**1) Una condición de carrera (Race Condition) ocurre cuando, independientemente del orden de ejecución de dos o más procesos, el recurso compartido queda siempre en un estado inconsistente.**

Falso, la condición de carrera es una situación en la cual múltiples hilos o procesos leen y escriben un dato compartido y el resultado final depende del orden de la ejecución de instrucciones en los múltiples procesos.

**2) La diferencia fundamental entre competencia y cooperación de procesos es que en la primera la sincronización es realizada solamente por los procesos, mientras que en la segunda necesariamente debe participar el sistema operativo.**

Falso, Es al revés. En la competencia, la sincronización es realizada por el sistema operativo, mientras que en la cooperación, es realizada por los procesos.

**4) Una falla en el mecanismo de sincronización de procesos concurrentes puede generar el fenómeno de inversión de prioridades.**

Verdadero ya que si un proceso de menor prioridad lockea un recurso necesario por ambos procesos el proceso de mayor prioridad no podrá continuar hasta que el otro haya finalizado.

**5) Sistemas que utilizan soluciones hardware para garantizar la mutua exclusión pueden presentar el problema de inversión de prioridades.**

Verdadero, las soluciones por hardware hacen que los procesos se amontonen cuando no pueden entrar a la sección critica, cuando un proceso puede entrar a esta la selección es arbitraria.

**6) La instrucción Test and Set, no sujeta a interrupción, como una solución hardware al problema de la exclusión mutua se define como sigue: La instrucción comprueba el valor de su argumento i. Si el valor es 1, entonces la instrucción reemplaza el valor por 0 y devuelve cierto. En caso contrario, el valor no se cambia y devuelve falso.**

Falso. Si el valor es 0, entonces la instrucción reemplaza el valor por 1 y devuelve cierto. En caso contrario, el valor no se cambia y devuelve falso. Stallings Pág. 213

**7) La gran desventaja que tienen las instrucciones TSL (test and set instructions) es que no pueden trabajar en sistemas con múltiples procesadores, lo que si pueden ser realizado por los semáforos.**

Falso,The use of a special machine instruction to enforce mutual exclusion has a number of advantages:

• It is applicable to any number of processes on either a single processor or multiple

processors sharing main memory.

• It is simple and therefore easy to verify.

• It can be used to support multiple critical sections; each critical section can be

defined by its own variable.

Stallings. 8th p213.

**8) El algoritmo de Peterson es una estrategia para la sincronización de procesos.**

Falso solo garantiza la exclusión mutua.

**9) Si sincronizo KLTS o procesos mediante semáforos y por lo menos tantas pruebas de escritorio como cantidad de semáforos se han colocado, funcionan correctamente, se puede inferir que la sincronización está bien realizada y funcionará correctamente.**

Falso, porque depende del orden en que se ejecuten los procesos.

**11) La sincronización de procesos mediante semáforos permite liberar al procesador más rápidamente cuando un proceso no consigue un recurso.**

Verdadero. Porque cuando sea hace un wait y no se consigue el recurso se bloquea el proceso permitiendo atender a otro proceso.

**12) El pasaje de mensajes es útil para la comunicación entre procesos pero necesita de los semáforos para realizar la mutua exclusión.**

Falso. El pasaje de mensajes es suficiente para la mutua exclusión; y no necesita el uso de semáforos.

**13) Los semáforos son herramientas que se emplean únicamente para resolver el problema de la exclusión mutua y para ello, basta con implementar un semáforo MUTEX.**

Falso, también existen semáforos contadores y binarios, se utilizan adecuadamente a los problemas de sincronizacion que requiera el programa que se quiere sincronizar.

**14) En los semáforos sin espera activa se utiliza una cola asociada para los procesos que esperan. Esto puede provocar en algunos procesos starvation, porque el algoritmo para dar de baja en la cola no necesariamente tiene que ser de tipo FIFO.**

Falso. los semáforos tienen una cola asociada para los procesos que esperan. Un semáforo que no especifica el orden en que los procesos son extraídos de la cola es un semáforo débil. Los semáforos que utilizan FIFO para extraer procesos de la cola de bloqueados se conocen como semáforos fuertes.[Stallings, 5ª ed. Esp., Pg. 218].

**15) Un semáforo es una función protegida por el sistema operativo que permite sincronizar el acceso a un recurso dentro de un ambiente multiprogramado.**

Falso, un semáforo no es una función es una variable.

**16) Si se tienen dos procesos concurrentes que utilizan algún mecanismo de sincronización, entonces no pueden estar simultáneamente dentro de una región crítica.**

Falso, podrían estar dentro de una región crítica si se sincronizó con semáforos contadores.

**17) Cuando un proceso quiere ingresar a una variable compartida, debe ejecutar un semáforo para preservar la mutua exclusión.**

Falso. Los semáforos no se ejecutan, lo que se ejecutan son instrucciones P/down/wait() o V/up/signal() sobre los semáforos.

Monitores

**19) Los monitores son una colección de procedimientos, variables y estructuras de datos que se agrupan en un cierto tipo de módulo particular y garantizan la exclusión mutua por si sola.**

Verdadero, por definición.

**20) Si un proceso, ejecutando código de un procedure de un monitor, ejecuta una operación WAIT sobre una variable condición, el proceso espera “dentro” del monitor en una cola de procesos bloqueados asociada a dicha variable**

Falso. Si el proceso queda bloqueado pero como “ejecutando” dentro de un monitor, ningún otro proceso podrá acceder a la variable condición (sobre la cual el proceso original hizo el WAIT) para hacer el SIGNAL correspondiente que despierte al proceso dormido. Recordar que cualquier acceso a una variable condición definida en un monitor DEBE hacerse ejecutando código del mismo, y que solamente un único proceso puede figurar como “ejecutando” código del monitor en un momento dado (por la exclusión mutua que estas construcciones proveen); cualquier otro proceso que haya invocado al monitor se bloquea en espera de que quede disponible.

**21) Los monitores son un mecanismo más fácil de usar que los semáforos, aunque tienen la desventaja de que sólo sirven para garantizar la mutua exclusión.**

Verdadero, sólo un proceso se puede estar ejecutando dentro del monitor al mismo tiempo, cualquier otro proceso que haya invocado al monitor se bloquea, en espera que el monitor quede disponible.

**22) Si dentro de un monitor se tienen varios procesos suspendidos en una condición “X” y un proceso ejecuta un signal() en esa condición, el orden de ejecución del próximo proceso suspendido es LIFO.**

Falso. Es FIFO. [Silberchatz-Galvin, 7ª ed. Esp., Pg. 191].

**23) En monitores con notify y broadcast, si el proceso que ejecuta un signal no termina su ejecución dentro del monitor, entonces son necesarios dos cambios de procesos (processes switch).**

Verdadero, ocurre cuando cambia el modo de usuario a kernel y de kernel a usuario.

**24) A pesar de implementar monitores como mecanismo de sincronización, en Java es posible implementar una solución del problema de los filósofos.**

Verdadero, con los monitores también se puede sincronizar los procesos. Los monitores garantizan la sincronización necesaria para la resolución del problema de los filósofos.

**25) Cuando un proceso que se encuentra dentro de un monitor realiza una operación WAIT sobre una variable de condición, el proceso espera fuera del monitor en una cola de procesos bloqueados asociada a dicha variable de condición.**

Falso, espera dentro del monitor por esa condición. En monitores la operación WAIT no existe, sino que la primitiva se llama CWAIT.

Espera Activa

**26) El término espera activa (busy waiting) se refiere a que un proceso no puede hacer nada o ejecutar hasta obtener permiso para entrar en su región crítica.**

Falso. Busy Waiting (espera activa) se refiere a que el proceso entra en un ciclo (while, for) preguntando por una condición y hasta que no se cumpla esa condición el proceso no sale del ciclo. [Stallings, 5ª ed. Esp., Pg. 213].

**27) El Bussy Waiting se produce siempre que un proceso ejecuta un ciclo y tiene como desventaja que empeora la performance del sistema.**

Falso, un proceso con espera activa empeora la performance del sistema porque usa tiempo de procesador para esperar un recurso.

Inanición

**28) La técnica para evitar la Inanición (Starvation) es matar al proceso que no consigue el recurso.**

Falso. Un proceso que sufre de inanición precisamente necesita que se le asigne el recurso que espera, si eliminamos ese proceso se termina la inanición de éste a un costo muy alto, ya que el proceso nunca habrá concluido su trabajo satisfactoriamente (muerte por inanición). La inanición se puede evitar mediante una correcta sincronización de procesos que compiten por recursos críticos. Ahora cuando se produce un deadlock, eliminar procesos que intervienen en él si es una alternativa (pero no es una técnica para evitarlo, sino para recuperarse de un deadlock). Otra alternativa de recuperación es la apropiación de recursos, desalojar en forma sucesiva recursos de proceso y asignarlos a otros hasta que se interrumpe el deadlock.[Stallings, 5ª ed. Esp., Pgs. 203, 209].

**29) Únicamente se produce starvation por el uso inadecuado de herramientas de sincronización.**

Falso, también puede ocurrir por una mala planificación. Puede producirse starvation en la utilización de herramientas de planificación. Por ejemplo en la utilización de SPN (Shortest Process Next) en la planificación de procesos, puede, en algunos casos, producirse starvation con los procesos largos.

Atomicidad

**31) Que ciertas instrucciones se ejecuten de forma atómica significa que no se pueden ejecutar concurrentemente.**

Falso, significa que es una operación que no es interrumpida hasta que finaliza su procesamiento, se podrían ejecutar concurrentemente en distintos procesadores.

**32) La Mutua Exclusión de dos secciones críticas asegura que éstas se ejecuten atómicamente en forma concurrente.**

Falso,la mutua exclusion asegura que un proceso pueda ejecutar una seccion critica sin que otro proceso la acceda.

Sistemas Multiprocesador

**33) Es deseable que todos los procesos (o al menos de misma prioridad) tengan la misma probabilidad de adquirir recursos.**

Verdadero, si un proceso tuviese una probabilidad muy baja de adquirir un recurso y a su vez este está reteniendo muchos otros recursos que son requeridos por otros procesos es poco deseable porque tardaría mucho tiempo para liberar sus recursos.

**35) En un sistema multiprocesador, deshabilitar las interrupciones de un procesador es suficiente para solucionar el problema de la sección crítica.**

Falso, ya que si se deshabilitan las interrupciones, otro procesador puede estar ejecutando en la sección critica que se intenta entrar.

**36) La inhabilitación de interrupciones como mecanismo para garantizar la exclusión mutua a una sección crítica tiene sentido en Sistemas Operativos con multiprocesamiento.**

Falso, ya que si se deshabilitan las interrupciones, otro procesador puede estar ejecutando en la sección critica que se intenta entrar.

**37) En un sistema multiprocesador, deshabilitar las interrupciones no garantiza la mutua exclusión.**

Verdadero. Ya que si se deshabilitan las interrupciones de un procesador, otro procesador puede estar ejecutando en la sección critica que se intenta entrar.

Deadlock

**38) Los conflictos involucrados en un Deadlock se deben a 2 o más procesos y no a la necesidad de recursos que necesitan esos procesos**

Falso, se debe a los recursos que necesitan los procesos. [Stallings, 5ª ed. Esp., Pg. 266].

**39) En un Sistema que contiene una única instancia de cada recurso, la espera circular es una condición necesaria y suficiente para la existencia de Deadlock.**

Verdadero, la existencia de espera circular siempre es una condición necesaria y suficiente para que exista deadlock

**41) Una de las formas de prevención del deadlock es eliminar la condición de mutua exclusión, sobre los recursos.**

Verdadero porque al compartir los recursos se dejaría de cumplir una de las 4 condiciones necesarias para la existencia de deadlock.

**43) Si en un conjunto de procesos existe una espera circular es probable que existe o se pueda producir un deadlock.**

Verdadero, es condición necesaria y suficiente

**45) Una forma de detectar deadlock, es detectar uno o más procesos con pedidos insatisfechos de recursos.**

Falso, hay que utilizar el algoritmo de detección (stallings 8th 276)

**46) Un Livelock es más fácil de detectar que un Deadlock.**

Falso porque en el livelock los procesos siguen ejecutándose.

**47) La forma más sencilla de prevenir un deadlock es lograr que se elimine la condición de retención y espera en donde el proceso antes de iniciar la ejecución pide todos los recursos que va a necesitar durante todo el tiempo que este ejecutando.**

Falso, eliminar la espera circular: es la más fácil de atacar. Se le permite a un proceso poseer sólo un recurso en un determinado momento, o una jerarquía puede ser impuesta de modo tal que los ciclos de espera no sean posibles

**48) Una técnica para evitar (avoidance) que se produzcan deadlocks es matar uno de los procesos que intervienen en el deadlock.**

Falso, matar un proceso es una tecnica de una recuperacion de deadlock, no es una técnica de evasion.

Algoritmo del Banquero

**50) Usando el algoritmo del banquero se puede prevenir la ocurrencia de deadlock independientemente de que se produzcan o no las cuatro condiciones necesarias para la existencia del mismo.**

Falso, el algoritmo del banquero evita el deadlock y un deadlock solo se puede dar si se produjeron las cuatro condiciones necesarias.

**51) Un estado seguro es aquel estado en el que por lo menos un proceso se puede ejecutar en su totalidad.**

Falso, Un estado seguro es uno en el cual existe una forma de asignar los recursos de manera que todos los procesos puedan ejecutar en su totalidad. (STL. 176) -> "Un estado seguro es uno en el cual hay al menos una secuencia que no resulta en deadlock".

**52) Si el estado de un sistema es un estado seguro, entonces no puede haber espera circular.**

Falso, El estado de un sistema es seguro si hay una secuencia de ejecución que permita terminar todos los procesos, independientemente si hay una espera circular o no, entonces en un estado seguro se puede dar una espera circular momentáneamente siempre que no se den las tres condiciones restantes (Coffman). Una espera circular por sí sola no es condición suficiente.

Algoritmo de detección de Deadlock

**53) Si después de correr el algoritmo de detección de deadlock observamos que no hay ningún tipo de interbloqueo podemos asegurar que el sistema está en estado seguro.**

Verdadero, porque se puede asignar los recursos a los procesos sin que se produzca deadlock.

**55) Si en un SO no existe ninguna cola de bloqueados, entonces está libre de deadlocks.**

VERDADERO en cierto contexto: En un Modelo de Proceso de dos estados [Stallings, 5ed, p113] existen los estados ejecutando y no ejecutando. En sistemas operativos monoprogramados, que solo pueden ejecutar un único proceso “simultáneamente”, un proceso pide una E/S interactuando directamente con el dispositivo, sin que intervenga el SO. No existe una cola de bloqueados. Al ser un único proceso en memoria, no existen deadlocks.

**56) Desde el punto de vista de la evitación del deadlock, si un sistema está en estado seguro, cualquier solicitud de recursos que efectúe un proceso puede satisfacerse y dejar nuevamente al sistema en estado seguro.**

Falso, puede ser que no pueda satisfacerla y por lo tanto no le asigna los recursos.

**57) En sistemas de tiempo real, la mejor implementación para gestionar los inconvenientes de deadlock es el algoritmo del avestruz, debido al bajo costo que acarrea su solución.**

Falso, el algoritmo de avestruz no es una solución, es justamente no hacer nada para evitar problema del deadlock.

**58) Solo puede haber Interbloqueos entre threads del mismo proceso.**

Falso puede haber deadlock entre threads de distintos procesos que necesiten un mismo recurso.

**59) La evasión de interbloqueos requiere que se proporcione de antemano al sistema operativo información adicional sobre qué recursos solicitará y utilizará un proceso durante su tiempo de vida.**

Verdadero, porque se necesita determinar una secuencia de ejecución de los procesos para que no termine en deadlock.