Sistemas Operativos
2° Rec 1er Parcial 2C2023 - 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/la docente del curso.

Teoría

1. a) F, si la interrupción se produjo durante la atención de otra interrupción, solamente es requerido un cambio de contexto y no de modo, ya que ambas se realizan en modo kernel, por ser el sistema operativo quien debe atenderlas.
   b) F, la ejecución de una syscall implica además un cambio de contexto, ya que se le solicita al sistema operativo que la ejecute por lo cual debe de hacerse un cambio de modo y de contexto para su correspondiente procesamiento, y al finalizar se ha de devolver el contexto al proceso y el modo debe de ser cambiado a modo usuario.

2. Los algoritmos sin desalojo son más propensos a monopolizar la CPU, por lo cual podrían llegar a generar que otros procesos entren en starvation, otro ejemplo podría ser utilizando un algoritmo como SJF sin desalojo donde siempre entren procesos más chicos que un proceso o grupo de procesos, produciendo que estos entren en starvation.

3. a) V, si el sistema operativo utiliza un algoritmo de planificación como RR o si se produjera una interrupción durante la ejecución de un ULT que provoque desalojar al proceso de la CPU, el estado del mismo pasaría de Running a Ready, pero el ULT dado que no es conocido por el sistema operativo, no tiene manera de actualizar su estado, quedando en Running a pesar de que su proceso se encuentre en Ready.
   b) V, ya que si los ULTs implementan en su biblioteca un algoritmo de planificación sin desalojo, los únicos momentos donde podría darse un thread switch son cuando finaliza uno de los ULTs permitiendo que continúe utilizando la CPU otro del mismo proceso o bien, cuando se utilizan wrappers para replanificar tras una IO, dado que esta última no sucede por ejecutar directamente las syscalls, sólo nos queda la primer opción.

4. Es necesario protegerlos debido a que en caso de no hacerlo podría darse una condición de carrera, produciendo que el recurso pueda quedar inconsistente, devolviendo valores no esperados, para ello se pueden utilizar semáforos tipo mutex para evitar que varios hilos accedan a estos recursos compartidos de manera “controlada” evitando la condición de carrera. Un ejemplo podría ser un contador manipulado por varios hilos, donde al incrementar la variable global, se interrumpa el uno al otro, produciendo valores inconsistentes.

5. Las estrategias válidas son aquellas donde NO se produzca bajo ninguna circunstancia deadlock, ya que en este caso podría ser muy perjudicial la existencia de uno, por lo que las estrategias de Detección y Recupero, y la estrategia de no tratarlo no son de utilidad en este sistema, quedando como posibles soluciones la de Prevención y la de Evasión; en caso de que el sistema no sea tolerante la overhead, se ha de descartar la estrategia de evasión, dado que se realizan constantes verificaciones ante cada solicitud de recursos, mientras que en prevención, son "reglas predefinidas" por lo cual se reduce drásticamente el overhead y asegura la no existencia de deadlock.

Práctica

1. a) Elimino del análisis a P3 por no tener recursos asignados.

Recursos Disponibles: 0 1 0 1
Con estos recursos se puede finalizar P2 -> Solicitud: 0 1 0 1 Asignados: 0 2 2 1
Actualizo Disponibles: 0 3 2 2
No es posible finalizar ninguno de los demás procesos:
P1 -> Solicitud: 1 3 3 2 P4 -> Solicitud: 1 3 5 2 P5 -> Solicitud: 3 0 3 0 ,
Por lo tanto P1, P4 y P5 se encuentran en Deadlock y P3 en inanición.

Debo eliminar un proceso para salir del Deadlock, seleccionando aquel proceso con mayor cantidad de recursos asignados:
P1 -> Asignados: 2 1 2 1 = 6
P4 -> Asignados: 3 0 2 1 = 6
P5 -> Asignados: 1 2 0 2 = 5

Si finalizo P1:
Recursos Disponibles: 0 3 2 2
Actualizo Disponibles: 2 4 4 3
No puedo finalizar ni P4 (falta 1 R3) ni P5 (falta 1 R1)

Si finalizo P4:
Recursos Disponibles: 0 3 2 2
Actualizo Disponibles: 3 3 4 3

Finaliza P5 -> Recursos Disponibles: 4 5 4 5
Finaliza P1 -> Recursos Disponibles: 6 6 6 6 -> Recursos Totales

2. En el instante 6 y 9 se puede apreciar que ocurrió un desalojo quedando como opciones posibles los siguientes algoritmos:

-SJF c/desalojo: descartado debido a que en T=3 llega PC con menor ráfaga que la restante de PD y este no es desalojado.
-RR: descartado ya que en T=5 PB ya se encontraba en la cola de ready pero PA es quien ejecuta su rafaga de CPU.
-Algún otro algoritmo por prioridades: el enunciado no proporciona ningún tipo de prioridad de un proceso sobre otro por lo cual se descarta teniendo esto como criterio.
-VRR: los dos instantes donde se producen desalojos de CPU, son los T=6 y T=9, y ambos encajan con un VRR de Q=3, ya que PA en T=6 completaria su quantum de 3, teniendo que ser desalojado y teniendo más prioridad que PA quien ya estaba en ready, además en T=9 podemos ver que PB es desalojado de la CPU pesar de tener aun mas rafagas para ejecutar, pero al agotar su quantum, es desalojado.

En conclusión, el algoritmo de planificación utilizado es VRR.

3. "A E E A Yo soy Sabalero. A E E A Sabalero, Sabalero".

A (1 instancia)	E (1 instancia)	Sabale (1 instancia)

while(1) {
wait(A);
print("A");
signal(E); x2
wait(A); x2
print("A");
signal(Sabale);
}
	while(1) {
	wait(E);
	print("E");
	signal(A);
	}
		while(1) {
		wait(Sabale);
		print("Yo soy Sabalero");
		signal(A);
		wait(Sabale);
		print("Sabalero, Sabalero")
		signal(A);
		}

Semaforos: A=1 E=0 Sabale=0