Sistemas Operativos
1° Parcial 1C2023 - 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. Se realiza un cambio de contexto y un cambio de modo (usuario => kernel), esto implica guardarse el contexto de ejecución (registros, PC, PSW, etc) del proceso en ejecución en la pila del sistema y cargar el nuevo contexto de ejecución de la rutina del SO que corresponda a esa syscall. Si llega una interrupción durante la ejecución de esta rutina, el procedimiento es similar, se debe cambiar el contexto pero en este caso no de modo ya que la CPU ya estaría ejecutando en modo kernel.

2.
    a) Falso, el cambio de proceso lo realiza el SO y por lo tanto se requiere estar en modo kernel. El cambio entre ULTs se realiza en modo usuario siempre que estemos hablando de hilos **pares** (pertenecientes al mismo KLT/Proceso) ya que sería realizado por su biblioteca de hilos.
    b) Verdadero, utilizar KLTs en sistemas multiprocesador permite que los mismos se ejecuten en **paralelo** ya que el SO puede planificarlos en las distintas CPU.

3. Propondría un algoritmo de planificación de colas multinivel con dos colas:
   - Cola 1 (de mayor prioridad, para procesos críticos): FIFO
   - Cola 2 (de menor prioridad, para procesos ordinarios): VRR

   Se pide que los procesos críticos se ejecuten lo antes posible por lo que el algoritmo de prioridades entre colas sería **con** desalojo.
   VRR prioriza más frecuentemente a los procesos más cortos, reduciendo el tiempo de espera promedio y garantizando que ningún proceso monopolice la CPU. El quantum debería alcanzar para que la mayoría de las ráfagas finalicen normalmente.
   Otra opción podría ser SRT pero al estimar podría ejecutarse un proceso que termine ejecutando indefinidamente y no se lo desalojaría.

4. Condición de carrera es una situación bajo la cual dos o más procesos/hilos comparten recursos y el estado final de los mismos luego de ejecutar no siempre es determinístico al depender de la velocidad relativa de ejecución de cada proceso/hilo y varios otros factores. Un ejemplo es el de varios hilos modificando una variable global de la forma CONTADOR++. Es difícil de detectar porque algunas veces podría arrojar resultados coherentes dependiendo de la ejecución, por lo tanto no sería tan evidente el error.

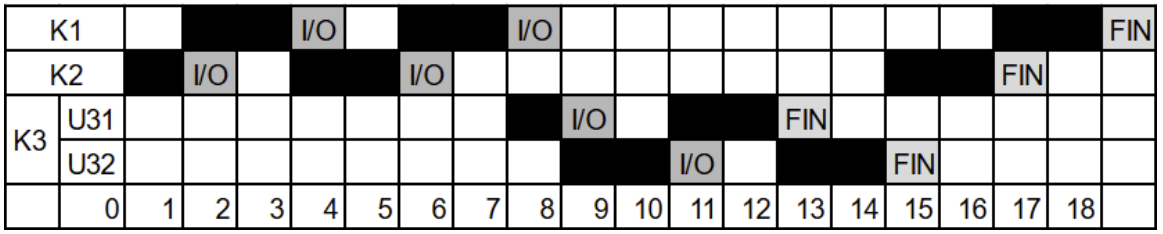
5. Comparacion:

	Ocurrencia	Overhead	Flexibilidad
Evasión	No ocurre	Alto, se corre el algoritmo del banquero ante cada petición.	Peticiones: El proceso debe declarar y cumplir peticiones máximas. Asignaciones: Cada asignación debe dejar al sistema en estado seguro.
Detección	Podría ocurrir	Medio, depende de la frecuencia con que se ejecute el algoritmo.	No se limitan las peticiones ni la asignación de recursos más allá de los disponibles.

Práctica

1. .

LLegada: K2->0, K1->1, K3->2



a) El SO utiliza un algoritmo HRRN, esto puede verse en:
   i) t=5 -> se elige para ejecutar a K1 a pesar de haber llegado a ready después que K3, por lo que se están priorizando los procesos cortos, en este caso K3 tendría que ejecutar una ráfaga de 7 unidades de tiempo.
   ii) t=7 -> se elige a K3 debido a que el tiempo de espera es tal que su Response Ratio supera al de K2 en ese momento a pesar de que éste tiene una ráfaga más corta (2 unidades de tiempo).
   RR(K2) = 1 + 1/2 = **1.5** ; RR(K3) = 1 + 5/7 = **1.71**

b) Dos desventajas de HRRN pueden ser:
   i) Es un algoritmo sin desalojo, por lo que algún proceso podría llegar a monopolizar la CPU.
   ii) Tiene mucho overhead debido a tener que recalcular el Response Ratio en cada evento de replanificación, además se deberían estimar las ráfagas de CPU para poder implementarlo.

c) La biblioteca de hilos de K3 podría estar utilizando FIFO, RR (Q >=2) o SJF.

2.

Cliente (N instancias)	Reservas (2 instancias)	Devoluciones (2 instancias)

libro = elegirLibro()
wait(hayCopias[libro.id])
wait(mutexReservas)
agregar(reservas, libro)
signal(mutexReservas)
signal(hayReservas)
 	
wait(hayRetiros[libro.id])
irABiblioteca()
wait(mutexRetiros[libro.id])
retirar(retiros[libro.id])
signal(mutexRetiros[libro.id])
 	
estudiar()
irABiblioteca()
wait(mutexDevoluciones)
agregar(devoluciones, libro)
signal(mutexDevoluciones)
signal(hayDevoluciones)
	while(1) {
	wait(hayReservas)
	wait(mutexReservas)
	libro = tomar(reservas)
	signal(mutexReservas)
	
	wait(mutexRetiros[libro.id])
	agregar(libro, retiros[libro.id])
	signal(mutexRetiros[libro.id])
	signal(hayRetiros[libro.id])
	}
		while(1) {
		wait(hayDevoluciones)
		wait(mutexDevoluciones)
		libro = tomar(devoluciones)
		signal(mutexDevoluciones)
		
		registrarDisponible()
		signal(hayCopias[libro.id])
		
		wait(mutexContFestejo)
		if(contFestejo < 100){
		contFestejo++
		signal(mutexContFestejo)
		} else {
		contFestejo = 0;
		signal(mutexContFestejo)
		festejar();
		}
		}

Semáforos:
hayCopias[5] = {20, 20, 20, 20, 20}
mutexReservas = 1
hayReservas = 0
mutexRetiros[5] = {1, 1, 1, 1, 1}
hayRetiros[5] = {0, 0, 0, 0, 0}
mutexDevoluciones = 1
hayDevoluciones= 0
mutexContFestejo = 1

3. .

VT = [2, 2, 2, 2]

Asignaciones:

	R1	R2	R3	R4
P1	1		1	
P2	1		1	
P3		1		1
P4		1		
P5				1

Peticiones actuales:

	R1	R2	R3	R4
P1	1	1		
P2		1		1
P3			1	1
P4	1			
P5	1		1	

a) VD= [0, 0, 0, 0]
Existe deadlock y todos los procesos están involucrados.

b) Si agregáramos 1 R1 -> VT = [3, 2, 2, 2]:
Disponibles = [1, 0, 0, 0] -> podríamos asignar a P4
Si P4 libera = [1, 1, 0, 0] -> podríamos asignar a P1
Si P1 libera = [2, 1, 1, 0] -> podríamos asignar a P5
Si P5 libera = [2, 1, 1, 1] -> podríamos asignar a cualquier otro y no habría deadlock

c) VT = [2, 2, 3, 3]:
Disponibles = [0, 0, 1, 1] -> podríamos asignar a P3
Si P3 libera = [0, 1, 1, 2] -> podríamos asignar a P2
Si P2 libera = [1, 1, 2, 2] -> podríamos asignar a cualquier otro y no habría deadlock