Sistemas Operativos
2° Parcial 2C2017 - TT - Resolución

Aclaración: La mayoría de las preguntas o ejercicios no tienen una única solución. Por lo tanto, si una solución particular no es similar a la expuesta aquí, no significa necesariamente que la misma sea incorrecta. Ante cualquier duda, consultar con el profesor del curso.

Teoría

1.

	Tamaño	Accesos Memoria	Accesos disco
Un nivel	1 entrada por página de proceso	2 accesos	2
Multinivel	Mucho menor que la de un nivel	Cantidad de niveles + 1	2
Invertida	1 entrada por frame (menor que un nivel y multinivel)	Idealmente uno (depende de la función de hash)	No es posible implementar MV de una manera sencilla.

2.
   a. Verdadero, al ejecutar de a uno los procesos (hasta que no termina no ejecuta otro, por más que este haga una E/S) no produce retención y espera. Caso especial: considerar entornos con hilos.
   b. Falso. Si pensamos como "justicia" que se respete el orden de los pedidos, podríamos considerar que es más justo porque va atendiendo pedidos a medida que barre el disco, y tiene menos chances de inanición.
3. Dado que, de no poder ser realizadas inmediatamente, devuelven un error, el programador debe ocuparse de realizarlas nuevamente cuando corresponda, o tomar otra decisión al respecto.
4. Principio por el cual durante un intervalo de tiempo, las referencias se circunscriben a un subconjunto reducido de páginas. El algoritmo clock modificado, al intentar no reemplazar aquellas páginas que fueron usadas recientemente, promueve la continuidad del principio de localidad.
5.
   a. Mutua exclusión: no garantizarla sobre los recursos. No es recomendable.
   b. Retención y espera: solicitar todos los recursos juntos.
   c. No expropiación: expropiar recursos si es necesario. No es algo sencillo, pero es factible.
   d. Espera circular: forzar una solicitud ordenada de recursos

Práctica

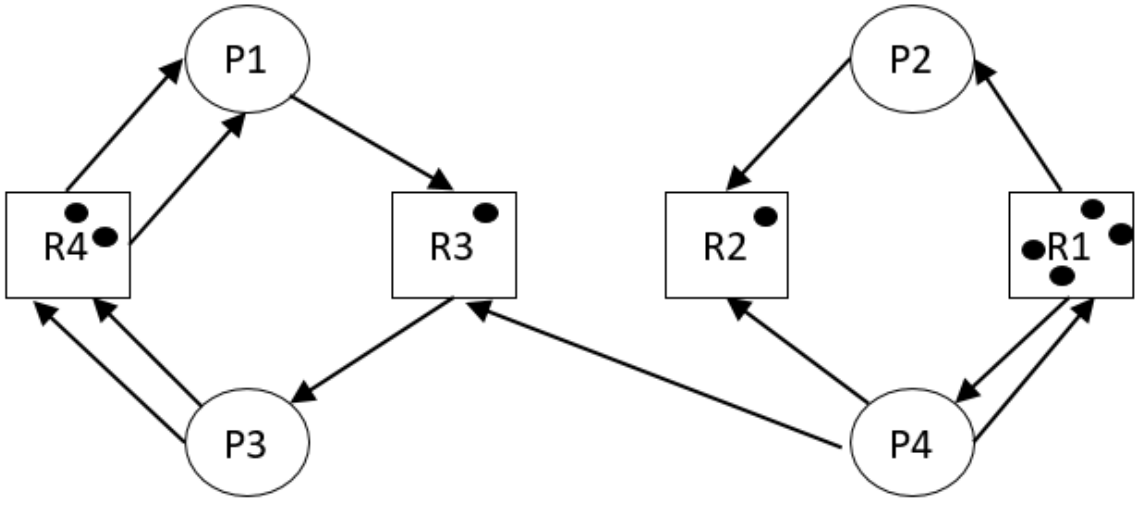
Ejercicio 1

a)
Recursos Disponibles (2,1,0,0)
Inicia el Algoritmo de Detección:
Ejecuto Proceso 2 y libera sus recursos --> Disponibles (3,1,0,0)
Deadlock entre Procesos 1, 3 y 4.
[Si se elimina el Proceso 4, no se soluciona el Deadlock. Debe eliminarse el Proceso 1 ó 3.]

*Opción 1:
Elimino Proceso 1 y se liberan sus recursos --> Disponibles (3,1,0,2)
Ejecuto Proceso 3 y libera sus recursos --> Disponibles (3,1,1,2)
Ejecuto Proceso 4 y libera sus recursos --> Disponibles (4,1,1,2)

*Opción 2:
Elimino Proceso 3 y se liberan sus recursos --> Disponibles (3,1,1,0)
Ejecuto Proceso 1 y libera sus recursos --> Disponibles (3,1,1,2)
Ejecuto Proceso 4 y libera sus recursos --> Disponibles (4,1,1,2)

b)



Según el punto a) los procesos 1, 3 y 4 están en deadlock. Pero en el grafo puede visualizarse que el proceso 4 no participa de ningún ciclo con otro proceso, por lo tanto, no forma parte del Deadlock. Los procesos en deadlock son los procesos 1 y 3.

c) Agregando una instancia de R3, no habría deadlock. Si se agregaran instancias de los Recursos 1, 2 y 4, el deadlock se producirá igual.

Ejercicio 2

a) La tasa de utilización de CPU es baja y la de disco es muy alta. Claramente estamos en presencia de thrashing. Aumentar el grado de multiprogramación empeoraría las cosas, porque ocasionaría más fallos de página.

Ej: 4 procesos (A, B, C, D), compartiendo 4 frames, sería: A0, B0, C0, D0, A1, B1, C1, D1, A2, B2, etc. Si la localidad de cada proceso no se puede cargar en memoria (por ejemplo, cada proceso necesita 3 páginas), esto ocasiona thrashing.

b) La tasa de utilización de CPU es alta y la de disco es muy baja. A simple vista podemos pensar que los procesos tienen su localidad en memoria principal. Aumentar el grado de multiprogramación podría aumentar la tasa de utilización de la CPU, siempre y cuando los procesos nuevos no generen thrashing.

Ej: 4 procesos (A, B, C, D), compartiendo 4 frames, sería: A0, B0, C0, D0, A0, B0, C0, D0, A0, B0, etc. Si la localidad de cada proceso se puede cargar rápidamente en memoria, tendremos una tasa de utilización de disco baja y mucho uso de CPU.

c) La tasa de utilización de CPU es baja y la de disco es muy baja. A simple vista podemos pensar que hay pocos procesos y tienen su localidad en memoria principal. Aumentar el grado de multiprogramación podría aumentar la tasa de utilización de la CPU, y estamos bastante lejos de generar thrashing (a diferencia del ejemplo anterior)

Ej: 1 proceso (A), usando 4 frames, sería: A0, A0, A0, A0, A0, A0, A0, A0, A0, A0, etc. La localidad de este proceso se puede cargar rápidamente en memoria, por lo que tendremos una tasa de utilización de disco baja y poco uso de CPU.

Ejercicio 3

a)
FSCAN
Orden de atención: 5 => 0 => 9 => 48 => 50 => 56 => 57 => 69 => 99 => 15

SCAN
Orden de atención: 5 => 0 => 9 => 48 => 50 => 56 => 57 => 69 => 99 => 15

b) Debería producirse, por ejemplo, el pedido 48 reiteradas veces.