Sistemas Operativos
1° Parcial 2C2022 – TT – 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. Modos de ejecución:
    a. Modo kernel: se pueden ejecutar todas las instrucciones disponibles en la cpu. Es usado por el Sistema Operativo.
    b. Modo usuario: se pueden ejecutar solamente las instrucciones no privilegiadas. Es usado por los Procesos.

    Eventos que podrían generar un cambio a modo kernel:
    - Interrupción
    - Syscall
2. Atributos:
    a. Exclusivamente en PCB: PID, PPID, recursos asignados (archivos, memoria), limites de memoria
    b. Exclusivamente en TCB: contexto de ejecución, estado, puntero a la pila

    Un hilo no podría leer su TCB, dado que al ser un hilo de kernel su estructura se encuentra en espacio del sistema operativo.
3.
    a. Verdadero, al no conocer la entrada salida y al ser el algoritmo no expropiativo, sería imposible que la biblioteca decida cambiar de hilo sin que el mismo finalice, ya que una vez pase a ejecutar un hilo, nunca volvería a encolarse (no lo haría luego de una entrada salida, y la biblioteca nunca lo desalojaría).
    b. Falso. En un sistema multiprocesador y una sección crítica no muy extensa suele ser más eficiente utilizar una solución con espera activa ya que el hecho de no bloquear procesos minimiza el overhead y en ese escenario no sería frecuente que los procesos deban desperdiciar varias ráfagas de CPU para ingresar a la sección crítica.
4. Finalizar procesos o desalojar recursos.
    Finalizar procesos: Se pueden finalizar todos los procesos involucrados o hacerlo uno por uno hasta solucionar el problema. La primera solución es más sencilla pero puede llevar a una utilización más ineficiente de los recursos. La segunda podría generar mucho overhead si se deben finalizar varios procesos ya que se debe correr el algoritmo luego de elegir cada víctima.
    Para elegir las víctimas, ya sea que finalice procesos o desaloje recursos, el sistema

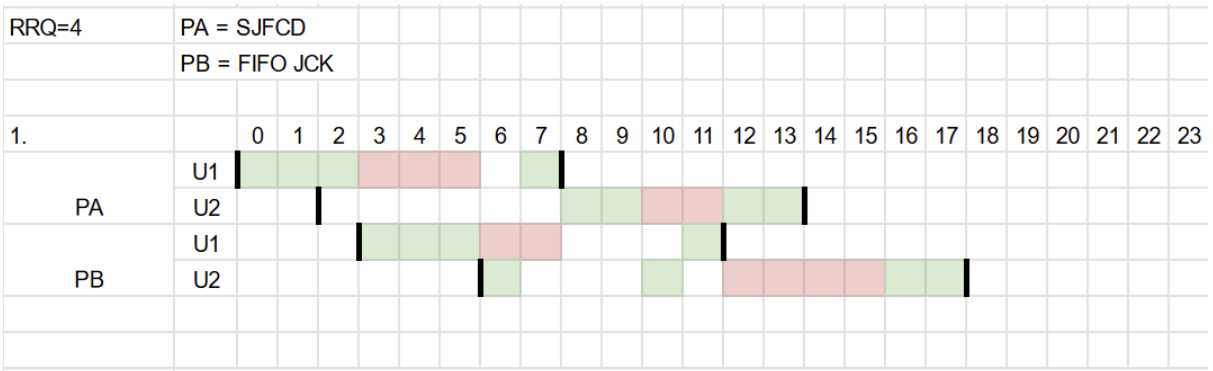
operativo considerará, entre otras cosas, la prioridad de cada proceso, cuántos y qué tipos de recursos está utilizando y el tiempo que el proceso lleva en el sistema.

5. Los semáforos necesitan ejecutar su código de forma atómica para evitar los problemas de las condiciones de carrera y en su implementación deben bloquear procesos, por lo cuál las mismas son herramientas provistas por el sistema operativo.
   Las soluciones de software como la solución de Peterson utilizan espera activa y en general requieren mucho código para ser implementadas y suelen estar limitadas a 2 procesos.

Práctica

1. RESOLUCION

RRQ=4	PA = SJFCD																							
	PB = FIFO JCK																							
1.		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	U1																							
PA	U2																							
	U1																							
PB	U2																							
2.	No cambia																							
3.	Instante 6																							



2.

Semáforos:
sem_encender_maquina = 0
sem_mano_derecha = 0
sem_acomodar_anteojos = 0
sem_boton = 0
pantalla_encendida = 0
sem_palanca_der = 0
bajar_pinza = 0
tomar_premio = 0
mutex_hueco = 1

Mano Derecha	Mano Izquierda	Máquina	Botón

wait(sem_mano_derecha)
recibir_ticket_maquina()
signal(acomodar_anteojos)
wait(sem_palanca_der)
mover_palanca_a_derecha()
pedir_bajar_pinza()
signal(bajar_pinza)
	depositar_moneda()
	signal(sem_encender_maquina)
	wait(acomodar_anteojos)
	acomodar_anteojos()
	for(1…5) {
	pulsar_boton()
	signal(sem_boton)
	}
	
	wait(tomar_premio)
	wait(mutex_hueco)
	tomar_premio()
	signal(mutex_hueco)
		wait(sem_encender_maquina)
		encender()
		dar_ticket_id()
		signal(sem_mano_derecha)
		wait(pantalla_encendida)
		printf(“Ready!”)
		signal(sem_palanca_der)
		
		wait(bajar_pinza)
		bajar_pinza()
		tomar_y_soltar_peluche()
		signal(tomar_premio)
			wait(sem_boton) x 5
			encender_pantalla()
			signal(pantalla_encendida)

3.

a)
Necesitados

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

Vector de Disponibles inicial = {1, 2, 1, 2}

Puede finalizar el proceso 2 que tiene los Recursos {1,2,1,0}

Vector de disponibles = {2,4,2,2}
Puede finalizar el proceso 1 que tiene los Recursos {0,0,1,1}

Vector de disponibles = {2,4,3,3}
Puede finalizar el proceso 4 que tiene los Recursos {1,0,0,0}

Vector de disponibles = {3,4,3,3}
Puede finalizar el proceso 3 que tiene los Recursos {0,0,1,1}

Vector de disponibles final = {3,4,4,4}

Hay secuencia segura P2 -> P1 -> P4-> P3. ES ESTADO SEGURO

b)
Necesitado:

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

Se simula la asignación de una instancia de R3 a P3.
Los “Recursos Necesitados” por R3 pasan a ser {3,2,0,1}

Vector de Disponibles inicial = {1, 2, 0, 2}
Puede finalizar el proceso 2 que tiene los Recursos {1,2,1,0}

Vector de disponibles = {2,4,1,2}
Puede finalizar el proceso 1 que tiene los Recursos {0,0,1,1}

Vector de Disponibles {2,4,2,3} Ya no puede finalizar otro Proceso. NO ES ESTADO SEGURO.

c) No hubiera sido la misma respuesta porque al finalizar P4 liberaría una instancia de R1 que era lo que le estaba faltando a P3 para que haya estado seguro en respuesta b.

d) Es pesimista porque el recurso estaba disponible y el SO denegó la asignación. Podría haber ocurrido que luego de esa asignación el proceso 3 finalice su ejecución y liberase todos sus recursos. Sin riesgo de deadlock en este caso particular.