procesador superescalar
- A cada paso de reloj, avanzaba el *juego de registros*
  - Quantum efectivo igual a la velocidad del reloj
  - Apariencia de 10 procesadores más lentos, cada uno de  $\frac{1}{10}$  la velocidad del *real*
  - Multitarea sin cambios de contexto



Gunnar Wolf

Planificación de procesos: Temas relacionados

Afinando al despachador  
El despachador y los hilos  
El despachador y el multiprocesamiento  
Planificación de tiempo real

## Compartir el procesador

- ¿Desventajas?
  - Nivel de concurrencia fijo
  - Difícil de adecuar a picos de ocupación
  - Costo muy elevado
- Esquema comparable con el *HyperThreading* de procesadores actuales
  - Aunque *HyperThreading* busca aprovechar las diferentes fases del *pipeline*, que no existía en 1964
  - Hoy en día... Mera curiosidad histórica.



## Multiprocesamiento: Afinidad

- Cuando un proceso se ejecutó por cierto tiempo, dejó porciones de su espacio de memoria en el caché del procesador
  - Tanto segmentos de instrucciones como de datos

## Multiprocesamiento: Afinidad

- Cuando un proceso se ejecutó por cierto tiempo, dejó porciones de su espacio de memoria en el caché del procesador
  - Tanto segmentos de instrucciones como de datos
- Cada procesador tiene *usualmente* un caché independiente
  - Puede haber un caché común a todo el sistema (L3); puede haber uno por chip multicore (L2), y puede haber uno específico para cada núcleo (L1)
- Despachar a un proceso en un procesador distinto del que ya lo ejecutó es un gasto innútil
  - Inicializar el caché que empleó
  - Re-poblar el nuevo

NUMA  $\triangleq$  Non Uniform Memory Access



^

Gunnar Wolf Planificación de procesos: Temas relacionados

Afinando al despachador  
El despachador y los hilos  
El despachador y el multiprocesamiento  
Planificación de tiempo real

## Multiprocesamiento: Afinidad

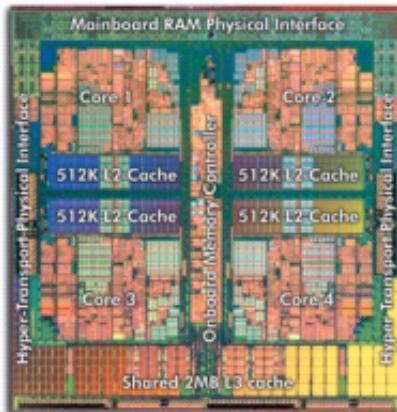


Figura: Fotografía de un CPU AMD Opteron Barcelona (2007) (Imagen: <http://www.elnexus.com/articles/barcelona.aspx>)

taskset --pid \$

## Multiprocesamiento: Afinidad

**Afinidad suave** Un proceso que *preferentemente* será ejecutado en un determinado procesador

Ciertos patrones de carga pueden llevar a que el despachador decida *migrarlo* a otro procesador

**Afinidad dura** Se *garantiza* que un proceso será ejecutado siempre en un procesador (o conjunto de procesadores)

## Multiprocesamiento: Afinidad

**Afinidad suave** Un proceso que *preferentemente* será ejecutado en un determinado procesador

Ciertos patrones de carga pueden llevar a que el despachador decida *migrarlo* a otro procesador

**Afinidad dura** Se *garantiza* que un proceso será ejecutado siempre en un procesador (o conjunto de procesadores)

Ejemplo: En un entorno NUMA, buscamos que los procesos tengan *afinidad dura* (y que el algoritmo de asignación de memoria considere dicha relación)



Gunnar Wolf

Planificación de procesos: Temas relacionados

Afinando al despachador  
El despachador y los hilos  
El despachador y el multiprocesamiento  
Planificación de tiempo real

## Multiprocesamiento: Balanceo de cargas

- La situación ideal es que todos los procesadores despidan trabajos al 100 % de su capacidad
  - Pero es un requisito demasiado rígido / irreal
  - (casi) siempre habrá un procesador con tiempo libre
  - (casi) siempre habrá procesadores con procesos encolados y en espera
  - ... O ambas situaciones
- Para mantener la divergencia lo menor posible, se emplea el *balanceo de cargas*



## Multiprocesamiento: Balanceo de cargas

- El balanceo de cargas *actúa en sentido contrario* de la afinidad al procesador
  - Algoritmos que analizan el estado de las colas y transfieren procesos entre ellas para homogeneizarlas
- Puede reubicar procesos con *afinidad suave*
  - Debe preferir reubicar procesos *sin afinidad declarada*
  - No debe reubicar procesos con *afinidad dura*

## Multiprocesamiento: Balanceo de cargas

- El balanceo de cargas actúa en sentido contrario de la afinidad al procesador
  - Algoritmos que analizan el estado de las colas y transfieren procesos entre ellas para homogeneizarlas
- Puede reubicar procesos con afinidad suave
  - Debe preferir reubicar procesos sin afinidad declarada
  - No debe reubicar procesos con afinidad dura



Navigation icons: back, forward, search, etc.

Gunnar Wolf

Planificación de procesos: Temas relacionados

Afinando al despachador  
El despachador y los hilos  
El despachador y el multiprocesamiento  
Planificación de tiempo real

## Estrategias de balanceo de cargas: Por empuje

Balanceo de cargas por empuje (*push migration*)

- El núcleo revisa periódicamente el estado de los procesadores
- Si el desbalance sobrepasa cierto umbral, *empuja* a algunos procesos de una cola a otra.
- Linux ejecuta esto cada 200ms.



## Estrategias de balanceo de cargas: Por jalón

Balanceo de cargas: Por jalón (*pull migration*)

- Cuando un procesador queda sin tareas pendientes, ejecuta el proceso especial *desocupado (idle)*
- *Desocupado* no significa *no hacer nada*: Puede ejecutar tareas del núcleo
- Puede averiguar si hay procesos en espera con otro procesador, y *jalarlos* al actual.

## Estrategias de balanceo de cargas: Por jalón

### Balanceo de cargas: Por jalón (*pull migration*)

- Cuando un procesador queda sin tareas pendientes, ejecuta el proceso especial *desocupado (idle)*
- *Desocupado* no significa *no hacer nada*: Puede ejecutar tareas del núcleo
- Puede averiguar si hay procesos en espera con otro procesador, y jalarlos al actual.

Los mecanismos no son mutuamente exclusivos, es común que un SO emplee ambos.

Todo balanceo de cargas conllevará una penalización en términos de afinidad al CPU.



Gunnar Wolf Planificación de procesos: Temas relacionados

Afinando al despachador  
El despachador y los hilos  
El despachador y el multiprocesamiento  
Planificación de tiempo real

Multiprocesamiento: ¿Una cola o varias?

En un entorno multiprocesador, ¿cómo encaramos los algoritmos antes descritos?

- Una cola global
  - Parecería una decisión más simple
  - Se ejecuta *un sólo despachador* → Ahorro de tiempo
  - No habría que preocuparse por balanceo de cargas — Sería natural
- Una cola por procesador
  - Necesaria si queremos soportar manejo de afinidad al CPU
  - Todos los sistemas en uso amplio implementan una cola por procesador



6:45 PM Tue 24 Mar

El despachador y el multiprocesamiento  
Planificación de tiempo real

## Hilos hardware (*hyperthreading*)

- Hilos es una palabra que sufre abuso en nuestro campo.
- *Hilos hardware (hyperthreading)*: No tienen relación con los hilos que abordamos
  - Pero sí con el multiprocesamiento
- La Ley de Moore no sólo ha llevado a un paralelismo expreso (multinúcleo)
  - El *pipeline* de los procesadores hace que cada *unidad funcional* del CPU pueda estar atendiendo a una instrucción distinta

## Hilos hardware (*hyperthreading*): Pipelines

- Un procesador simple/clásico (ejemplo: MIPS) puede dividir la ejecución de una instrucción en 5 etapas:
  - IF Recuperación de la instrucción (*Instruction Fetch*)
  - ID Decodificación de la instrucción (*Instruction Decode*)
  - EX Ejecución (*Execution*)
  - MEM Acceso a datos
  - WB Almacenamiento (*Writeback*)

## Hilos hardware (*hyperthreading*): Pipelines

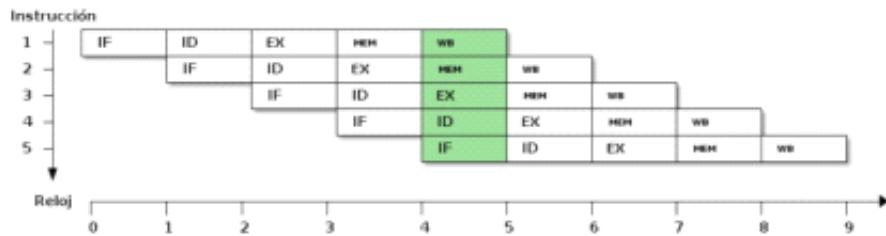


Figura: Descomposición de una instrucción en sus cinco pasos clásicos para organizarse en un *pipeline*



Gunnar Wolf Planificación de procesos: Temas relacionados

Afinando al despachador  
El despachador y los hilos  
El despachador y el multiprocesamiento  
Planificación de tiempo real

## Hilos hardware (*hyperthreading*): Pipelines

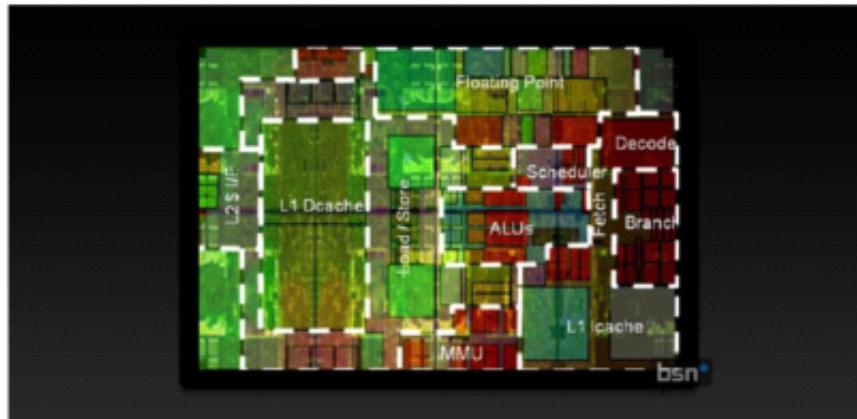
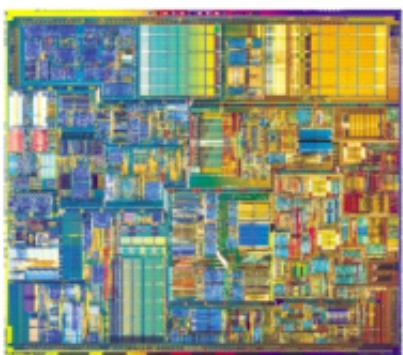


Figura: Fotografía de un *core* nVidia Denver (Imagen:  
<http://www.brightsideofnews.com/news/2011/3/9/nvidia-reveals-64-bit-project-denver-cpu-silicon-die>.

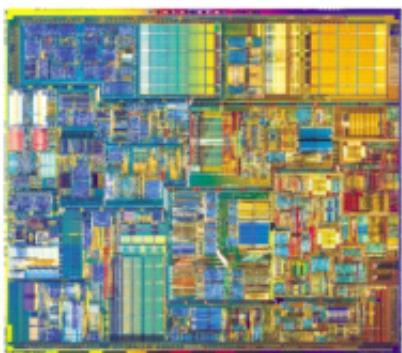


## Hilos hardware (*hyperthreading*): Pipelines



Un procesador moderno presenta mucho más etapas (Pentium 4: 20 etapas)

## Hilos hardware (*hyperthreading*): Pipelines



Un procesador moderno presenta mucho más etapas (Pentium 4: 20 etapas)

Figura: Fotografia de un CPU Intel Pentium 4

(Imagen: Calvin College 2005, <http://www.calvin.edu/academic/rit/webBook/chapter2/design/hardware/cpu/look.htm>)



Gunnar Wolf Planificación de procesos: Temas relacionados

Afinando al despachador  
El despachador y los hilos  
El despachador y el multiprocesamiento  
Planificación de tiempo real

## Hilos hardware (*hyperthreading*): Pipelines

- Es común que se presenten patrones de uso que requieren servicio de diferentes componentes del procesador
  - A veces con diferentes duraciones
  - Lleva a la inserción de demasiadas *burbujas* → Pérdida de tiempo
- Para remediarlo, un sólo procesador (un solo núcleo) se presenta como compuesto de dos o más *hilos hardware*
  - Conocidos en el mercado como *hyper-threads*
  - Puede aumentar el paralelismo — jaunque es muy improbable que sea en 2x!



## Hilos hardware (*hyperthreading*): Pipelines

- Es común que se presenten patrones de uso que requieren servicio de diferentes componentes del procesador
  - A veces con diferentes duraciones
  - Lleva a la inserción de demasiadas *burbujas* → Pérdida de tiempo
- Para remediarlo, un sólo procesador (un solo núcleo) se presenta como compuesto de dos o más *hilos hardware*
  - Conocidos en el mercado como *hyper-threads*
  - Puede aumentar el paralelismo — jaunque es muy improbable

- Conocidos en el mercado como *hyper-threads*
- Puede aumentar el paralelismo — aunque es muy improbable que sea en 2x!



## Hilos hardware (*hyperthreading*): Pipelines

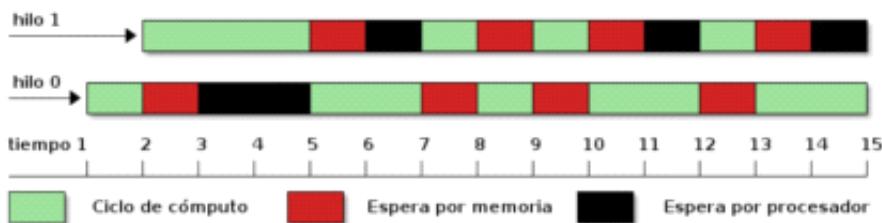


Figura: Alternando ciclos de cómputo y espera por memoria, un procesador que implementa hilos hardware (*hyperthreaded*) se presenta como dos procesadores



## ► Tiempo Real

### ¿Qué es el *tiempo real*?

- Nos hemos enfocado a repartir el tiempo disponible entre varios procesos
- No hemos tocado a los procesos que requieren *garantías de tiempo*
  - Para poder realizar su trabajo, tienen que recibir determinado tiempo de procesador *antes de un tiempo límite*
- Estos procesos son conocidos como *de tiempo real*



## ¿Cuándo nos enfrentamos con el tiempo real?

### Controladores de dispositivos

- Entregan un determinado *bloque* de información cada tanto *tiempo*
- Si ese bloque no es procesado completo, se sobrescribe por el siguiente

### Reproductores o recodificadores de audio y video

- Si un *cuadro* se pierde, el resultado puede escucharse
- Sea como una demora (reproducción) o como un corte (recodificación)

### Procesadores de criptografía

- Si un bloque se pierde, el documento completo *desde ese punto en adelante* puede quedar corrupto



## Reserva de recursos

### Para poder manejarse como de tiempo real, un proceso debe declarar de inicio cuánto tiempo requerirá

- Puede ser una sola vez: *Necesito 600ms en los próximos 2s*
- Puede ser periódico: *Cada segundo necesito 30ms*

### El sistema operativo le asignará el tiempo solicitado, o le notificará que no puede garantizárselo

- Mensaje de error → El proceso puede intentar continuar de todos modos *sin prioridad especial*
- O puede notificar al usuario, antes de haber gastado tiempo, que no podrá realizar la operación exitosamente en tiempo.



## Tiempo real duro y suave

Tiempo real duro *El sistema puede dar garantía de que el proceso tendrá el tiempo que reservó*

Tiempo real suave *El sistema intentará dar la prioridad requerida al*

tendrá el tiempo que reservó

Tiempo real suave El sistema intentará dar la prioridad requerida al proceso, pero puede haber pequeñas demoras

## Restricciones del tiempo real duro

- El planificador debe saber con certeza cuánto tiempo toman todas las tareas de sistema que ejecutará en el periodo
- Algunos dispositivos introducen demoras con demasiada varianza
  - Almacenamiento en disco
  - Memoria virtual
- Imposibilitan que un sistema que los maneje implemente tiempo real duro



Gunnar Wolf

Planificación de procesos: Temas relacionados

Afinando al despachador  
El despachador y los hilos  
El despachador y el multiprocesamiento  
Planificación de tiempo real

## Tiempo real suave: Prioridad del planificador

- Los procesos de tiempo real simplemente reciben una prioridad mucho más alta ante el planificador
- Los procesos de tiempo real pueden llevar a la inanición de otros procesos
  - Es esperable y aceptable — ¡No debemos correr tantos procesos de tiempo real! (rompería expectativas)

Retroalimentación  
Multinivel

Colas no son  
de tiempo  
real duro

Mac Os  
Windows



## Retroalimentación multinivel y tiempo real suave

Puede implementarse con una modificación al esquema de retroalimentación multinivel

- La cola de tiempo real recibe prioridad sobre todas las demás colas

- La cola de tiempo real recibe prioridad sobre todas las demás colas
- La prioridad de un proceso de tiempo real *no se degrada* conforme se ejecuta repetidamente
  - Aunque puede indicar que ya terminó con su trabajo sensible a tiempo
  - El SO podría entonces *reclasificar* al proceso con una prioridad normal



Gunnar Wolf

Planificación de procesos: Temas relacionados

Afinando al despachador  
El despachador y los hilos  
El despachador y el multiprocesamiento  
Planificación de tiempo real

## Retroalimentación multinivel y tiempo real suave

- La prioridad de los demás procesos *nunca llegan a subir* al nivel de tiempo real por un proceso automático
  - Aunque sí puede hacerse por una llamada explícita
- La latencia de despacho debe ser mínima

Casi todos los sistemas operativos hoy en día presentan facilidades básicas de tiempo real suave.



## Sistema operativo interrumpible (*prevenible*)

- Hay llamadas al sistema que toman demasiado tiempo
- Para poder ofrecer tiempo real suave con buenas expectativas, *hay que poder interrumpir* al propio núcleo
- Primer enfoque: *Puntos de interrupción*
  - Marcar explícitamente puntos en que todas las estructuras están en un estado estable
- No muy eficiente: *Hay pocos puntos aptos para declarar puntos de interrupción*
  - Muy rígido, difícil estructurar para poner puntos adicionales



Gunnar Wolf

Planificación de procesos: Temas relacionados

Afinando al despachador  
El despachador y los hilos

## Núcleo completamente interrumpible

- Otro enfoque: **Proteger a todas las modificaciones a estructuras internas del núcleo con mecanismos de sincronización**
- Enfoque mucho más flexible
- Hace que el sistema operativo *completo* sea más lento (aunque más seguro)
  - Más operaciones a hacer por cada solicitud
- **Permiten que funciones del SO puedan correr como hilos concurrentes en los distintos procesadores**

