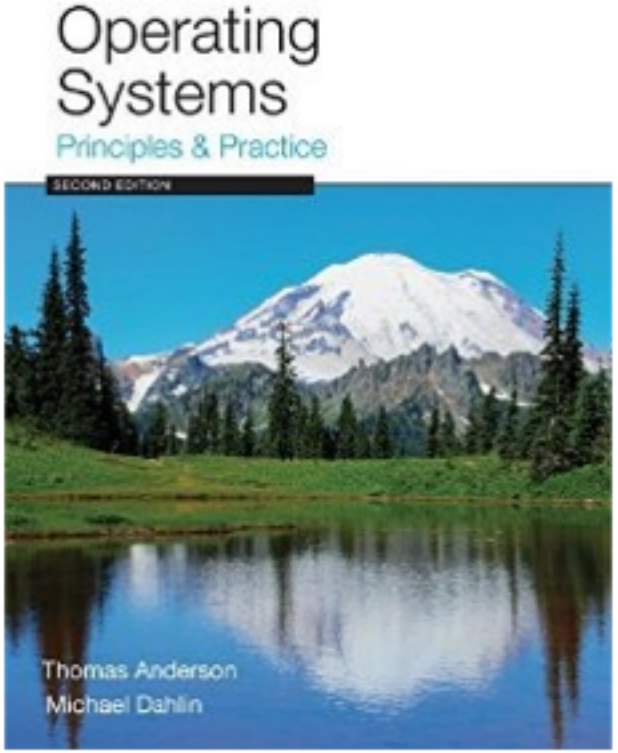
Sistemas Operativos

Capítulo 7 Itineración

Prof. Javier Cañas R.

Estos apuntes están tomados en parte del texto: “Operating System: Principles and Practice” de T. Anderson y M. Dahin



Motivación

- Cuando tenemos muchas tareas que hacer, ¿cuál hacemos primero?
- En cualquier instante de tiempo, algunos threads están corriendo en el procesador, otros esperan su turno, y otros están en espera. La intineración (scheduling) del procesador determina cuál se elige para correr cuando hay procesador disponible.
- El criterio utilizado para elegir, determina la naturaleza del SO.

Temario

1. Introducción
2. Itineración FIFO
3. Itineración SJF
4. Itineración Round Robin
5. Colas Multinivel Realimentadas (MFQ)
6. Itineración en Multiprocesadores

1 Introducción

- Cualquier política de itineración presenta un complejo conjunto de decisiones entre varias propiedades deseables.
- La itineración se aplica en cualquier contexto de recursos escasos, por ejemplo uso de discos, red, energía y por supuesto el procesador.
- Por ejemplo, para Google, Amazon y Yahoo un aumento del tiempo de respuesta de 100mseg, significa la pérdida del 5% al 10% de clientes.

Definiciones básicas

- **Task**: tarea o job. Es cualquier requerimiento de usuario. Un thread o proceso, podría ser responsable de múltiples task.
- **Tiempo de respuesta** (retardo): El tiempo percibido por el usuario que toma el sistema en hacer una tarea.
- **Predictibilidad**: Baja varianza en tiempos de respuesta para requerimientos repetidos.

… Definiciones básicas

- **Overhead de Itineración**: Tiempo en pasar de una tarea a otra.
- **Equidad**: Igualdad en los tiempos y recursos entregados a cada tarea.
- **Inanición** (Starvation): Una tarea no avanza debido a que los recursos están asignados a tareas de mayor prioridad.

… Definiciones básicas

- **Carga de trabajo (Workload)**: Conjunto de tareas a realizar por el sistema. Considera tiempos de arribo, tiempo que toma el ser completada. Dado una carga de trabajo, el itinerador decide en qué momento a cada tarea se le asigna un procesador.

- **Itineración Interrumpible** (Preemtive): Los recursos se pueden quitar por otra tarea.

- **Conservación de trabajo**: Un algoritmo de itineración es conservador de trabajo si nunca deja al procesador ocioso cuando hay trabajo que hacer. Un ejemplo a imitar!

Criterios de Optimización

- Máxima utilización de CPU
- Máxima productividad (throughput)
- Mínimo turnaround time (tiempo entre que el proceso llega hasta que termina)
- Mínimo tiempo de espera
- Mínimo tiempo de respuesta

Itineración de Uniprocesadores

- El análisis de algoritmos de itineración comenzará analizando el caso de un solo procesador.

- Se comenzará con 3 algoritmos:
  - FIFO (First In First Out)
  - SJF (El más corto primero)
  - Round Robin

- Asumiremos que el itinerador tiene la habilidad de interrumpir el procesador y asignarle otra tarea. Esto puede ocurrir por interrupción de tiempo o por llegada una tarea de mayor prioridad.

Perfiles de carga

- Las tareas tienen perfiles distintos. Se pueden clasificar en dos conjuntos:
  - **Delimitadas por computación (CPU-bound)**: Uso intensivo del procesador con breves momentos de E/S.
    - Ej. solución numérica de una PDE
  - **Delimitadas por E/S (I/O-bound)**: La mayor parte del tiempo se está esperando por E/S con breves momentos de cálculo.
    - Ej. descarga de BitTorrent
  - **Mezcla (CPU+I/O)**: compilador, navegador

Quitar CPU a un proceso (Preemption)

- Asumiremos que el itinerador tiene la facultad de quitar el procesador a un proceso y asignarlo a otro.
- Esto puede ocurrir por interrupción de tiempo o porque llegó otra tarea con mayor prioridad a la Ready List.

2 FIFO (First In- First Out)

- También se denomina FCFS (First Come, First Served) o algoritmo de la peluquería.
- Cuando se elige una tarea, se continua hasta que ésta termina (o debe esperar por E/S). Cada tarea espera pacientemente por su turno. También favorece la equidad.
- FIFO minimiza el overhead que significa el context switch, y tiene el mejor throughput.

Ejemplo

Supongamos que los procesos llegan en el orden: P1, P2, P3

Proceso	Tiempo CPU
P1	24
P2	3
P3	3

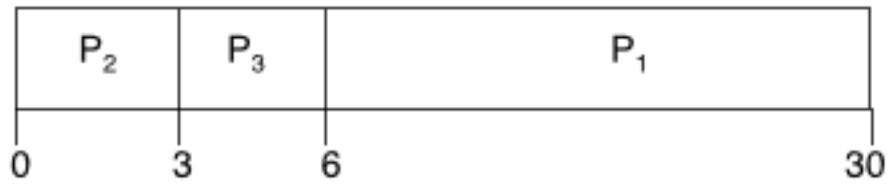
La carta Gantt para la itineración es:

Los tiempos de espera para cada proceso son: P1=0, P2=24, P3=27. El tiempo de espera promedio es: (0 + 24 + 27)/3 = 17

Observaciones

- FIFO tiene una debilidad: cuando muchas tareas cortas están detrás de una tarea grande, se hace muy ineficiente (efecto convoy).

- En el ejemplo anterior, si se reordenan las tareas comenzando por las más cortas:

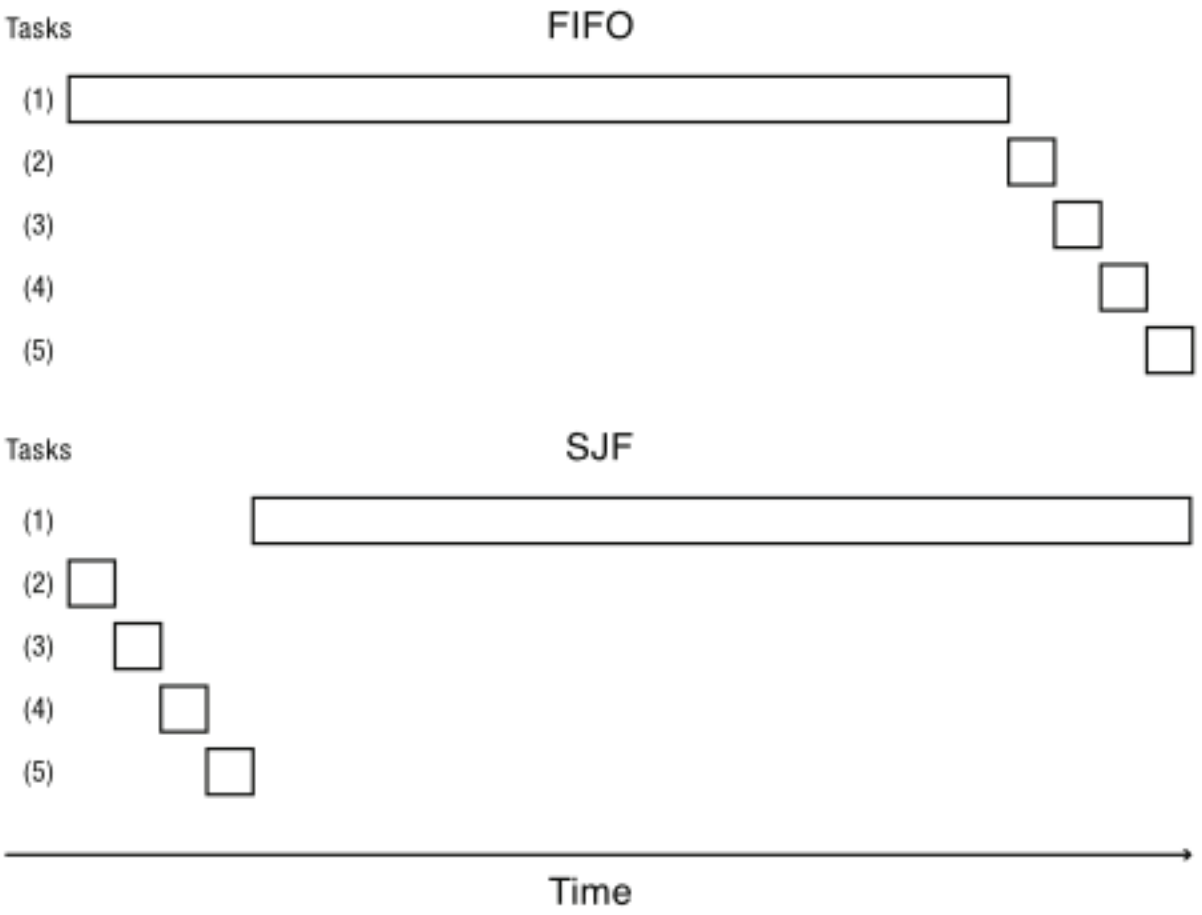


Los tiempos de espera para cada proceso son: P1=6, P2=0, P3=3. El tiempo de espera promedio es:
(6 + 0 + 3)/3 = 3

3 La tarea más corta primero (SJF)

- El algoritmo óptimo para minimizar el tiempo de respuesta promedio es SJF. Se favorece a las tareas más cortas.
- El algoritmo SJF elige primero la tarea que le queda menos tiempo restante para terminar. En otras palabras SJF no elige la tarea con menos tiempo, sino la que le queda menos por hacer.
- SJF sufre de starvation y frecuentes context switch si llegan un número suficiente de tareas cortas.
- Tampoco es posible leer el futuro para saber cuánto tiempo le queda a una tarea, pero se puede estimar.

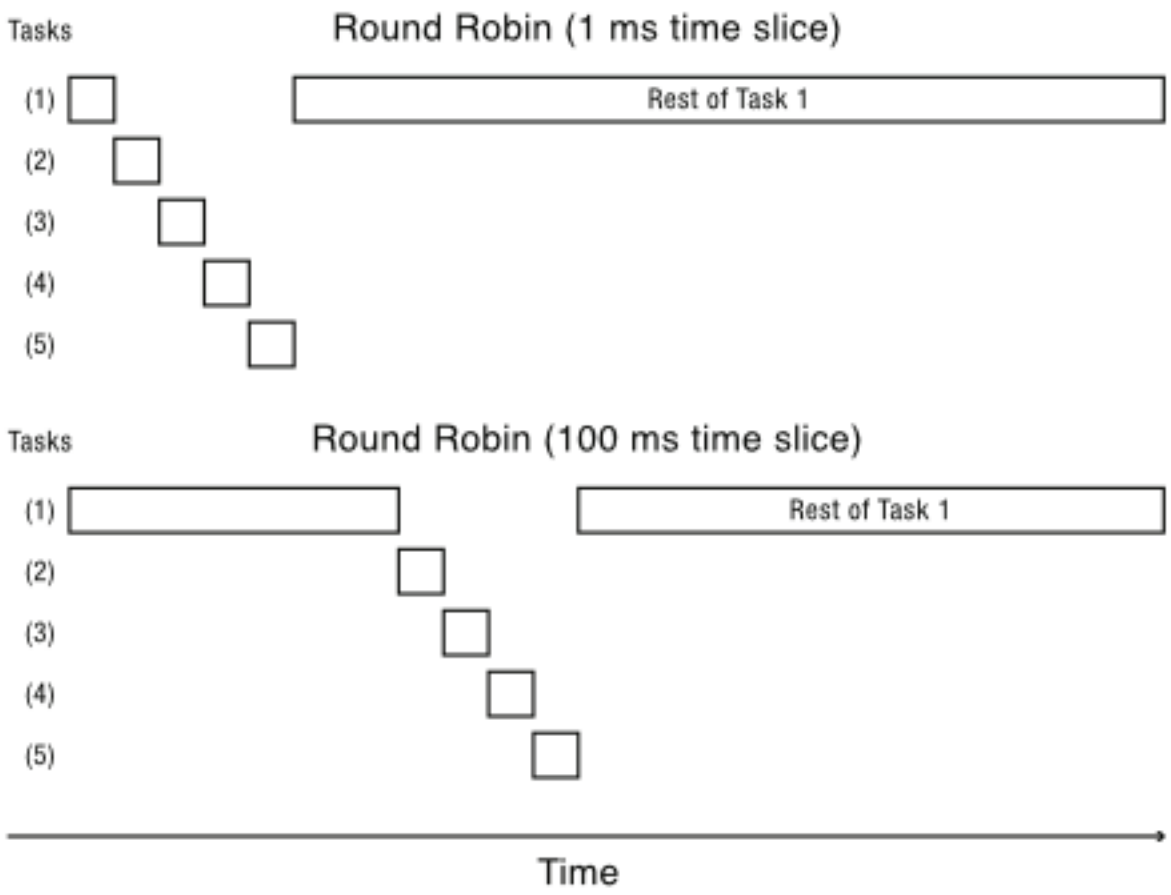
FIFO vs SJF



4 Round Robin (RR)

- Las tareas se turnan en períodos de tiempo limitados.
- El tiempo asignado a cada tarea se denomina **quantum de tiempo (q)**. Al final del quantum de tiempo, si la tarea no ha terminado, es interrumpida y el scheduler selecciona la siguiente.
- No hay starvation.

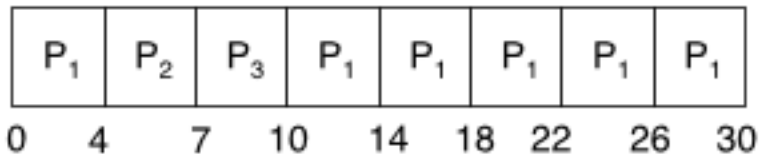
RR con q=1mseg y q=100mseg



Ejemplo de RR con tq=4

Proceso	Burst Time
P1	24
P2	3
P3	3

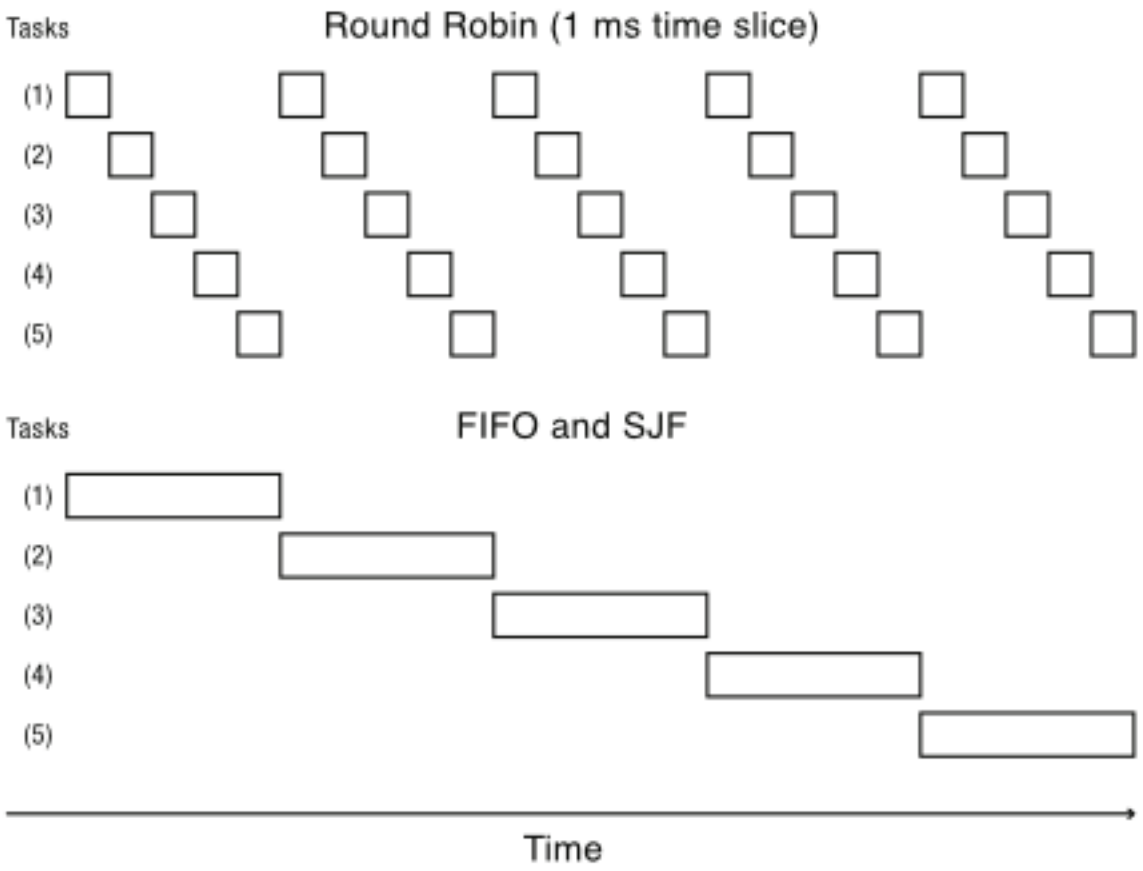
Carta Gantt:



¿Turnaround time promedio?

¿Tiempo de espera promedio?

RR vs FIFO y SJF



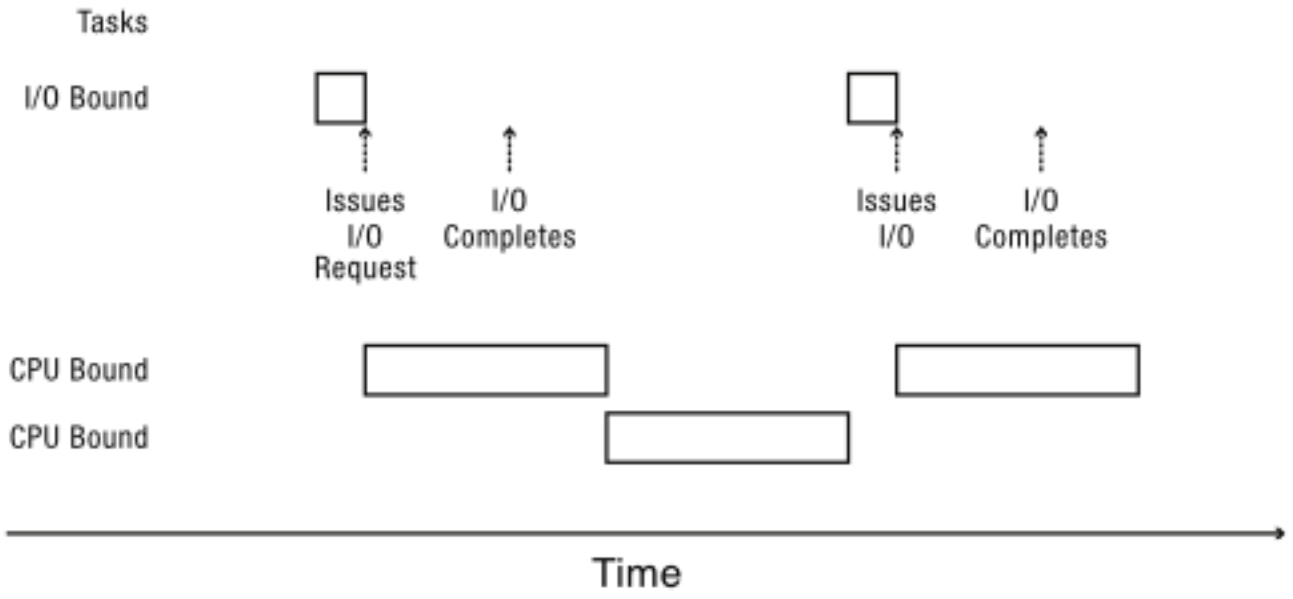
Observaciones

- ¿Cómo elegir el tiempo q?. Si $q \rightarrow 0$, mucho overhead por los constantes context switch. Si $q \rightarrow \infty$, se transforma en FIFO.
- Se puede ver RR como un compromiso entre SJF y FIFO. Si q es pequeño, las tareas terminan en el orden de su tiempo, similar a SJF, pero más lento (si hay n tareas, el factor será n).
- RR tiene algunas debilidades. Por ejemplo, consideremos una carga de trabajo con tareas que toman el mismo tiempo. Se puede ver que RR es la peor solución.

… Observaciones

- Dependiendo de q, RR puede ser malo cuando la carga de trabajo es una mezcla de tareas que requieren CPU y tareas que requieren E/S.
- Consideremos un editor que toma algunos mseg para mostrar los caracteres por pantalla. Si este editor lo compartimos con otras tareas utilizando RR, el editor necesitará esperar muchos tiempos q por cada tecla.
- El siguiente ejemplo muestra el comportamiento de RR al correr una mezcla de tareas CPU y E/S.

RR con tareas CPU y E/S



Tareas de E/S deben esperar su turno por CPU.
El resultado es un pobre desempeño.
Se podría achicar q, pero esto generará mucho overhead.

Equidad Max-Min

- En muchas configuraciones de schedulers, una asignación equitativa de recursos es tan importante como la capacidad de respuesta y el bajo overhead.
- Por ejemplo en un server no se puede permitir que un usuario pueda monopolizar todos los recursos de la máquina.
- Por simplicidad consideremos equidad sólo en procesos, pero los mismos principios se pueden utilizar al considerar usuarios, aplicaciones o threads.

… Equidad Max-Min

- Max-Min iterativamente maximiza la asignación mínima a un proceso particular (usuario, aplicación o thread) hasta que todos los recursos son asignados.

- Si todos los procesos son CPU-bound, se maximiza el mínimo entregando a cada proceso el mismo tiempo de procesador, o sea RR.

- Si hay procesos I/O-bound, se entrega todo lo que necesitan y el resto se distribuye entre los demás procesos. Algunos de los procesos que reciben una parte extra, no usarán todo y lo que sobra se redistribuye.

… Equidad Max-Min

- Consideremos una mezcla de I/O-bound y dos procesos CPU-bound:
  - El proceso I/O-bound necesita sólo un 10% del procesador para permanecer ocupado.
  - Con RR podrá obtener 0.5% del procesador y los procesos CPU-bound cerca de un 50%.
  - Con Max-Min podría obtener 10% y el resto 45% cada uno.

- Una forma de implementar es: cada vez que el scheduler necesita hacer una elección elige la tarea que tiene el mínimo tiempo acumulado de procesador.

5 Colas Multinivel Realimentadas (MFQ)

- La mayoría de los SO de propósito general actuales, utiliza MFQ (Multilevel Feedback Queue).

- Con MFQ se pueden lograr los siguientes objetivos simultáneamente:
  - Capacidad de respuesta: Correr tareas cortas en forma rápida (como SJF).
  - Bajo overhead: Minimizar el número de interrupciones para quitar CPU (como FIFO).
  - Libre de inanición: Todas las tareas progresan en el tiempo (como RR).

- MFQ es un buen compromiso entre objetivos distintos.

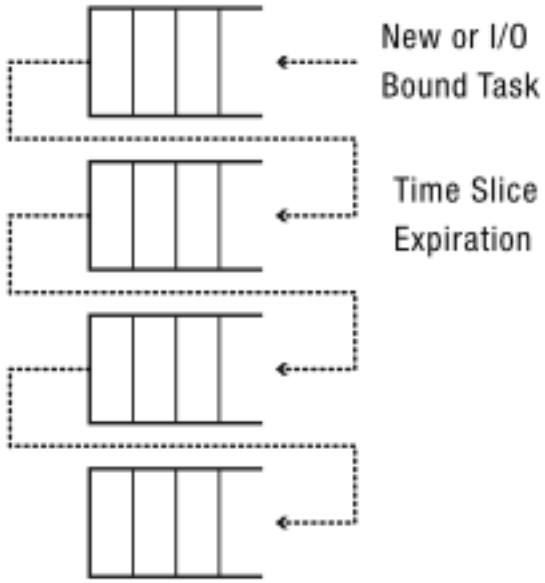
Algoritmo MFQ

- MFQ es una extensión de RR. En vez de una cola simple, MFQ tiene múltiples colas cada una con diferente nivel de prioridad y quantum de tiempo.
  1. Tareas de alta prioridad interrumpen tareas de baja prioridad. Las tareas del mismo nivel son itineradas utilizando RR.
  2. Tareas de mayor prioridad tienen q más pequeños.
  3. Las tareas van descendiendo de nivel en la medida que no alcanzan a terminar. Se privilegia tareas cortas.
  4. El último nivel puede ser FIFO o RR circular.
  5. Si en medio de un quantum, se genera I/O, la tarea vuelve al mismo nivel o vuelve al superior.

MFQ

Priority	Time Slice (ms)
1	10
2	20
3	40
4	80

Round Robin Queues



Observaciones

- El algoritmo MFQ no está libre de starvation y tampoco tiene equidad max-min.
- Si hay muchas tareas I/O-bound, tareas CPU-bound pueden no recibir procesador. Para solucionar esto, el scheduler debe monitorear cada proceso para asegurar que reciba una justa porción de recursos. Esto se puede hacer cambiando dinámicamente la prioridad.
- El itinerador de Linux es un buen ejemplo de MFQ.

6 Itineración de Multiprocesadores

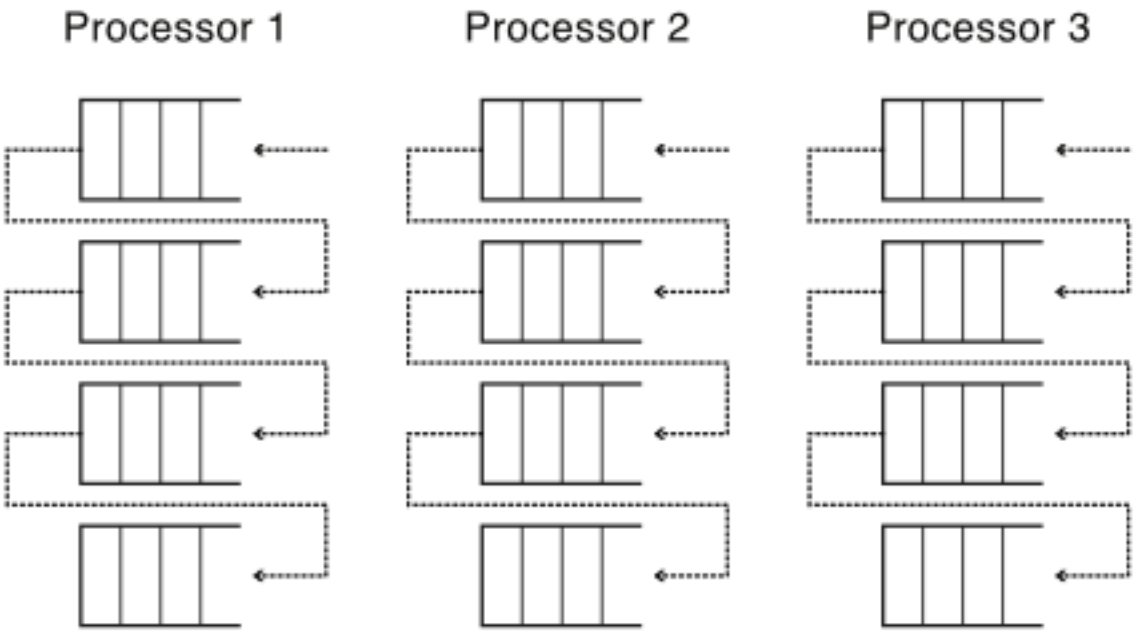
- Actualmente todos los computadores de propósito general son multiprocesadores.
- Desafíos: ¿Como utilizar multiprocesadores para correr tareas secuenciales?. ¿Cómo adaptar algoritmos de scheduling para aplicaciones paralelas?.

Ejemplo: Web Server

- Un server con un gran número de requerimientos:
  - Un thread separado por cada conexión.
  - Cada thread consulta una estructura compartida para ver si el documento solicitado está en la caché.
  - El thread recupera elementos que no estén en caché desde el disco.
  - El thread envía el resultado por la red.
  - ¿Cómo se debe itinerar este thread?
- Cada thread es I/O-bound. Para lograr buenos tiempos de respuesta hay que favorecer tareas cortas.
- Una posibilidad es manejar una cola MFQ centralizada, pero se generan problemas de contención (lock) y overhead de coherencia de cachés.

MFQ por procesador

SO comerciales utilizan una MFQ por procesador.



Afinidad de Procesador

- **Afinidad de procesador**: Una vez que un thread es itinerado en un procesador, cuando es re-itinerado vuelve al mismo procesador.

- La afinidad permite mejorar desempeño por aprovechamiento de las cachés.

- La afinidad de procesador puede involucrar rebalances cuando existe un desbalance persistente en los largos de cola. Si hay rebalance, las colas deben estar protegidas por locks.

Sistemas Operativos

Capítulo 7 Itineración

Prof. Javier Cañas R.