Sistemas Operativos avanzados

Apuntes: Clase del 6 de Marzo de 2017

José Daniel Salazar Vargas
200611533

Tabla de Contenidos

Administrativo

Se hace la revisión presencial del proyecto 1.
También se especifican los siguientes papers para los resúmenes semanales:

- 302
- 305
- 306
- 307
- 609

Scheduling (Continuación)

PCB

- Process Control Block.
- Estructura de datos con que el *SO* representa a los procesos.
- Todas las manipulaciones que hacen los schedulers son sobre los PCB.

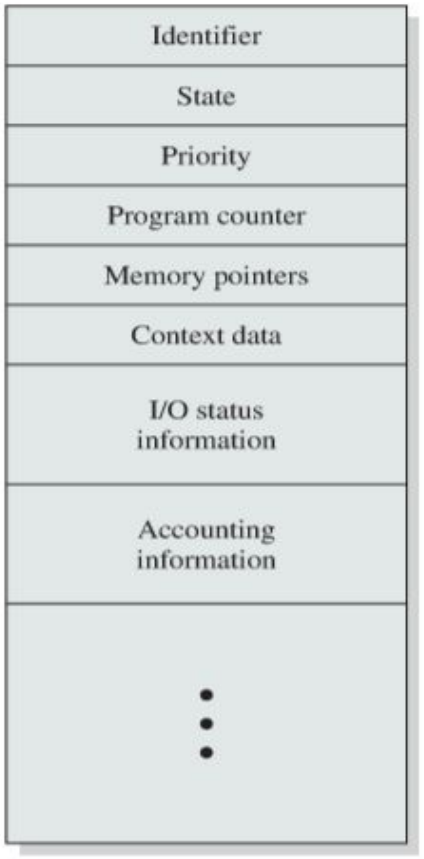
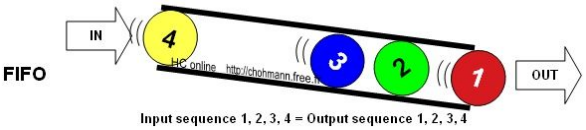


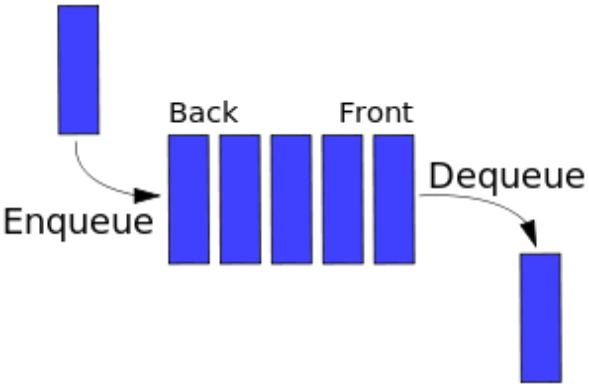
Figure 3.1 Simplified Process Control Blo

Algoritmos clásicos para administrar PCBs en Ready y asignar CPU

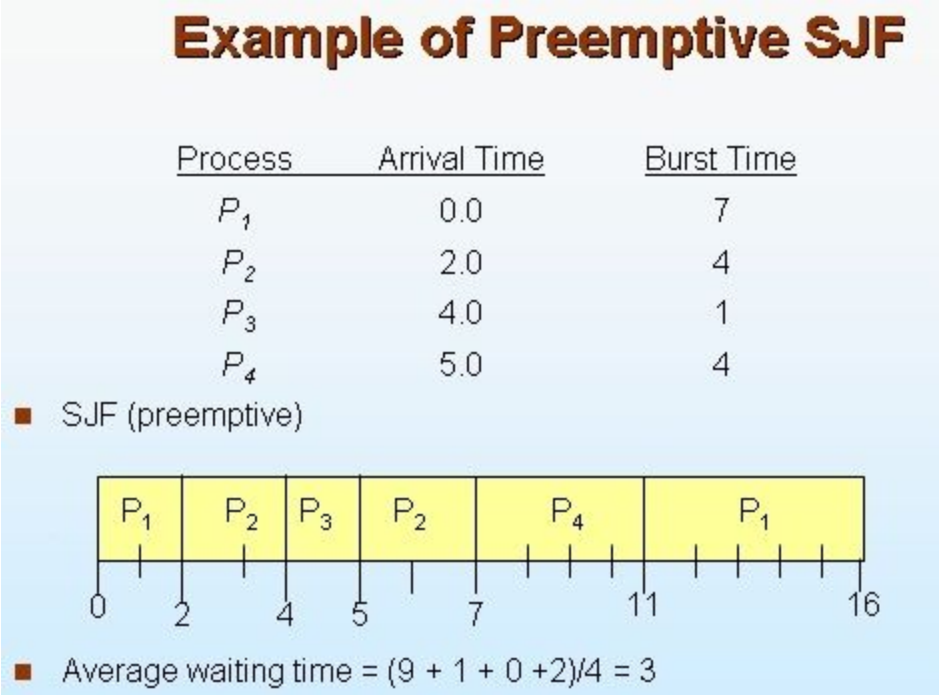
FIFO

1. First In, First Out
2. Considerado el peor algoritmo de scheduling que existe.
3. Ventajas:
    a. Simple.
    b. Sencillo de implementar.
    c. Eficiente.
    d. Ineherentemente justo.
4. Desventajas:
    a. Procesos de alta prioridad tienen que esperar mucho.
    b. No considera los recursos que cada proceso necesita.
    c. Los procesos corren hasta finalizar, por lo que otros procesos deben esperar a que el procesos actual termine.





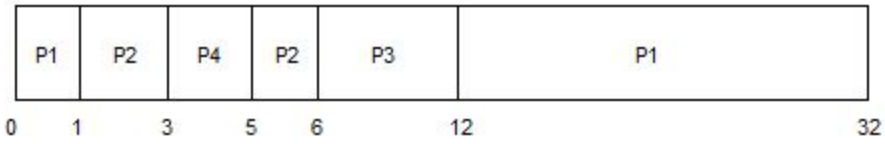
SJF



1. Shortest Job First
2. Ventajas:
    a. Los procesos grandes pueden esperar a que los pequeños corran primero.
    b. Minimiza el tiempo promedio de corrida de los procesos.
3. Desventajas:
    a. Procesos muy grandes tardarían mucho en procesarse.
    b. Se necesita saber de antemano el tiempo de ejecución (o CPU burst) de los procesos, lo cual es imposible en muchos ambientes.

PROCESS	BURST TIME	ARRIVAL TIME
P1	21	0
P2	3	1
P3	6	2
P4	2	3

The GANTT chart for Preemptive Shortest Job First Scheduling will be,



The average waiting time will be, ((5-3) + (6-2) + (12-1))/4 = 4.25 ms

The average waiting time for preemptive shortest job first scheduling is less than both, non-preemptive SJF scheduling and FCFS scheduling.

4. Promedio Exponencial: Sirve para estimar (o predecir) el CPU burst de los procesos, de tal manera que se puedan asignar prioridades inversas al CPU burst:

$$T_i = \alpha \cdot t_{i-1} + (1 - \alpha) \cdot T_{i-1}$$

   a. Con $0 \leq \alpha \leq 1$.

   b. Donde T_{i-1} es la estimación anterior y representa la historia del proceso.

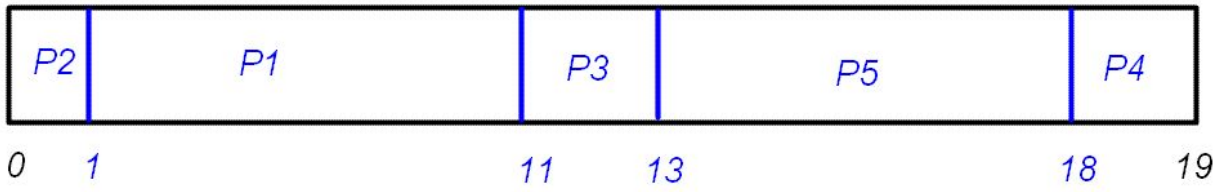
   c. Y t_{i-1} es el tiempo real de la corrida anterior.

   d. T_0 es un promedio del sistema.

HPF

	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	3

Given these processes, draw a Gantt Chart showing the execution.



1. Highest Priority First
2. La prioridad de los procesos determina el orden en que se les asigna el CPU.
3. Puede ser **expropiativo** (el proceso nuevo corre inmediatamente) o **no expropiativo** (el proceso nuevo es el siguiente en la cola a la espera de que el proceso actual termine).
4. Ventajas:
    a. Nos permite mapear la importancia de los procesos.
    b. Relativamente fácil de implementar.
5. Desventajas:
    a. Los procesos de baja prioridad van a esperar mucho tiempo para ser ejecutados (**starvation**).

PROCESS	BURST TIME	PRIORITY
P1	21	2
P2	3	1
P3	6	4
P4	2	3

The GANTT chart for following processes based on Priority scheduling will be,

The average waiting time will be, (0 + 3 + 24 + 26)/4 = 13.25 ms

6. **Aging**: Prioridades Dinámicas:
    a. Ir subiendo la prioridad a los procesos con baja prioridad si llevan mucho tiempo esperando.
    b. Sirve para combatir **starvation**.
    c. α, β : Grado en que se incrementa la prioridad:
        i. $\alpha \rightarrow Ready$.
        ii. $\beta \rightarrow Run$.
    d. Ejemplos:
        i. $0 \leq \beta \leq \alpha$
        ii. $\beta < 0 < \alpha$

Lottery Scheduling

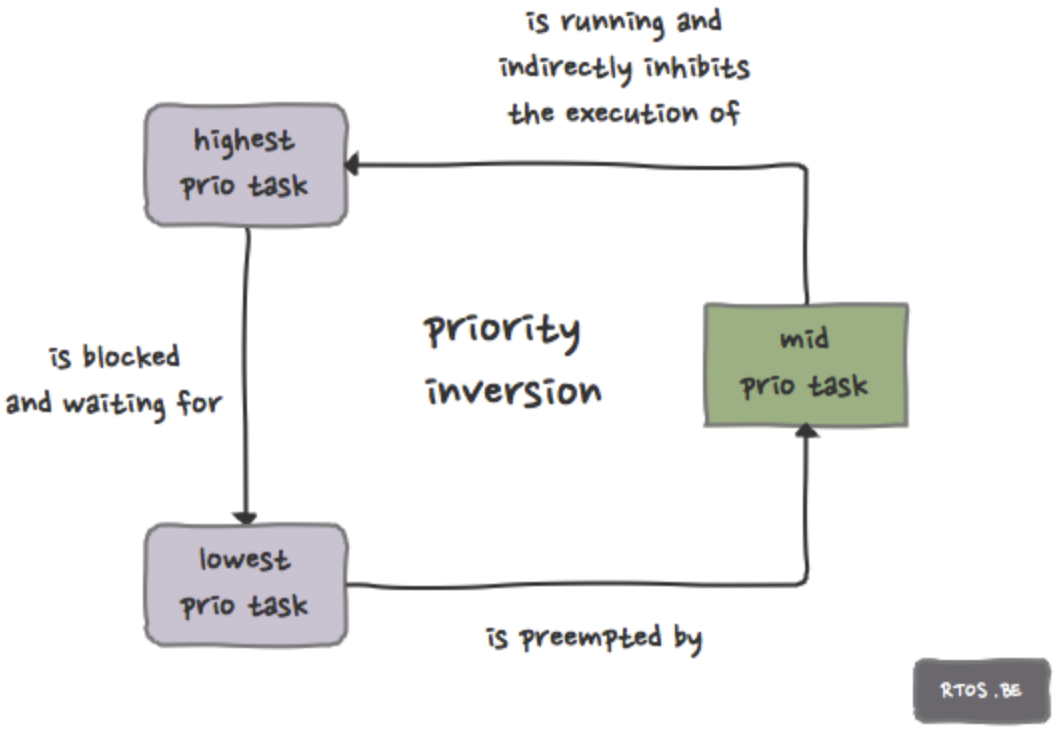
1. Es un tipo de HPF.
2. Se asignan “tiquetes de lotería” a los procesos según sus prioridades.
3. Los procesos con prioridad alta reciben más “tiquetes”, los procesos con prioridad baja reciben menos “tiquetes”.
4. Una vez asignados los tiquetes, aleatoriamente se elige el “tiquete ganador”, el proceso que tenga asignado dicho “tiquete” será el siguiente en usar la CPU.
5. Es fácil de implementar.
6. Funciona muy bien!
7. Menos propenso a **starvation**.

Inversión de Prioridades

1. Es un problema que se presenta cuando el orden es determinado por las prioridades, típicamente HPF.
2. Sucede cuando un proceso de prioridad media/baja corre primero (como si tuviera una prioridad mayor) que un proceso de prioridad alta.
3. Este caso no se presenta con frecuencia, sin embargo cuando se presenta (dadas las circunstancias necesarias) puede ocasionar un funcionamiento no deseado del sistema.
4. Se presenta cuando un proceso de prioridad baja B bloquea un recurso compartido R que necesita para correr. Mientras B corre, un nuevo proceso de alta prioridad A requiere el recurso R que actualmente está bloqueado por B, por lo tanto A no está listo para correr y es bloqueado. Un nuevo proceso de prioridad media M desplaza

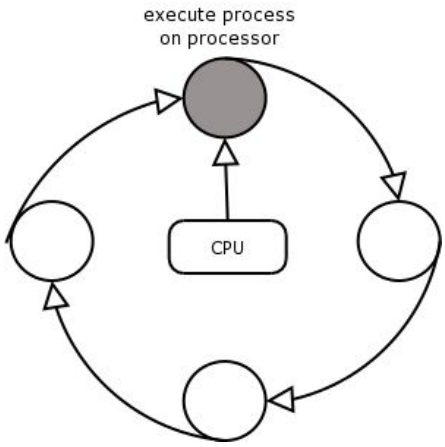
a B (debido a que tiene mayor prioridad) y empieza a correr.
El problema se presenta porque mientras se ejecute M (y/o cualquier otro proceso de prioridad media/alta) B no va a poder terminar de correr, y mientras B no termine de correr el recurso R no va a ser liberado, por lo tanto A no podrá correr. Y de esta manera es que procesos con prioridad media correrán con mayor “prioridad” que A, esto es inversión de prioridades.

5. Este problema se combate con **Herencia de Prioridades**: Los procesos de menor prioridad “heredan” o adquieren la prioridad del proceso que está esperando el recurso compartido que está siendo utilizado por el proceso de baja prioridad.
En el ejemplo del punto 4, el proceso B heredaría la prioridad del proceso A, de esta manera la nueva prioridad de B le permite terminar de correr antes que cualquier otro proceso con prioridad menor a A. Lo que se logra con esto es que B logre terminar de manera rápida para que así el proceso A de alta prioridad pueda correr.



Round Robin

1. En principio es FIFO, con un *quantum* o *time slice.*
2. Si el *quantum* es muy grande se comporta como FIFO (FIFO es un Round Robin con un *quantum* infinito).
3. Si el *quantum* es muy pequeño se desperdicia casi todo el tiempo en cambios de contexto.
4. El *quantum* debe ser mayor al tiempo que toma el cambio de contexto.



Process	Burst Time
P1	12
P2	8
P3	4
P4	10
P5	5

Round Robin scheduling algorithm
Gantt chart

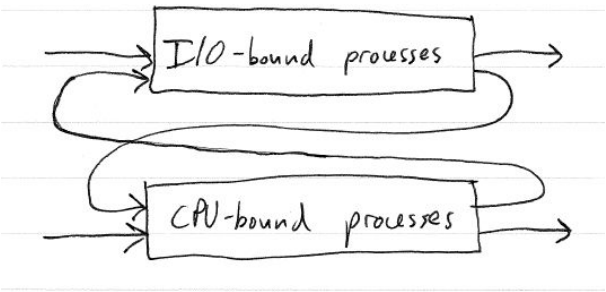
P1	P2	P3	P4	P5	P1	P2	P4	P1

0 5 10 14 19 24 29 32 37 39

Colas Retroalimentadas

1. Es un sistema de colas donde cada una administra los PCBs de manera independiente (FIFO, SJF, HPF, RR, etc.), sin embargo los PCBs pueden ser movidos de una cola a otra dependiendo de su comportamiento dinámico.

2. Podría existir una prioridad entre colas (colas con procesos de alta prioridad, colas con procesos de baja prioridad).



3. El movimiento de PCBs se puede dar según el tipo de colas, por ejemplo:

    a. Si un procesos ha utilizado (en sus *time slice* anteriores) mucho I/O como teclado o mouse puede ser interpretado como un proceso interactivo, y moverse a la cola respectiva.

    b. Otro proceso ha utilizado mucho CPU durante sus *quantum* anteriores, entonces es movido a la cola de procesos que utilizan mucho CPU.

    c. Los procesos pueden tener distintos comportamientos en su vida, pueden ser de alto consumo de CPU al inicio pero luego pueden utilizar mucho I/O, así que un mismo proceso puede ser movido entre colas dependiendo de su comportamiento dinámico.