Sistemas Operativos
1 Recuperatorio 1° Parcial 2C2022 - 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. Una condición de carrera se produce cuando procesos o hilos comparten recursos y según el orden en que se ejecutaron puede variar el resultado. La sección crítica es la parte del proceso donde puede producirse la condición de carrera y para evitarla se deben utilizar técnicas de exclusión mutua.
   Se puede utilizar semáforos o monitores para asegurar la exclusión mutua.
2. a) Falso: No evita el Deadlock. El algoritmo de detección se ejecuta periódicamente para detectar y recuperar deadlocks.
   b) Verdadero: Al evitar la retención y espera puede asignarse todos los recursos que necesite un procesos aunque no los necesite inmediatamente, impidiendo que otro proceso los utilice.

3. Un ejemplo podría ser realizar una lectura de disco y que el mismo se encuentre ocupado, por lo que el sistema operativo bloquearía al proceso hasta que la i/o en curso finalice, para evitar overhead de parte del proceso. En caso de que la syscall fuera no bloqueante, el proceso no se bloquearía y seguiría ejecutando, teniendo que reintentar la operación realizando espera activa.

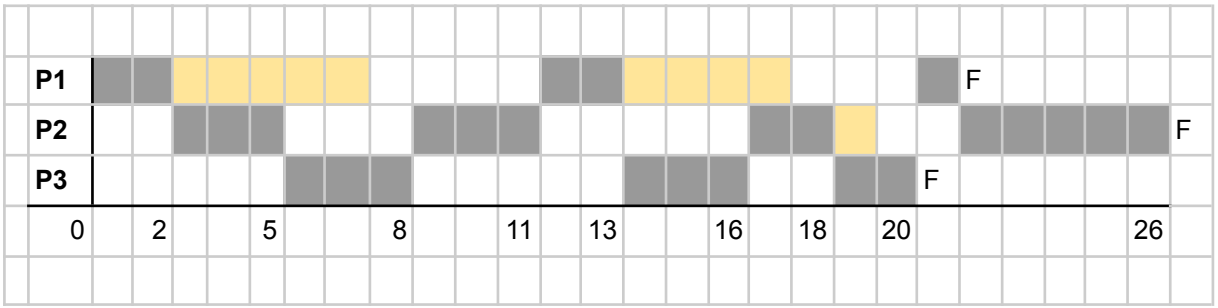
4. El máximo nivel de multiprogramación permite limitar la carga de un sistema en términos de procesos activos, evitando que el mismo quede sobrecargado disminuyendo la productividad del mismo. Configurarlo en niveles muy altos sería equivalente a que el límite no existiera. Configurarlo en niveles muy bajos bajaría la productividad del sistema innecesariamente, porque varios procesos esperarían con la cpu probablemente ociosa.

5. Un motivo para deshabilitar las interrupciones podría ser una syscall de semáforos que esté implementada deshabilitando las mismas para lograr no ser interrumpida en el manejo de las estructuras de los semáforos. Sí, el SO debería ser el único en poder realizar dicho cambio por cuestiones de seguridad y control del sistema.

Práctica

1.

a)



b) P1 = [ráfagas cpu (2+2+1 = 5) + esperas (11-7 + 20-17 = 7)] / [5] = 12/5 = 2,40
P2 = [ráfagas cpu (8+5 = 13) + esperas (2-1 + 8-5 + 16-11 + 21-19 = 11)] / [13] = 24/13 = 1,84
P3 = [ráfagas cpu (8) + esperas (5-2 + 13-8 + 18-16 = 10)] / [8] = 18/8 = 2,25

Dado que la fórmula sopesa el tiempo de ráfagas más la espera con el tiempo de ráfagas, lo que se mide es cuál proceso tuvo que esperar más en Ready acorde a lo que tenía para ejecutar en cpu. Por lo tanto, a un valor más grande, más perjudicado se ve el proceso. Se observa que P1 se ve mayoritariamente perjudicado.
Como dato adicional, se observa que P1 es un proceso IO bound, mientras que los otros son CPU bound. Es esperable que el algoritmo round robin perjudique a los procesos IO bound por sobre los cpu bound.

c) Se podría cambiar el algoritmo a VRR (mismo quantum) ya que priorizaría más a los procesos IO bound.

2.

Necesidad

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

Recursos Disponibles = [1, X, 9, 4]
Recursos Totales = [5, 9+X, 21, 11]

a) Análisis de estado seguro, suponemos un R2 inicial de 0:
Disponibles inicial: [1, 0, 9, 4]
Asigno P4 y libera: [3, 0, 11, 4]
Asigno P1 y libera: [3, 3, 12, 7]
Desde este punto ya puedo asignar en cualquier orden -> **ESTADO SEGURO, no necesito agregar ningún R2 -> X = 0**

b) Si P2 pidiera 2 R2 esto superaría lo máximo restante que puede pedir, por lo que el sistema debería finalizar al proceso por no cumplir con el protocolo de evasión.

3.

Paciente (N instancias)	**Recepción (1 instancia)**	**Médico Clínico (1 instancia)**

wait(recepción)
entregar_documentación()
signal(docu)
 	
wait(entrega_num)
Id_paciente = recibir_número()
 	
wait(atendido[id_paciente])
explicar_problema()
signal(problema)
 	
wait(receta)
recibir_receta()
irse_a_casa()
	while(1){
	wait(docu)
	obtener_documentación()
	nuevo_id = siguiente_id()
	entregar_numero(nuevo_id)
	signal(entrega_num)
	
	wait(mutex_cola)
	add(cola_pacientes, nuevo_id)
	signal(mutex_cola)
	signal(pacientes)
	signal(recepción)
	}
		while(1){
		wait(pacientes)
		wait(mutex_cola)
		id_a_atender = get(cola_pacientes)
		signal(mutex_cola)
		signal(atendido[id_a_atender])
		
		wait(problema)
		escuchar_problema()
		entregar_receta()
		signal(receta)
		}

recepción=1
docu=0
entrega_num=0
pacientes=0
receta=0
problema=0
mutex_cola=1
atendido[20]={0…0}