Sistemas Operativos
1° Parcial 2C2023 - 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.

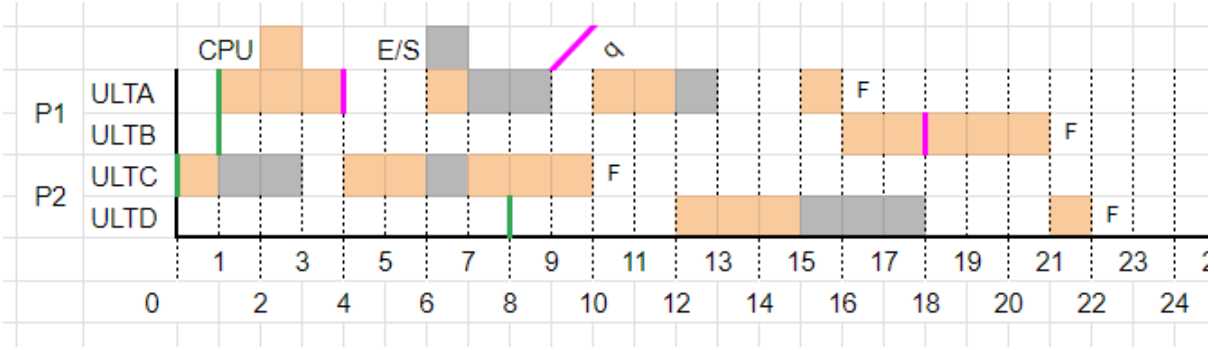
Teoría

1. La instrucción del proceso que se está ejecutando finalizará y luego, cómo están habilitadas las interrupciones, se atenderá la primera. Para ello, el contexto del proceso es apilado en el stack del sistema y el contexto de la interrupción es cargado en el procesador. Se comenzarán a ejecutar las instrucciones correspondientes a la rutina de la primera interrupción y, cuando llegue la segunda, ocurrirán eventos similares. El contexto de la rutina de la primera interrupción será apilado en el stack del sistema y se procederá a atender la segunda interrupción. Cuando su rutina finalice, se retomará la ejecución de la primera, y al finalizar esta, continuará ejecutando el proceso.
   Es posible que se dé un cambio de proceso luego de atender las interrupciones, si el planificador de corto plazo planifica utilizando desalojo.
2. Al crear un proceso, se crea su PCB que contiene al menos su identificador, contador de programa y otros registros del CPU e información del estado de proceso y de
   Sería posible que cualquier hilo del proceso cree un nuevo hilo y todos tendrían la misma jerarquía.
3. Utilizar algoritmos de planificación con desalojo le permite al Sistema Operativo tener más control de los procesos que están ejecutando en la CPU, ya que los mismos pueden replanificar cuando un proceso llega a Ready, permitiendo que los procesos que son más prioritarios accedan más rápido a la CPU y evitando la monopolización de la misma por procesos CPU Bound con baja prioridad. La principal desventaja es que, al tener en cuenta más eventos, el overhead es mayor, ya que la cantidad de decisiones que tiene que tomar es más grande.
   Sí, un algoritmo con desalojo podría producir inanición de todos modos. SJF con desalojo es un ejemplo de esto, ya que el algoritmo podría elegir constantemente otros procesos con ráfagas más cortas por sobre uno con ráfagas largas y el mismo podría nunca acceder a la CPU.
4. a) Verdadero, el uso de semaforos (tipo mutex) para proteger recursos compartidos podría generar Deadlock si por ejemplo dos hilos solicitan los mismos recursos pero de manera cruzada, ya que no se está garantizando una sincronización en el uso del recurso sino la consistencia del mismo.
   b) Falso, depende del tipo de recurso que se esté leyendo y principalmente si este al leerse se consume o no, siendo verdadero para el primer caso dado que al consumirse el resto de los hilos no lo tendrán disponible (ejemplo lista compartida), a diferencia de si el recurso no se consume, su lectura concurrente no se ve afectada (ejemplo var. global).

5. Livelock: es una situación (problema) donde múltiples procesos/hilos cambian continuamente sus estados en respuesta a cambios en otros procesos sin realizar ningún trabajo útil mientras tanto, generando mucho overhead, por otro lado Deadlock es una situación en la que varios procesos se bloquean entre sí debido a que estos tienen recursos asignados que está solicitando el otro proceso para poder avanzar.
   Prevención sería idóneo para un sistema donde necesite no caer nunca en deadlock, tener bajo overhead y pueda evitar alguna de sus condiciones, Evasión es útil para sistemas en los cuales el overhead no sea un factor determinante y necesite evitar el deadlock, mientras que la estrategia de Detección y Recupero sería candidato para sistemas donde la ocurrencia de deadlock no sea crítica pero se requiera tratarlo, por otro lado No hacer nada es una opción viable para sistemas donde la ocurrencia o no de esta problemática no sea importante al igual que su tratamiento.

Práctica

1. .a).



b)
i) Interrupciones de reloj:
Instante t=4: el proceso P1 es desalojado y comienza a ejecutar el P2
Instante t=10: el proceso P2 es desalojado y comienza a ejecutar el P1. Si el quantum hubiese sido mayor podría haber seguido ejecutando ULTD.
Instante t=18: el proceso P1 es desalojado y comienza a ejecutar el P2. El quantum es asignado al proceso y no a cada hilo de usuario.

ii) ULTB sufre inanición ya que inicia en el mismo instante que ULT1, pero debido al algoritmo utilizado tiene baja prioridad.
ULTB tiene una ráfaga de 5 ut. Pero las ráfagas de ULTA son todas menores. En los instantes t=1, t=6, t=10, t=15 el algoritmo de la biblioteca de hilos prioriza a ULTA.
En caso de utilizar la técnica de jacketing. Al bloquearse el ULTA, podría ejecutar ULTB a partir del instante t=7, cuando ULTA se bloquea y al P1 le quedan 2 ut de quantum.

2.

Solucion a)
PR = C = 1; UR[N] = {2, 2, ...}, E = 0;

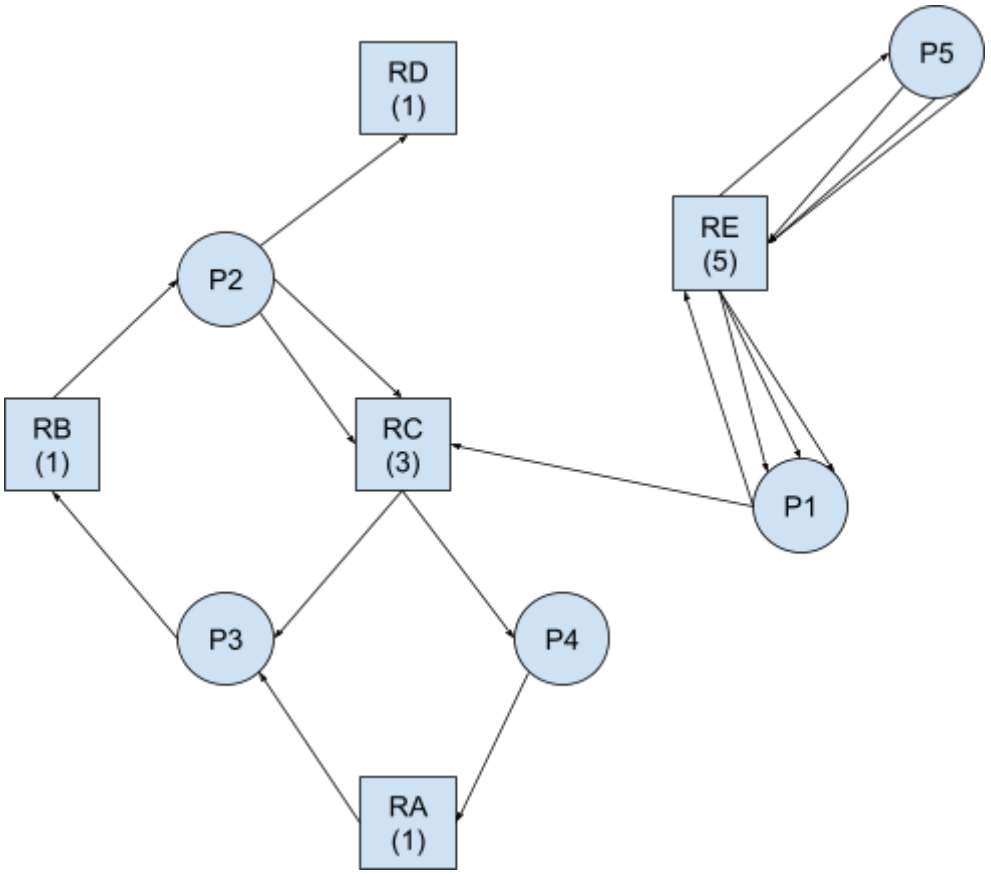
Productor (N instancias)	**Consumidor** (1 instancia)
WAIT(PR) id = pedir_recurso() SIGNAL(PR) WAIT(UR[id]) resultado = usar_recurso(id) SIGNAL(UR[id]) WAIT(C) depositar(resultado, cola) SIGNAL(C) SIGNAL(E)	WAIT(E) WAIT(C) resultado = retirar(cola) SIGNAL(C) imprimir(resultado)

Solucion b)
PR = C = 1; ~~UR[N] = {2, 2, ...}~~, E = 0;

Productor (N instancias)	**Consumidor** (1 instancia)
WAIT(PR) id = pedir_recurso() resultado = usar_recurso(id) SIGNAL(PR) WAIT(C) depositar(resultado, cola) SIGNAL(C) SIGNAL(E)	WAIT(E) WAIT(C) resultado = retirar(cola) SIGNAL(C) imprimir(resultado)

3.
a)
Dado el vector de recursos disponibles {0, 0, 1, 1, 1}:
P1 puede finalizar -> Libera Recursos -> Nuevos disponibles: 0, 0, 1, 1, 4
P5 puede finalizar -> Libera Recursos -> Nuevos disponibles: 0, 0, 1, 1, 5

Ni P2 ni P3 ni P4 pueden finalizar -> Deadlock.



b)
Lo más eficiente sería desalojarle 1 RA a P3. De esa manera:
Disponibles {1, 0, 1, 1, 5}

P4 puede finalizar -> Libera Recursos -> Nuevos disponibles: 1, 0, 2, 1, 5
P2 puede finalizar -> Libera Recursos -> Nuevos disponibles: 1, 1, 2, 1, 5
P3 puede finalizar.