**1) Un proceso es una entidad estática formada por los datos, código, pila y su contexto de ejecución.**

Falso, no se puede decir que es una entidad estática dado que la información referente a este no es inmutable. (en ninguno de los libros se cita a un proceso como una entidad estatica/dinamica, habria que preguntar si se refiere estatica en cuanto a la estructura de la informacion que maneja el so, o si a la informacion respecto al proceso y su estado)

**2) Si un proceso efectúa una llamada al sistema bloqueante, entonces el proceso es bloqueado y enviado a la cola de BLOCKED.**  
Falso, contraejemplo: un proceso multihilo con klt con un planificador global en fifo, si se bloquea uno de los klt por io por ej, puede seguir ejecutando otro klt del mismo proceso y sigue en execute.

**3) Al hablar de procesos, se puede afirmar que la E/S no bloqueante plantea un modelo de programación más complejo y propenso a errores que la E/S bloqueante.**

Verdadero, La E/S no bloqueante requiere de algun mecanismo de sincronizacion.

**4) A pesar que el procesador sólo puede ejecutar una instrucción en cada instante de tiempo, gracias a la multiprogramación, es posible que dos procesos se completen en un tiempo menor que si se ejecutaran de forma secuencial, incluso en un equipo con un solo procesador.**

Verdadero, With interrupt-driven I/O or DMA, the processor can issue an I/O command for one job and proceed with the execution of another job while the I/O is carried out by the device controller. <- lo que hace ganar un monton de tiempo en la planificacion

(Stallings 8th pag 58)

**5) El PCB (Process Control Block) es mantenido en un almacenamiento secundario (generalmente disco) además de la localización en Memoria Central.**

Falso, el PCB es parte de la imagen del proceso. "La posición de la imagen dependerá del esquema de gestión de memoria" [Stallings pág.128]

**6) Un proceso está formado por los siguientes tres componentes: porción de un programa ejecutable, datos asociados que necesita el programa (variables, espacio de trabajo, buffers, etc.) el contexto de ejecución del programa y de estos, el componente más esencial son los datos asociados que necesita el programa.**

Falso. La más esencial es el contexto de ejecución. Que es el conjunto de datos interno por el cual el S.O. es capaz de supervisar y controlar el proceso. [Stallings pág. 69]

**7) Aceptando que dentro del PCB encontramos información respecto a tres categorías: Identificación del proceso, Información del estado del procesador e Información de control de proceso; los registros de códigos de condición (signo, cero, acarreo, igual, etc.) se encuentran dentro de la categoría Información del estado del procesador.**

Verdadero. En la informacion de estado del proceso se contienen datos acerca de registros de usuario, de control o estado, siendo los registros de condicion pertenecientes a los registros de control o estado . (Stallings 8th pag 130)

**8) Durante la ejecución de los procesos puede ocurrir que estos alternen entre CPU burst e I/O burst, pudiendo terminar normalmente el proceso durante cualquiera de ellos.**

Falso, un proceso que termina en forma correcta debe terminar con un CPU burst sino el so nunca se entera de que el proceso termino..

Process execution begins with a CPU burst. That is followed by an I/O burst ,which is followed by another CPU burst, then another I/O burst, and so on. Eventually, the final CPU burst ends with a system request to terminate execution (silberschatz 9th p262)

**9) En un sistema con un diagrama de procesos de siete estados que utiliza memoria virtual se elimina la necesidad de usar Swapping Explícito para enviar los procesos de Bloqueado a Suspendido.**

Falso. Incluso con un sistema de Memoria Virtual el SO necesita hacer swapping de los procesos de forma explícita y completa, de vez en cuando, con la intención de mejorar el rendimiento. [Stallings, 5ª ed. Esp., Pg. 122].

“El swapping de procesos es independiente del manejo de memoria virtual.”

**10) El cambio de proceso (Process Switch) puede ocurrir al tiempo que el SO ha ganado el control desde el proceso en ejecución.**  
Verdadero. “Un cambio de proceso puede ocurrir en cualquier momento en el que el SO obtiene el control sobre el proceso actualmente en ejecución”. [Stallings, 5ª ed. Esp., Pg. 138].

**11) El cambio de contexto es una operación que realiza el propio hardware.**

Falso, es el SO quien realiza los cambios de contexto.

When a context switch occurs,the kernel saves the context

of the old process in its PCB and loads the saved context of the new process scheduled to run. (silber 9th p114)

**12) Si procesos estrechamente relacionados ejecutan en paralelo (en distintos procesadores), el bloqueo por sincronización puede llegar a ser alto, menos process switching pueden llegar a ser necesarios, y la performance se incrementará.**

Falso. Si el bloqueo por sincronización es alto, la performance disminuye.

**13) En un sistema con un diagrama de 7 estados, un proceso podría pasar directamente del estado Ready al estado Exit.**

Verdadero. el usuario o el proceso padre o el sistema operativo decide finalizar el proceso mientras este se encontraba en la cola de ready.

**14) Un proceso puede pasar del estado de Bloqueados al estado de Finalizados.**

Verdadero. idem anterior pero en la cola de block

**15) Solo cuando un proceso termina puede devolver un valor de estado a su padre, pidiendo que lo elimine.**

Verdadero. "Un proceso termina cuando ejecuta su última instrucción y pide al sistema operativo que lo elimine usando la llamada al sistema exit(). En ese momento, el proceso puede devolver un valor de estado a su proceso padre (a través de la llamada al sistema wait() )". [Silberchatz-Galvin, 7ª ed. Esp., Pg. 85].

ULT/KLT

**16) Los procesos IO-Bundle (orientados al uso de I/O) son los más eficientes para dividir en múltiples procesos livianos.**

Verdadero. Ya que puede haber muchos hilos que realicen I/Os diferentes simultáneamente y bloquearse solamente ellos (si trabajamos con KLT, con ULT recordemos que se bloquea todo el proceso y esta afirmación sería falsa), permitiendo que corran otros hilos del proceso listos para ejecutar.

**17) Al usar ULTs combinados con KLTs, las syscalls bloqueantes desde un ULT no bloquean al resto de los UTLs**

Verdadero. The multiple ULTs from a single application are

mapped onto some (smaller or equal) number of KLTs. The programmer may adjust the number of KLTs for a particular application and processor to achieve the best overall results. In a combined approach, multiple threads within the same application can run in parallel on multiple processors, and a blocking system call need not block the entire process(stallings 8th p 169)

**18) Los UTL tienen la ventaja de que implican un menor overhead de planificación en comparación con los KLT.**

Verdadero. Thread switching does not require kernel mode privileges because all of the thread management data structures are within the user address space of a single process. Therefore, the process does not switch to the kernel mode to do thread management. This saves the overhead of two mode switches (user to kernel; kernel back to user). (stallings 8th p 167)

**19) Una ventaja de los ULTs es que el algoritmo de planificación puede ser propio de cada aplicación, pero como contrapartida debe generar una llamada al sistema por cada cambio de hilo en ejecución, lo que provoca overhead al cambiar de modo usuario a modo kernel.**

Falso, los ult son creados en el espacio de usuario asi que no implican nunca un mode switch.

**20) Los hilos a nivel usuario son creados, eliminados y planificados por la biblioteca de hilos. Para realizar el cambio de contexto de los hilos, la biblioteca cambia de modo, salva el contexto del hilo saliente y del control a otro hilo que está listo.**

Falso, no necesita cambiar a modo kernel. The threads library creates a data structure for the new thread and then passes control to one of the threads within this process that is in the Ready state, using some scheduling algorithm. When control is passed to the library, the context of the current thread is saved, and when control is passed from the library to a thread, the context of that thread

is restored. (stallings 8th 165)

**21) En un proceso con dos o más ULT, si el que está en ejecución solicita una I/O bloqueante, cuando vuelva a ejecutar el proceso, podría seguir ejecutando el mismo hilo, independientemente del algoritmo de planificación de dicha biblioteca.**

Como el SO no se entera de los ULT, el algoritmo de planificación es a KLT, entonces, al volver, sigue con el mismo hilo, excepto que el programador indique lo contrario (recordar que como el SO no los ve, el orden de ejecución y planificación dependen, realmente, de lo que haga el programador)

**22) Cuando en un proceso se usan Threads siempre se necesita la estructura PCB u otra similar, independientemente de que sean KLT o ULTs.**

Verdadera, un proceso siempre utiliza un PCB, y si hay hilos cada hilo tiene su TCB.

**24) En el caso de los ULT (User Level Threads) el código para crear y destruir los threads son rutinas del Kernel.**

Falso, son rutinas de la biblioteca de threads. [Stallings, 5ª ed. Esp., Pg. 166].

**25) Entre las categorías de implementación de hilos ULT y KLT, los KLT presentan respecto a los ULT la ventaja de que pueden ejecutar en cualquier sistema operativo.**

Falso. Al revés, una ventaja de las ULT respecto a los KLT es que pueden ejecutar en cualquier sistema operativo. [Stallings pág. 168]

**26) Una de las ventajas del uso de kernel level threads (KLTs) es que el cambio de threads (Thread switch) de un mismo proceso es más rápido que el cambio entre procesos (process switch) ya que los primeros comparten el stack (pila) del proceso.**

Falso, no es cierto que compartan la pilas, cada thread tiene su propia pila.(stalings 8th p160 fig4.2)

**29) Los Threads son más rápidos que los procesos pesados debido a que al procesador le cuesta menos tiempo una instrucción proveniente de un Thread que la de un proceso.**

Falso. Se debe a:

Toma menos tiempo crear un nuevo thread en un proceso existente, que crear un nuevo proceso.

Toma menos tiempo terminar un thread que un proceso.

El context switch entre threads de un mismo proceso es más rápido que el context switch entre procesos.

**30) Los procesos pesados ejecutan solo un programa por proceso, de la misma manera que los procesos ligeros ejecutan ese mismo programa entre varios.**

Falso, no hay relacion entre la cantidad de programas que se ejecutan con el tipo de proceso.

**33) El tiempo de creación de un Thread es mucho menor que el de un proceso pesado, debido a que la estructura TCB es más pequeña que el PCB.**

Falso. La asignación de memoria y recursos para la creación de procesos es costosa, mientras que la creación de hebras es más sencilla debido a que comparten recursos del proceso. [Silberchatz-Galvin, 7ª ed. Esp., Pg. 115].

**35) Los Threads son más rápido que los procesos pesados solo si se ejecutan en equipos con multiprocesadores o que el único procesador tenga más de un hilo de ejecución.**

Falso, son más rápidos porque tienen menor overhead de context switch.

Los procesadores tiene hilos de ejecución?

**36) Todo proceso tiene al menos un hilo.**

Verdadero, por default cuando se crea un proceso siempre tiene un hilo corriendo.

**37) Los hilos de un mismo proceso comparten las variables globales, el heap, la pila (stack) y los manejadores de señales (signal handlers), pero no comparten el contador de programa (Program Counter).**

Falso, no comparten el stack.

**38) Como el Sistema Operativo los desconoce, no tiene sentido sincronizar a los ULTs (User Level Threads), en cambio es fundamental sincronizar los KLTs (Kernel Level Threads).**

Falso, independientemente de si es ULT, KLT o un proceso, es necesario asegurar la sincronización en el acceso a los recursos críticos.

**39) En un sistema que cuenta con un solo procesador y dicho procesador no cuenta con múltiples hilos de ejecución no se obtienen mejoras al dividir los procesos pesados en Threads.**

Falso. Los procesadores nunca cuentan con múltiples hilos de ejecución, son los procesos los que pueden contar con varios hilos/threads (o no). Por otro lado, en sistemas con un solo procesador sí se obtienen mejoras al utilizar Threads, ya que por ejemplo mientras un hilo (o varios) permanece bloqueado a la espera de un E/S, otro puede estar ejecutando.

**40) Un proceso suspendido siempre está a la espera de un evento.**

Falso, un proceso puede estar en el estado listo/suspendido, y en este estado no está a la espera de un evento sino que está listo para ser llevado a memoria central.

Planificadores

**41) La existencia de un planificador a corto plazo sólo tiene sentido en los sistemas con multiprogramación.**

Falso. Pues el planificador a corto plazo puede ser necesario igualmente al momento de la suspensión del proceso. Por ejemplo Operating system calls, i/o interrupts or signals. (pag 398).

**42) ¿Cuál de los planificadores es el encargado de bajar la tasa de multiprogramación del sistema?**

Planificador a largo plazo. Éste es el planificador que hace lo indicado. El planificador a largo plazo, ya que determina qué programas se admiten en el sistema para su procesamiento (controla el grado de multiprogramación). [Stallings, 5ª ed. Esp., Pg. 403].

**44) El planificador de corto plazo (Short Term Scheduler) es invocado solamente cuando se produce una interrupción de reloj.**

Falso se invoca la ejecución del planificador de corto plazo para decidir qué proceso continuará en uso del procesador después de un context switch y esto ocurre frente a cualquier interrupción.

**45) La ejecución muy frecuente de los planificadores de corto y largo plazo es la principal característica de los mismos.**

Falso, el planificador largo plazo se ejecuta con mucha menos frecuencia que el de corto plazo.

**46) En un algoritmo de planificación de corto plazo sin desalojo (non-preemptive), no es posible que un proceso pase del estado RUNNING al estado READY**

Verdadero, porque si es sin desalojo se ejecuta hasta que termina o hasta que tenga que hacer una E/S y vaya a la cola de bloqueados, pero nunca puede ir a Ready directo.

**47) La siguiente afirmación sobre planificación de procesos “todos los métodos basados en prioridades tiene riesgo de inanición”.**

Falso. Existen algoritmos de planificación basados en prioridades que utilizan la técnica de aging para evitar la inanición, mediante la cual la prioridad de los procesos incrementa gradualmente con el tiempo.

**48) Se puede producir la inanición de un proceso cuando se use una planificación de procesos por prioridades fijas.**   
Verdadero, por ejemplo un proceso de prioridad muy baja puede sufrir inanición si constantemente entran al sistema procesos de prioridades altas.

**49) Referido al algoritmo de Round Robin para scheduling de CPU de corto plazo ¿Es posible que se presente inanición?**

Falso. Porque le da un quantum de tiempo a cada proceso equitativamente por turnos.

**50) La planificación RR (Round Robin) es particularmente efectiva en sistemas de tiempo compartido de propósito general y en sistemas de procesamiento transaccional.**

Verdadero. Extracción textual de [Stallings, 5ª ed. Esp., Pg. 413].

**51) En un Sistema que utiliza Round Robin como política de planificación de procesos, si existen procesos que ejecutan solamente operaciones atómicas entonces el Short Term Scheduler degenera en FCFS, ya que no se va a poder interrumpir el proceso en ejecución hasta que finalice.**

Falso, lo que no se puede interrumpir es la operación atómica, pero ni bien termina esa operación, el planificador a corto plazo evalúa si hay que cambiar de proceso.

**52) Una duración muy baja del quantum de tiempo en el algoritmo Round Robin puede llegar a producir una baja performance en el sistema.**

Verdadero, si el quantum de tiempo es bajo va a ser mayor el tiempo consumido en hacer los cambios de proceso que lo que se consume en procesar.

**53) El algoritmo de planificación de procesos SRT (Shortest Remaining Time) es el que menor Turnarround Time posee para cualquier proceso que se encuentre corriendo en el sistema.**

Falso, SRT beneficia a los procesos cortos y no a los largos por lo que puede haber inanición y el turn around de un proceso ser eterno.

**54) El algoritmo de planificación SJF (Shortest Job First) se puede implementar para la cola de nuevos, pero no para la cola de listos.**  
Verdadero, "Aunque el algoritmo SJF es óptimo no se puede implementar en el nivel de la planificación a corto plazo ya que no hay forma de conocer la duración de la siguiente ráfaga de CPU". [Silberchatz p.144]

**55) Cuando se produce un cambio de estado por una interrupción de E/S, este es una parte mínima del Quantum del proceso en ejecución.**

Falso, el tiempo que insume el process switch no se cuenta adentro del quantum sino aparte.

**56) Es imposible implementar Feedback con round robin en cada nivel en un sistema operativo tipo UNIX ya que se desconoce el largo de los procesos.**

Falso, para hacer el algoritmo feedback no se necesitan conocer el largo de cada proceso. Pues sí es posible aplicar feedback en los casos en que se desconoce el largo de los procesos.

**57) El algoritmo Feedback tiene como desventaja la necesidad de estimar la duración del burst de CPU.**

Falso, no necesita estimar la duración del burst del CPU porque puede utilizar prioridades fijas.

**58) La planificación por prioridades fijas no se puede implementar mediante múltiples colas multinivel con retroalimentación.**

Falso, en la planificación mediante colas multinivel realimentadas, cada cola puede tener una prioridad con respecto a la siguiente cola, pero al ser retroalimentadas permite mover un proceso de una cola a otra.

**59) En la planificación del procesador, el modelo de varias colas resuelve el problema de que el procesador esté la mayor parte del tiempo desocupado.**

Falso.

Lo minimiza pero no lo resuelve.

**60) En un sistema que trabaja con el algoritmo de planificación HRRN (high ratio response next) sólo los procesos nuevos tendrán el ratio igual a 1 (uno).**

Verdadero.

The minimum value of R is 1.0, which occurs when a process first enters the system.

*Stallings 8th 413*

**61) "En un Sistema Operativo PREEMPTIVE no se pueden hacer presupuestos sobre el tiempo de corrida de un proceso" (cita del libro de Tanenbaum). Es por este motivo que el Sistema Operativo es el que hace el context switch, dado que es el único que sabe cuándo termina el quantum de tiempo de un proceso.**

Falso.

Que un SO sea preemptive no implica que use planificación por RR o que use el concepto de quantum (hay otras planificaciones preemptive). Es decir que hay SO que planifican de forma preemptive, sin quantum y el SO sigue siendo el encargado de realizar el context switch. Finalmente, el hecho de que el SO conozca la duración del quantum no es el motivo por el cual es el encargado de realizar los context switch.

**62) El turnaround time promedio de un conjunto de procesos no necesariamente mejora si aumenta el tamaño del quantum.**   
Verdadero.

Turnaround time also depends on the size of the time quantum. As we can see from Figure 6.5, the average turnaround time of a set of processes does not necessarily improve as the time-quantum size increases. In general, the average turnaround time can be improved if most processes finish their next CPU burst in a single time quantum.

*Silberschatz 9th 273*

**63) El throughtput viene influenciado por la duración promedio de los procesos y a su vez por la política de planificación de procesos.**   
Verdadero.

Throughput The scheduling policy should attempt to maximize the number of processes completed per unit of time. This is a measure of how much work is being performed. This clearly depends on the average length of a process but is also influenced by the scheduling policy, which may affect utilization.

*Stallings 8th 401*

**64) Una de las funciones que realiza el dispatcher es el de guardar el estado del proceso actual en su PCB y restaurar el estado del proceso siguiente a ejecutar.**

Verdadero.

The kernel dispatcher provides the foundation for the executive and the subsystems. Most of the dispatcher is never paged out of memory, and its execution is never preempted. Its main responsibilities are thread scheduling and **context switching**, implementation of synchronization primitives, timer management, software interrupts (asynchronous and deferred procedure calls), and exception dispatching.

*Silberschatz 9th 839*

When a **context switch** occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.

*Silberschatz 9th 114*

**65) Dentro de los algoritmos de planificación de tiempo real aquellos que cuando llega una tarea, el sistema le asigna una prioridad basada en las características de la misma es conocido como: Planificación dinámica basada en un plan.**

Falso.

With **dynamic planning-based scheduling**, after a task arrives, but before its execution begins, an attempt is made to create a schedule that contains the previously scheduled tasks as well as the new arrival. If the new arrival can be scheduled in such a way that its deadlines are satisfied and that no currently scheduled task misses a deadline, then the schedule is revised to accommodate the new task.

**Dynamic best effort scheduling** is the approach used by many real-time systems that are currently commercially available. When a task arrives, the system assigns a priority based on the characteristics of the task.

*Stallings 8th 448*

**66) La política de planificación por grupos o Gang Scheduling reduce la sobrecarga de planificación, ya que una sola decisión afecta a varios procesadores y procesos.**

Verdadero.

Scheduling overhead may be reduced because a single decision affects a number of processors and processes at one time.

*Stallings 8th 439*