Sistemas Operativos
1° Parcial 1C2022 - TM - Resolución

<u>Aclaración</u>: 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 actualmente se está ejecutando en modo Kernel, por ejemplo, la rutina de atención de otra interrupción o la llamada a una syscall, no se necesitaría realizar un cambio de modo para cambiar el contexto y atender la interrupción pendiente.
    b. V. Las llamadas al sistema son realizadas por los procesos de usuario, por lo que las mismas deben cambiar a modo Kernel no solo para poder cambiar de contexto sino también porque normalmente las mismas terminan ejecutando instrucciones privilegiadas.
2. Se podría lograr manejar concurrencia sin bloquear distintos hilos, cuando por ejemplo realizaran tareas bloqueantes, mientras al mismo tiempo dentro de un KLT que sea cpu-bound realizar tareas concurrentes con muy bajo overhead. No sería posible forzar jacketing para todos los ULTs, dado que el sistema operativo no puede forzar a que un proceso use una biblioteca en particular, y los ULTs no son administrados por el S.O.
3. El proceso puede estar en estado suspendido-listo. Puede ocurrir que mientras espera un evento (bloqueado) sea suspendido (suspendido-bloqueado) y luego se produzca dicho evento. Mientras el planificador de mediano plazo no decida restaurarlo a memoria principal, el proceso queda en estado suspendido-listo.
4. No sería recomendable ya que, al ser una constante, estos accesos serán siempre de lectura, por lo que no habría una condición de carrera. Usar el semáforo de todas maneras generaría un overhead innecesario.
5. Al pedir esos recursos en un orden predefinido, ese grupo de procesos ya no podría tener una espera circular por los mismos. Sin embargo, no sería fácil implementar esta misma estrategia para todo el sistema ya que sería muy restrictivo que todos los recursos deban ser pedidos en orden.

Práctica

1.

Errores:

1. T = 2 -> El estimado inicial de K3 es 1, K2 ya ejecutó 1 y su estimado era de 3, por lo que creemos que le queda 2, debería haberlo desalojado y ejecutar a K3
2. T = 5 -> La biblioteca de ULTs había sido desalojada por el planificador, no se entera, por lo que no tiene que replanificar, de hecho la ráfaga de UA no había terminado, por lo que no ocurrió un evento considerado por FIFO.
3. T = 6 -> K2 se desbloquea y queda con un nuevo estimado de 2,5, menor a 3 que es lo que creemos que le queda a K1, se debería haber desalojado
4. T = 9 -> Hay dos errores
    a. No hay razón para desalojar a K3, aunque se haya pasado de su estimado no se desaloja, se espera a que termine y luego se penaliza en la siguiente estimación.
    b. K1 debería haber sido elegido en lugar de K2 , a K1 creemos que le quedan 2 y K2 tiene un nuevo estimado de 2,5
5. T = 10 -> Hay dos errores
    a. K3 debería haber sido elegido en lugar de K1, lo que creemos que le queda es negativo.
    b. La biblioteca usa FIFO, por lo que no desaloja, no debería haber cambiado de ULT.
6. T = 11 -> El fin de un ULT no genera una replanificación del SO

2.

mutex_codigo = 1
formulario_lleno = 0
censista = [0, 0, 0, 0]
censo = [0, 0, 0.....] (N)
fin_censo = [0, 0, 0, 0]

Habitante (N)	Censista (4)	Web INDEC (1)

wait(mutex_codigo)
codigo = llenarFormulario()
signal(mutex_codigo)
signal(formulario_lleno)
wait(censo[getId()])
id_censista = idCensista()
darAlCensista(codigo)
signal(fin_censo[id_censista])
	while(TRUE){
	wait(censista[getId()])
	id_habitante = visitarDomicilio()
	signal(censo[id_habitante])
	wait(fin_censo[getId()])
	confirmarHabitanteCensado()
	}
		while(TRUE){
		wait(formulario_lleno)
		validarDatos()
		id_censista = elegirCensista()
		signal(censista[id_censista])
		}

3.

Necesidades máximas

	R1	R2	R3	R4
P1	2	1	0	0
P2	0	1	4	2
P3	6	5	5	2
P4	6	5	3	4
P5	2	5	8	0

Recursos asignados

	R1	R2	R3	R4
P1	2	1	0	0
P2	0	0	0	2
P3	4	3	0	0
P4	4	5	3	2
P5	2	3	3	0

Necesidades pendientes

	R1	R2	R3	R4
P1	0	0	0	0
P2	0	1	4	0
P3	2	2	5	2
P4	2	0	0	2
P5	0	2	5	0

Recursos totales

R1	R2	R3	R4
12	12	8	6

Recursos disponibles

R1	R2	R3	R4
0	0	2	2

a)
Disponibles = 0 0 2 2
P1 finaliza = 2 1 2 2
P4 finaliza = 6 6 5 4
P5 finaliza = 8 9 8 4
P2 finaliza = 8 9 8 6
P3 finaliza = 12 12 8 6 => hay secuencia segura, está en estado seguro

b)
i) P3 -> 2 R4

Es una solicitud válida(menor a su necesidad) y está disponible. Hay que simular la asignación y ver si el sistema queda en estado seguro.

Asignados P3 = [4,3,0,2]
Necesidad P3 = [2,2,5,0]
Disponibles = 0 0 2 0
P1 finaliza = 2 1 2 0 => No puedo asignar ningún vector de necesidad, no hay secuencia segura, quedaría en estado seguro, por lo tanto no realizaría la asignación.

ii) P2 -> 1 R4

Es una solicitud inválida(mayor a su necesidad) y, aunque esté disponible, no se asigna.

c) Falso. Esta estrategia se basa en el caso pesimista, es decir, que todos los recursos vayan a pedir el máximo de sus recursos, y que no liberen recursos hasta finalizar su ejecución. Sin embargo puede ocurrir que no requieran todos esos recursos, o que los vayan liberando a medida que no los requieren más, permitiendo que el resto se ejecute sin generar interbloqueos.