

Administración de Procesos

Explicación de práctica 4

Introducción a los Sistemas Operativos

Facultad de Informática
Universidad Nacional de La Plata

2018



- Programa en ejecución
- Los conceptos de tarea, job y proceso hacen referencia a lo mismo
- Según su historial de ejecución, los podemos clasificar en:
 - CPU Bound (ligados a la CPU)
 - I/O Bound (ligados a entrada/salida)

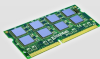


Definición Procesos (cont.)

- Programa
 - Es estático
 - No tiene program counter
 - Existe desde que se edita hasta que se borra



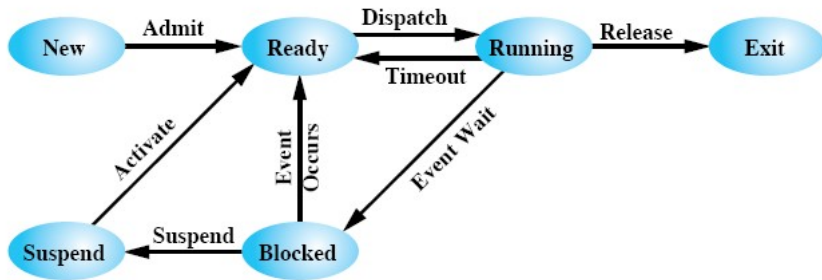
- Proceso
 - Es dinámico
 - Tiene program counter
 - Su ciclo de vida comprende desde que se lo ejecuta hasta que termina



- Una por proceso
- Contiene información del proceso
- Es lo primero que se crea cuando se realiza un *fork* y lo último que se desaloca cuando termina

| |
|------------------------|
| Identifier |
| State |
| Priority |
| Program counter |
| Memory pointers |
| Context data |
| I/O status information |
| Accounting information |
| • • • |



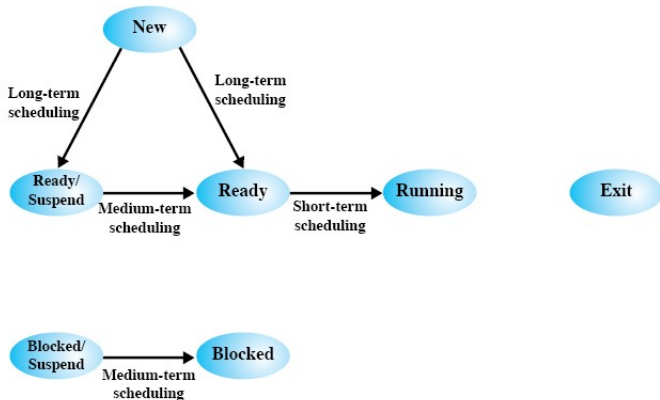


- Es la clave de la multiprogramación
- Esta diseñado de manera apropiada para cumplir las metas de:
 - Menor Tiempo de Respuesta
 - Mayor rendimiento
 - Uso eficiente del procesador

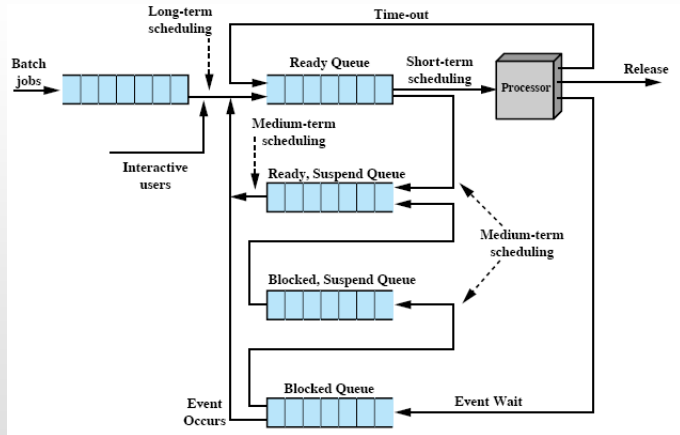


- **Long term scheduler:** admite nuevos procesos a memoria (controla el grado de multiprogramación)
- **Medium term scheduler:** realiza el *swapping* (intercambio) entre el disco y la memoria cuando el SO lo determina (puede disminuir el grado de multiprogramación)
- **Short term scheduler:** determina que proceso pasará a ejecutarse





Planificadores y Colas



- **Retorno:** tiempo que transcurre entre que el proceso llega al sistema hasta que completa su ejecución
- **Espera:** tiempo que el proceso se encuentra en el sistema esperando, es decir el tiempo que pasa sin ejecutarse (**TR - Tcpu**)
- **Promedios:** tiempos promedio de los anteriores



- **Nonpreemptive:** una vez que un proceso esta en estado de ejecución, continua hasta que termina o se bloquea por algún evento (e.j. I/O)
- **Preemptive:** el proceso en ejecución puede ser interrumpido y llevado a la cola de listos:
 - Mayor overhead pero mejor servicio
 - Un proceso no monopoliza el procesador



- First come first served
- Cuando hay que elegir un proceso para ejecutar, se selecciona el mas viejo
- No favorece a ningún tipo de procesos, pero en principio prodíamos decir que los *CPU Bound* terminan al comenzar su primer ráfaga, mientras que los *I/O Bound* no



Algoritmo FIFO (cont.)

| Job | Llegada | CPU | Prioridad |
|-----|---------|-----|-----------|
| 1 | 0 | 9 | 3 |
| 2 | 1 | 5 | 2 |
| 3 | 2 | 3 | 1 |
| 4 | 3 | 7 | 2 |

#Ejemplo 1

TAREA '1' PRIORIDAD=3 INICIO=0
[CPU, 9]

TAREA '2' PRIORIDAD=2 INICIO=1
[CPU, 5]

TAREA '3' PRIORIDAD=1 INICIO=2
[CPU, 3]

TAREA '4' PRIORIDAD=2 INICIO=3
[CPU, 7]

¿Cuáles serían los tiempos de retorno y espera?



- Shortest Job First
- Política *nonpreemptive* que selecciona el proceso con la ráfaga más corto
- Calculo basado en la ejecución previa
- Procesos cortos se colocan delante de procesos largos
- Los procesos largos pueden sufrir *starvation* (inanición)
- Veamos el ejemplo anterior



- Shortest Job First
- Política *nonpreemptive* que selecciona el proceso con la ráfaga más corto
- Calculo basado en la ejecución previa
- Procesos cortos se colocan delante de procesos largos
- Los procesos largos pueden sufrir *starvation* (inanición)
- Veamos el ejemplo anterior



- Round Robin
- Política basada en un reloj
- **Quantum (Q)**: medida que determina cuanto tiempo podrá usar el procesador cada proceso:
 - Pequeño: overhead de *context switch*
 - Grande: ¿pensar?
- Cuando un proceso es expulsado de la *CPU* es colocado al final de la *Ready Queue* y se selecciona otro (*FIFO circular*)



- Existe un “contador” que indica las unidades de CPU en las que el proceso se ejecuto. Cuando el mismo llega a 0 el proceso es expulsado
- El “contador” puede ser:
 - Global
 - Local \rightarrow PCB
- Existen dos variantes con respecto al valor inicial del “contador” cuando un proceso es asignado a la CPU:
 - **Timer Variable**
 - **Timer Fijo**



Algoritmo RR - Timer Variable

- El “contador” se inicializa en Q ($\text{contador} := Q$) cada vez que un proceso es asignado a la *CPU*
- Es el más utilizado
- Utilizado por el simulador
- Veamos el ejemplo 1 nuevmanete



Algoritmo RR - Timer Variable

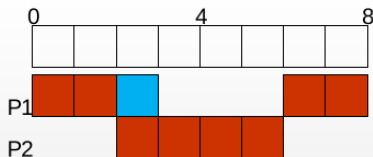
- El “contador” se inicializa en Q ($\text{contador} := Q$) cada vez que un proceso es asignado a la *CPU*
- Es el más utilizado
- Utilizado por el simulador
- Veamos el ejemplo 1 nuevmanete




- El “contador” se inicializa en Q cuando su valor es cero
 - if (contador == 0) contador = Q;
- Se puede ver como un valor de Q compartido entre los procesos




Timer Variable

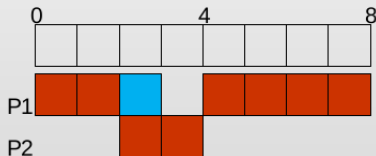


Round Robin, Q=4

 = E/S

 = Uso de CPU

Timer Fijo



Algoritmo con Uso de Prioridades

- Cada proceso tiene un valor que representa su prioridad → menor valor, mayor prioridad
- Se selecciona el proceso de mayor prioridad de los que se encuentran en la *Ready Queue*
- Existe una *Ready Queue* por cada nivel de prioridad
- Procesos de baja prioridad pueden sufrir *starvation* (inanición)
 - Solución: permitir a un proceso cambiar su prioridad durante su ciclo de vida → **Aging o Penalty**
- Puede ser un algoritmo **preemptive** o no
- Veamos el ejemplo 1 nuevamente

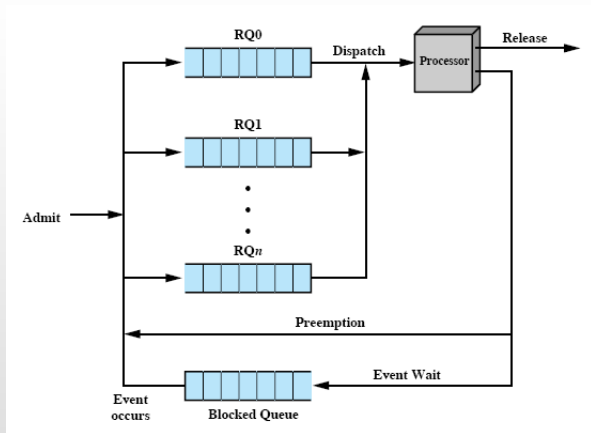


Algoritmo con Uso de Prioridades

- Cada proceso tiene un valor que representa su prioridad → menor valor, mayor prioridad
- Se selecciona el proceso de mayor prioridad de los que se encuentran en la *Ready Queue*
- Existe una *Ready Queue* por cada nivel de prioridad
- Procesos de baja prioridad pueden sufrir *starvation* (inanición)
 - Solución: permitir a un proceso cambiar su prioridad durante su ciclo de vida → **Aging o Penalty**
- Puede ser un algoritmo **preemptive** o no
- Veamos el ejemplo 1 nuevamente



Algoritmo con Uso de Prioridades (cont.)



- Shortest Remaining Time First
- Versión *preemptive* de *SJF*
- Selecciona el proceso al cual le resta menos tiempo de ejecución en su siguiente ráfaga.
- ¿A qué tipos de procesos favorece? → I/O Bound
- Veamos el ejemplo 1 nuevamente



- Shortest Remaining Time First
- Versión *preemptive* de *SJF*
- Selecciona el proceso al cual le resta menos tiempo de ejecución en su siguiente ráfaga.
- ¿A qué tipos de procesos favorece? → **I/O Bound**
- Veamos el ejemplo 1 nuevamente



- Shortest Remaining Time First
- Versión *preemptive* de *SJF*
- Selecciona el proceso al cual le resta menos tiempo de ejecución en su siguiente ráfaga.
- ¿A qué tipos de procesos favorece? → **I/O Bound**
- Veamos el ejemplo 1 nuevamente



- Ciclo de vida de un proceso: uso de CPU + operaciones de I/O
- Cada dispositivo tiene su cola de procesos en espera → un scheduler por cada cola
- Se considera I/O independiente de la CPU (DMA, PCI, etc.)
→ uso de CPU y operaciones de I/O en simultaneo



- Orden de aplicación:
 - Orden de llegada de los procesos
 - **PID** de los procesos
- Siempre se mantiene la misma política



Algoritmos de planificación - Un recurso por proceso

| Job | Llegada | CPU | E/S (rec., inst., dur.) |
|-----|---------|-----|-------------------------|
| 1 | 0 | 5 | (R1, 3, 2) |
| 2 | 1 | 4 | (R2, 2, 2) |
| 3 | 2 | 3 | (R3, 2, 3) |

#Ejemplo 2

RECURSO 'R1'

RECURSO 'R2'

RECURSO 'R3'

TAREA '1' INICIO=0

[CPU, 3] [1, 2] [CPU, 2]

TAREA '2' INICIO=1

[CPU, 2] [2, 2] [CPU, 2]

TAREA '3' INICIO=2

[CPU, 2] [3, 3] [CPU, 1]



| Job | Llegada | CPU | E/S (rec., inst., dur.) |
|-----|---------|-----|-------------------------|
| 1 | 0 | 5 | (R1, 3, 3) |
| 2 | 1 | 4 | (R1, 1, 2) |
| 3 | 2 | 3 | (R2, 2, 3) |

#Ejemplo 3

RECURSO 'R1'

RECURSO 'R2'

TAREA '1' INICIO=0

[CPU, 3] [1, 3] [CPU, 2]

TAREA '2' INICIO=1

[CPU, 1] [1, 2] [CPU, 3]

TAREA '3' INICIO=2

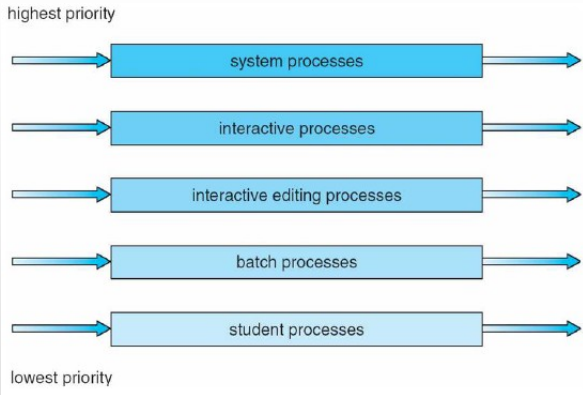
[CPU, 2] [2, 3] [CPU, 1]



- Schedulers actuales → combinación de algoritmos vistos
- La *ready queue* es dividida en varias colas (similar a prioridades)
- Los procesos se colocan en las colas según una clasificación que realice el sistema operativo
- Cada cola posee su propio algoritmo de planificación → **planificador horizontal**
- A su vez existe un algoritmo que planifica las colas → **planificador vertical**
- Retroalimentación → un proceso puede cambiar de una cola a la otra



Esquema Colas Multinivel (ejemplo 1)

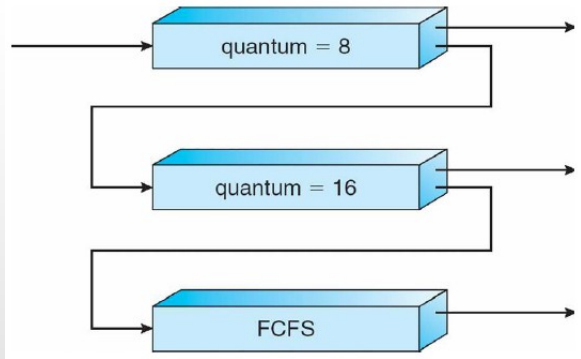


Esquema Colas Multinivel (ejemplo 2)

- El sistema consta de tres colas:
 - Q0: se planifica con *RR*, $q=8$
 - Q1: se planifica con *RR*, $q=16$
 - Q2: se planifica con *FCFS*
- Para la planificacion se utilizan los siguientes criterios:
 - Los procesos ingresan en la Q0. Si no se utilizan los 8 cuantos, el job es movido a la cola Q1
 - Para la cola Q1, el comportamiento es similar a Q0. Si un proceso no finaliza su ráfaga de 16 instantes, es movido a la cola Q2



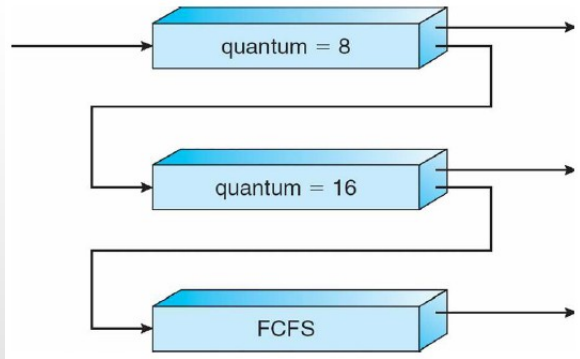
Esquema Colas Multinivel (ejemplo 2 cont.)



- ¿A qué procesos beneficia el algoritmo? → **CPU Bound**
- ¿Puede ocurrir inanición? → Si, con los procesos ligados a E/S si siempre llegan procesos ligados a CPU



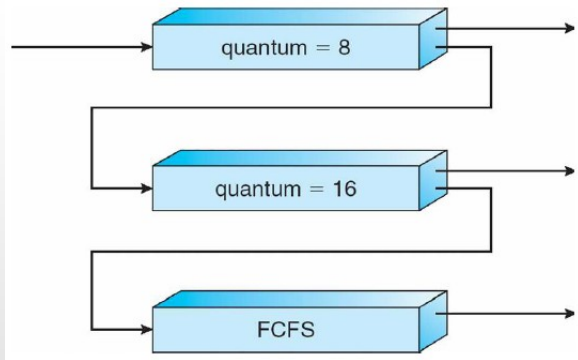
Esquema Colas Multinivel (ejemplo 2 cont.)



- ¿A qué procesos beneficia el algoritmo? → CPU Bound
- ¿Puede ocurrir inanición? → Si, con los procesos ligados a E/S si siempre llegan procesos ligados a CPU



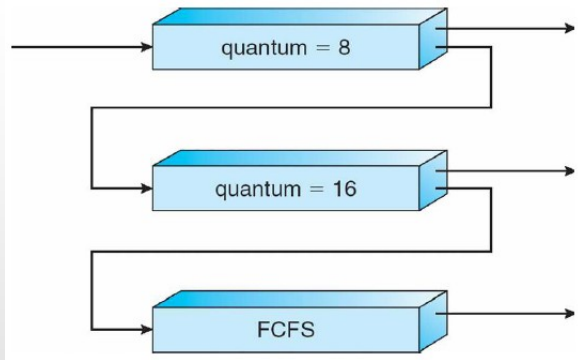
Esquema Colas Multinivel (ejemplo 2 cont.)



- ¿A qué procesos beneficia el algoritmo? → **CPU Bound**
- ¿Puede ocurrir inanición? → Si, con los procesos ligados a E/S si siempre llegan procesos ligados a CPU



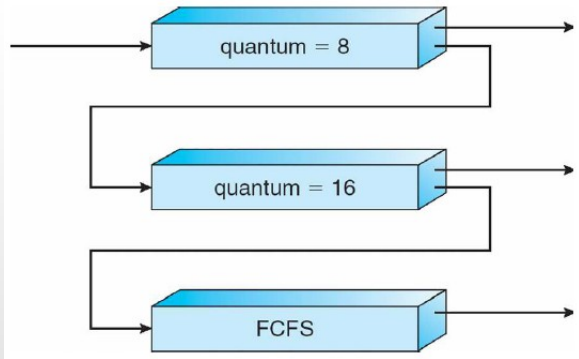
Esquema Colas Multinivel (ejemplo 2 cont.)



- ¿A qué procesos beneficia el algoritmo? → **CPU Bound**
- ¿Puede ocurrir inanición? → Si, con los procesos ligados a E/S si siempre llegan procesos ligados a CPU



Esquema Colas Multinivel (ejemplo 2 cont.)



- ¿A qué procesos beneficia el algoritmo? → **CPU Bound**
- ¿Puede ocurrir inanición? → Si, con los procesos ligados a E/S si siempre llegan procesos ligados a CPU



- La planificación de CPU es más compleja cuando hay múltiples CPUs
- Este enfoque fue implementado inicialmente en *Mainframes* y luego en *PC*
- La carga se divide entre distintas CPUs, logrando capacidades de procesamiento mayores
- Si un procesador falla, el resto toma el control



Planificación con múltiples procesadores: Criterios

- **Planificación temporal** → que proceso y durante cuanto
- **Planificación espacial** → en que procesador ejecutar:
 - **Huella:** estado que el proceso va dejando en la cache de un procesador
 - **Afinidad:** preferencia de un proceso para ejecutar en un procesador
- La asignación de procesos a un procesador puede ser:
 - **Estática:** existe una afinidad de un proceso a una CPU
 - **Dinámica:** la carga se comparte → balanceo de carga
- La política puede ser:
 - **Tiempo compartido:** se puede considerar una *cola global* o una *cola local* a cada procesador
 - **Espacio compartido:**
 - Grupos (*threads*)
 - Particiones



Planificación con múltiples procesadores (cont.)

- *Clasificaciones:*
 - **Procesadores homogéneos:** todas las CPUs son iguales. No existen ventajas físicas sobre el resto
 - **Procesadores heterogéneos:** cada procesador tiene su propia cola, su propio clock y su propio algoritmo de planificación
- *Otra clasificación:*
 - **Procesadores débilmente acoplados:** cada CPU tiene su propia memoria principal y canales
 - **Procesadores fuertemente acoplados:** comparten memoria y canales
 - **Procesadores especializados:** uno o más procesadores principales de uso general y uno o más procesadores de uso específico



¿Preguntas?

