Sistemas Operativos
1° Parcial 1C2022 - TT - 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. El Proceso no podría escribir en el espacio de direcciones de otro proceso, por lo tanto se produciría una interrupción. Se produciría un cambio a modo kernel, y el sistema operativo atendería la misma, tomando alguna acción relativa al manejo de dicho error, como por ejemplo finalizar dicho proceso. Si fuera el Sistema Operativo el que escribiera por error, no se produciría interrupción alguna, dado que el SO ejecuta en modo kernel y por lo tanto tiene permisos para acceder a cualquier dirección de memoria (sin embargo, el comportamiento del sistema podría arrojar resultados inesperados)
2.
   Prevención:
   1. Posibilidad de ocurrencia: No es posible porque no se cumple una de las condiciones para deadlock.
   2. Overhead: Bajo. A diferencia de los anteriores no requiere cálculos con matrices.
   3. Limitación para la solicitud: Limitados a las políticas propias del sistema/programador para prevenir alguna de las condiciones.
   4. Limitación para la asignación: Limitados a las políticas propias del sistema/programador para prevenir alguna de las condiciones.

   Evasión:
   1. Posibilidad de ocurrencia: No es posible que ocurra, el sistema debe estar siempre es estado seguro.
   2. Overhead: Alto, cálculos con matrices con cada solicitud de recurso.
   3. Limitación para la solicitud: No hay limitación.
   4. Limitación para la asignación: Aunque haya recursos disponibles, puede negarse la solicitud.

   Detección:
   1. Posibilidad de ocurrencia: Es posible que ocurra Deadlock.
   2. Overhead: Medio, cálculos con matrices cada ciertos períodos de tiempo o por ciertas condiciones en el sistemas.
   3. Limitación para la solicitud: No hay limitación
   4. Limitación para la asignación: Mientras existan recursos disponibles, estos son asignados a los procesos a demanda.

3. No deberían sufrir deadlock ya que, de estar los wait y signal bien ubicados y el semáforo inicializado correctamente, se garantizará la mutua exclusión y solo uno de los Procesos podrá utilizar la porción de memoria en un momento dado. Básicamente no se cumple la 'espera y retención', condición necesaria para la ocurrencia de un deadlock. El deadlock podría presentarse si hay más de un recurso a proteger y más semáforos asociados a estos nuevos recursos.
   Es posible que alguno de los Procesos sufra starvation, dado que el semáforo no garantiza el orden de desbloqueo de los procesos de su cola ante un signal. De este modo, un proceso podría bloquearse en el semáforo junto a otros, pero no ser desbloqueado si es que existen más procesos que siguen ejecutando un wait sobre ese mismo semáforo y son liberados antes.
4.
   a. Cuando un Proceso es desalojado por una interrupción, la biblioteca de hilos de usuarios no se entera, por lo tanto no cambia el estado del ULT que estaba en ejecución en ese momento. En esta situación podría ocurrir que el Proceso llegase a la cola de Ready mientras que el estado del ULT diría Running.
   b. Un microkernel sí tiene la ventaja de poder agregar nuevas funcionalidades, o arreglar código existente, sin la necesidad de recompilar todo, pero sí es tolerante a fallos, y por ende más estable. Un error en un módulo del microkernel podría no afectar al resto. Pero un error en un kernel monolítico podría ser fatal para todo el sistema.
5.
   SJF sin desalojo:
   1. Aging: No posee, la prioridad de los procesos no varía mientras espera en Ready.
   2. Starvation: Sí, cuando la siguiente ráfaga es grande.
   3. Prioridad para procesos limitados por la E/S: Tienen ráfagas de CPU cortas, por lo tanto tienen prioridad alta.

   HRRN:
   1. Aging: Si posee, la prioridad de los procesos aumenta mientras esperan en Ready.
   2. Starvation: No.
   3. Prioridad para procesos limitados por la E/S: Tienen ráfagas de CPU cortas, por lo tanto tienen prioridad alta. Aunque si existiesen procesos con ráfaga de CPU más largas pero con mayor tiempo de espera, estos podrían tener mayor prioridad.

   VRR:
   1. Aging: No.
   2. Starvation: No.
   3. Prioridad para procesos limitados por la E/S: Al momento de planificar un proceso, los procesos con ráfaga de CPU chicas (limitados por la E/S) y con ráfaga restante de CPU chica, tienen mayor prioridad frente a los procesos con ráfagas que aprovechan todo el quantum.

Práctica

1. Una posible solución (depende del ult que elijan para comenzar)

VRR, Q=2 - E/S manejada por Biblioteca

SO							X	X	X		X	X	X		X	X		X						X							
KA	ULT1			X	X	X	C	C	C	C							X		R	R	R							X (f)			
KA	ULT2									X				C	C	C						X	X		R	R	R	R		X	X (f)	
KB	ULT3	X	X	C	C	C								X	R	R	R	R							X (f)							
KB	ULT4						X				C	C	C	C						X	X	X	R	R	R		X	X	X (f)				
		0	1	2	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

VRR, Q=2 - E/S manejada por SO

SO							X	X	X							X						X				X					
KA	ULT1			X	X	X	C	C	C	C	X	R	R	R	X (f)																
KA	ULT2															X		C	C	C	X	X		R	R	R	R	X	X (f)		
KB	ULT3	X	X	C	C	C	X	R	R	R	R	X (f)																			
KB	ULT4												X	C	C	C	C	X	X	X	R	R	R	X	X	X (f)					
		0	1	2	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29			

2. –

Variable compartida: Lista

MUTEX = 1, HAY_OP = 0, MAX_OPS = 20, TURNO_A = 1, TURNO_B = 0

Trader	Arbitrador A	Arbitrador B

while(TRUE){
Op.vender = GREEN
Op.comprar = SUN
WAIT(MAX_OPS)
WAIT(MUTEX)
depositar(Op, Lista)
SIGNAL(MUTEX)
SIGNAL(HAY_OP)
}
	while(TRUE){
	WAIT(TURNO_A)
	WAIT(HAY_OP)
	WAIT(MUTEX)
	Op = retirar(Lista)
	SIGNAL(MUTEX)
	SIGNAL(MAX_OPS)
	destruir(Op.vender)
	crear(Op.comprar)
	SIGNAL(TURNO_B)
	}
		while(TRUE){
		WAIT(TURNO_B)
		WAIT(HAY_OP)
		WAIT(MUTEX)
		Op = retirar(Lista)
		SIGNAL(MUTEX)
		SIGNAL(MAX_OPS)
		destruir(Op.vender)
		crear(Op.comprar)
		SIGNAL(TURNO_A)
		}

3.

a)

Los Recursos disponibles se calculan haciendo Rec Totales – Rec Asignados

Disp = {1,1,0,0}

Se marca P4 por no tener recursos asignados

Finaliza el P5 y los disponibles quedan Disp = {1,2,0,0}

Finaliza el P3 y los disponibles quedan Disp = {1,2,0,2}

Ya no hay procesos para finalizar, entonces hay deadlock entre los procesos P1 y P2.

P1 y P2 en Deadlock. P3 y P5 no tienen problemas. P4 Inanición, a la espera que P3 y P5 liberen recursos .

b)

Como P2 tiene menor prioridad que el P1, entonces es finalizado por el SO.

Se marca P4 por no tener recursos asignados, y se finaliza P5 y P3.
Se elimina P2 y los disponibles quedan Disp = {2,2,2,2}
Finaliza el P1 y los disponibles quedan Disp = {3,3,3,3}
Eliminando el proceso 2 y al poder finalizar el P1,se muestra que ya no hay deadlock.

c)
Puede haber varios criterios. Algunos ejemplos podrían ser:
-Seleccionar para eliminar al proceso que tenga más recursos asignados.
-Eliminar los procesos que más tiempo llevan en el sistema.