Sistemas Operativos
1° Recuperatorio Parcial 1C2017 - TM - 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.

TEORIA:

1)

- Se produce la interrupción, frenando la ejecución del proceso actual
- Al finalizar la instrucción actual se chequea si hay interrupciones pendientes
- Se guarda el contexto de ejecución del proceso actual en la pila del sistema operativo
- Se produce un cambio de modo usuario a modo kernel
- Se pasa el control al gestor de interrupciones del sistema operativo
- El sistema operativo pasa el proceso bloqueado a la lista de listos
- Suponiendo que el planificador de corto plazo no implica desalojo:
    - Se cambia el modo de ejecución de kernel a usuario
    - Se restaura el contexto de ejecución del proceso original de la pila del sistema

2)

a. Verdadero. Los KLTs de un mismo proceso comparten recursos, por lo tanto, recién cuando finalice el proceso se liberarán los mismos. Entonces, si un usuario no libera correctamente la memoria que pide dinámicamente en sus KLTs, podrán existir problemas de secciones alocadas no referenciadas hasta que el proceso finalice y se liberen.

b. Falso. El jacketing salva la situación de que un ULT bloqueado bloquee a otro ULT del mismo KLT/proceso asociado. Sin embargo, para el SO sigue siendo una única unidad de planificación, por lo que no se podría aprovechar el multiprocesamiento en caso de tener más de un procesador.

3)

Running -> Ready: cuando el proceso es desalojado del procesador, por ejemplo al finalizar el quantum en RR
Suspended/Ready -> Ready: Cuando el nivel de multiprogramación disminuye y el planificador de mediano plazo decide llevar al proceso a memoria
Ready -> Exit: cuando se decide finalizar el proceso y el mismo no estaba ejecutando
Ready -> Blocked: esta transición no es posible

4)

Se intenta solucionar la condición de carrera, es decir, que se puedan obtener distintos resultados dependiendo en el orden en que se ejecuten distintos procesos/hilos. Esto se debe a que acceden a datos compartidos y que acciones que para nosotros son atómicas en realidad no lo son para el procesador. Por lo tanto, es necesario lograr que esas instrucciones (Sección crítica) ejecuten atómicamente.

Ejemplo Condición de carrera:
var común -> a = 0

P1	**P2**

var c= a;
c = c + 1;
a = c;
 	
print(a)
	var b= a;
	b = b - 1;
	a = b;
	
	printf(a)

Según el orden en que ejecuten pueden dar distintos resultados como:
Si se ejecutan todas las instrucciones de un proceso y luego las del otro, el resultado final de a será 0.
Sin embargo, si primero ejecutan la primera instrucción en ambos casos, el resultado final podrá ser 1 o -1 según cuál sea el último que setee el valor a a.

… usando mutex
mutex = 1

P1	**P2**

wait(mutex)
var c= a;
c = c + 1;
a = c
 	
print(a)
signal(mutext);
	wait(mutex)
	
	var b= a;
	b = b - 1;
	a = b;
	
	printf(a)
	
	signal(mutext);

… deshabilitando interrupciones

P1	**P2**

deshabilitarInterrupciones()
var c= a;
c = c + 1;
a = c
 	
print(a)
	deshabilitarInterrupciones()
	
	var b= a;
	b = b - 1;
	a = b;

habilitarInterrupciones();	printf(a) habilitarInterrupciones()

5) La estrategia de prevención del interbloqueo consiste a grandes rasgos en diseñar un sistema de manera que esté excluida la posibilidad de interbloqueo, existen dos métodos:

*Indirectos: Impiden la aparición de las tres condiciones antes mencionadas.

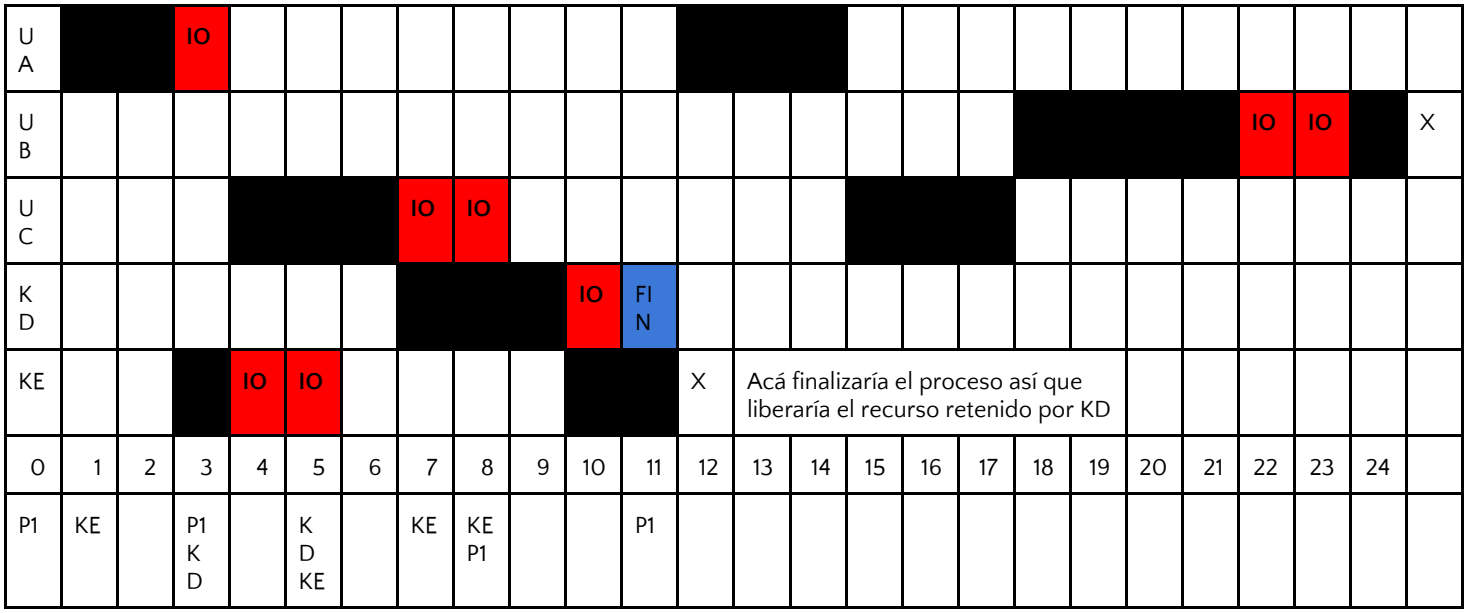
*Directos: Evitan la aparición del círculo vicioso de espera

PRACTICA

Ejercicio 1:

Proceso 1			Proceso 2	
Llegan en T=O			T=3	T=1
ULT A	ULT B	ULT C	KLT D	KLT E
CPU(2)	CPU(4)	CPU(3)	CPU(3)	CPU (1)
Impresora (1)	Impresora(2)	Impresora(2)	Impresora(2)	Impresora(2)
CPU(3)	CPU(1)	CPU(3)	CPU(1)	CPU(2)

a)



b)
En T = 24 finaliza la ejecución de todos los hilos

c)
En T = 3 KD llega de nuevo (resultado syscall) y P1 vuelve a listo de una interrupción de fin de IO. Como primero se debe tratar la interrupción, queda primero P1 en la cola FIFO

Ejercicio 2:

maxBusquedas = 10
m_colaBusquedas = 1
busquedaPendiente = 0

m_colaSecciones = 1
seccionesPendientes = 0

resultadosParcialesListos (array de 10 semáforos) = {0, 0,...0}

resultadoListo (array de 10 semáforos) = {0,0,...0}

m_resultados = 1

Peter + amigos (5)	Distribuidor búsquedas (M)	Contador(N)

palabra = ingresarPalabra();
archivoLibro = ingresarPathArchivo();
 	
w(maxBusquedas)
w(m_colaBusqeudas)
idBusqueda = agregarBusqueda(colaBusquedas, busqueda, archivoLibro)
s(m_colaBusqeudas)
 	
s(busquedaPendiente)
 	
w(resultadoListo[idBusqueda]
 	
w(m_resultado)
cantidad = obtenerResultado(colaResultados, idBusqueda);
s(m_resultado)
	while(1) {
	
	w(busquedaPendiente)
	
	w(m_colaBusqeudas)
	busqueda = obtenerBusqueda(colaBusquedas)
	s(m_colaBusqeudas)
	
	w(m_colaSecciones)
	seccionesAAnalizar = dividirAnalisis(busqueda, colaSecciones)
	s(m_colaSecciones)
	
	s(seccionesPendientes, seccionesAAnalizar)
	
	w(resultadosParcialesListos[busqueda.idBusqeu
		while(1){
		
		w(seccionesPendientes)
		w(m_colaSecciones)
		seccion = obtenerSeccion(colaSecciones)
		s(m_colaSecciones)
		
		cantidadParcial = contarOcurrencias(seccion)
		
		agregarResultadoParcial(seccion.idBusqueda, cantidadParcial)

printf(cantidad);	da], seccionesAAnalizar)
	resultadosParciales = obtenerResultadosParciales(busqueda.id)
	
	cantidadTotal = sumarizar(resultadosParciales);
	
	w(m_resultado)
	notificarResultadoTotal(cantidadTotal);
	s(m_resultado)
	
	s(resultadoListo[busqueda.idBusqueda]
	s(maxBusquedas)
	}
		s(resultadosParcialesListos[seccion.idBusqeuda])
		
		}

Podrían agregar un mutex sobre las funciones que modifican los resultados parciales, como el enunciado no dice nada de si esa función ya es atómica o que hace, no es necesario que lo agreguen

Ejercicio 3

Dadas las siguientes matrices:

Máx	R1	R2	R3	R4
P1	0	0	1	2
P2	2	7	5	0
P3	6	6	5	6
P4	4	3	5	6
P5	0	7	5	2

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

Recursos máximos: (6, 8, 12, 12)

a) ¿El sistema está en estado seguro? Justifique utilizando el algoritmo apropiado.

b) Sabiendo que los recursos disponibles NO varían ¿el sistema aceptaría una solicitud de P2 de 2 instancias de R2? Justifique utilizando el algoritmo apropiado.

c) En caso de que la respuesta a la pregunta anterior sea afirmativa, indique al menos dos secuencias en las cuales procesos finalizarán. En caso contrario, ¿Podría afirmarse que de otorgarse el recurso, se producirá un deadlock? ¿Por qué?

Nec	R1	R2	R3	R4
P1	0	0	0	0
P2	0	7	5	0
P3	6	6	2	2
P4	2	0	0	2
P5	0	4	2	0

Asignados	4	6	12	12

Disponibles	2	2	0	0

a) Se utiliza evasión, algoritmo del banquero

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

b) P2 -> 2 R2
Es una solicitud correcta (menor a su necesidad) y está disponible. Hay que simular la asignación y ver si el sistema queda en estado seguro.

Nuevos disponibles = 2 0 0 0
P1 finaliza = 2 0 1 2
P4 finaliza = 4 3 6 6
No se puede ejecutar a ninguno más en la simulación ->deja al sistema en estado inseguro => 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.