Sistemas Operativos
1° Parcial 2C2023 - 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

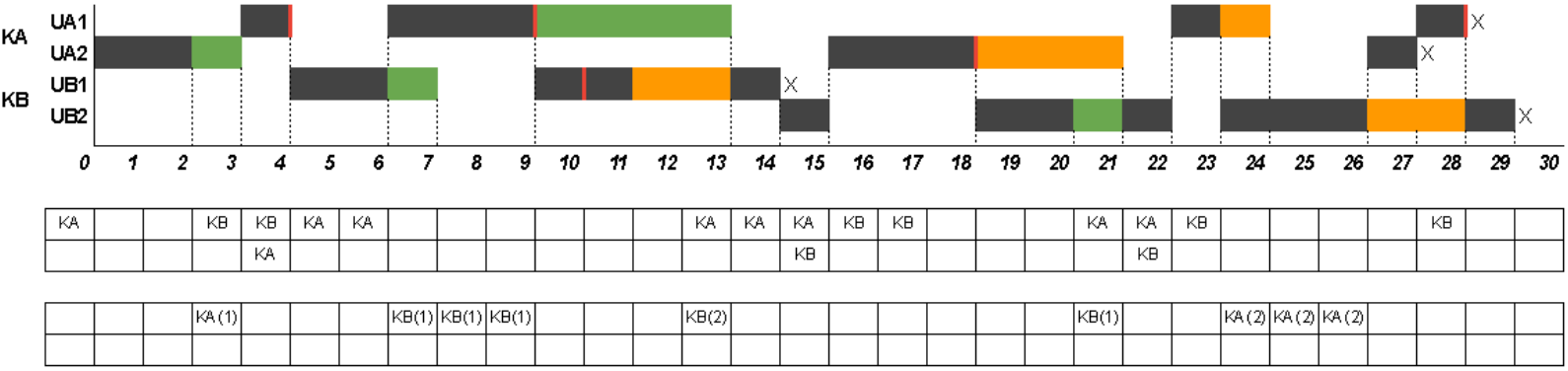
1. Para atender una interrupción (etapa final del ciclo de instrucción), el procesador cambia a modo kernel, se guarda el contexto actual y carga el contexto de la rutina correspondiente a la atención de esa interrupción. Si el contexto anterior ya estaba en modo kernel, por ejemplo porque se estaba ejecutando la rutina correspondiente de otra interrupción o una syscall, no habría cambio de modo pero sí de contexto.
2. Lo más seguro sería crear un proceso por cada petición, de esta manera, aunque algunas de estas fallen y su proceso finalice, no afectarían al resto de los procesos/peticiones. Como desventaja principal, esta propuesta no es tan eficiente en el uso de recursos como utilizar hilos dentro de un mismo proceso.
3. FIFO: tiene el menor overhead, ningún proceso puede sufrir starvation pero cualquiera puede monopolizar la CPU.
   RR: puede tener mucho overhead por cambios de contexto si su Q es pequeño, los procesos no sufren starvation ni pueden monopolizar la CPU.
   Prioridades con desalojo: su overhead suele ser bajo pero puede ser mayor si ocurren desalojos frecuentes, los procesos de menor prioridad podrían sufrir starvation mientras que los de prioridad máxima podrían monopolizar la CPU.
4. a. Falso, además de acceder al recurso en forma concurrente, al menos uno de ellos debe ser en modo escritura.
   b. Verdadero (podría considerarse como falso también), podría ser más performante ya que si la región crítica es corta ese tiempo de espera activa podría ser mejor a tener que bloquearse y luego volver elegido para ejecutar, sin embargo esto solamente es cierto si nuestro sistema tiene más de 1 CPU.
5. Cant. instancias: R1 = 1; R2 = 1

P1	P2
pedir_recurso(R2) pedir_recurso(R1) utilizar_recursos() liberar_recurso(R1) liberar_recurso(R2)	pedir_recurso(R1) pedir_recurso(R2) utilizar_recursos() liberar_recurso(R1) liberar_recurso(R2)

   Si ejecuta de manera contigua cualquiera de los 2 no habría deadlock (no hay retención y espera), sin embargo si ejecutaran de forma alternada de a un pedido quedarían en deadlock con P2 esperando por R2 y P1 esperando por R1.

Práctica

1. a)



KA			KB	KB	KA	KA							KA	KA	KA	KB	KB				KA	KA	KB					KB		
				KA											KB							KB								

			KA (1)				KB(1)	KB(1)	KB(1)				KB(2)								KB(1)			KA (2)	KA (2)	KA (2)				

b) Syscalls:

Creación de proceso/KLT: 0-3
Iniciar I/O: 2-6-9-11-18-20-23-26
Finalización de procesos: 28-29

Interrupciones:

Fin de Q: 4-10-15-22

c) Si ambas bibliotecas de ULTs implementaran Jacketing, en el instante 2, cuando UA2 comienza su I/O, el hilo UA1 podría ejecutar una unidad de tiempo de su ráfaga de CPU.

2.

hilo_main:
```
crear_hilo(hilo_1);
WAIT(hilo2_creado)
crear_hilo(hilo_3);
SIGNAL(hilo3_creado)
WAIT(finalizar)
finalizar_programa();
```

Hilo_1:
```
crear_hilo(hilo_2);
SIGNAL(hilo2_creado)
WAIT(hilo3_creado)
crear_hilo(hilo_4);
//termina el hilo
```

hilo_2:
```
while(1){
   WAIT(máximo)
   WAIT(mutex_contador)
   contador++;
   SIGNAL(mutex_contador)
   SIGNAL(semContador)
}
```

hilo_4:
```
while(1){
   WAIT(máximo)
   WAIT(mutex_contador)
   contador++;
   SIGNAL(mutex_contador)
   SIGNAL(semContador)
}
```

Hilo_3:
```
WAIT(semContador) x 100
printf("Contador llegó a 100");
SIGNAL(finalizar)
//termina el hilo
```

hilo2_creado = 0 hilo3_creado = 0
mutex_contador = 1
máximo = 100 semContador= 0
finalizar = 0

3.

a)

Elimino del análisis a P3 por no tener recursos asignados.
Recursos Disponibles: 0 1 0 1
Con estos recursos se puede finalizar P2 -> Solicitud: 0 1 0 1 Asignados: 0 2 2 1
Actualizo Disponibles: 0 3 2 2
No es posible finalizar ninguno de los demás procesos:

P1 -> Solicitud: 1 3 3 2 P4 -> Solicitud: 1 3 5 2 P5 -> Solicitud: 3 0 3 0 ,

Por lo tanto P1, P4 y P5 se encuentran en Deadlock y P3 en inanición.

Debo eliminar un proceso para salir del Deadlock, seleccionando aquel proceso con mayor cantidad de recursos asignados:

P1 -> Asignados: 2 1 2 1 = 6
P4 -> Asignados: 3 0 2 1 = 6
P5 -> Asignados: 1 2 0 2 = 5

Si finalizo P1:
Recursos Disponibles: 0 3 2 2
Actualizo Disponibles: 2 4 4 3
No puedo finalizar ni P4 (falta 1 R3) ni P5 (falta 1 R1)

Si finalizo P4:
Recursos Disponibles: 0 3 2 2
Actualizo Disponibles: 3 3 4 3

Finaliza P5 -> Recursos Disponibles: 4 5 4 5
Finaliza P1 -> Recursos Disponibles: 6 6 6 6 -> Recursos Totales

b)

Recursos totales: 7 6 7 6
Recursos disponibles: 1 1 1 1
Finaliza P2 -> Recursos Disponibles: 1 3 3 2
Finaliza P1 -> Recursos Disponibles: 3 4 5 3
Finaliza P4 -> Recursos Disponibles: 6 4 7 4
Finaliza P5 -> Recursos Disponibles: 7 6 7 6 -> Recursos Totales

Por lo tanto, si el sistema tuviese una instancia más de R1 y R3 no habría deadlock.