Respuestas Orientadoras e incompletas (en algunos casos) a las preguntas de revisión. Favor tomarlas como orientadoras.

Clase 1 – Revisión

1. ¿Cuáles son los elementos básicos de un computador?

-Procesador (CPU): controla el funcionamiento del computador, realiza funciones de procesamiento de datos, intercambia datos con la memoria principal.
-Memoria Principal: Almacena datos y programas, cojunto de ubicaciones secuenciales numeradas, volatil (cuando se apaga el computador se pierde su contenido) a diferencia de la memoria secundaria (disco rígido, por ejemplo) la cual mantiene su contenido aunque se apague el computador.
-Modulos E/S: transfieren datos entre el computador y su entorno externo. El entorno externo esta formado por dispositivos, de todo tipo, dispositivos de memoria secundaria (discos), de comunicaciones y terminales.
-Bus del Sistema: interconexion del sistema, comunica procesador con memoria principal y modos E/S.

2. ¿Qué es una interrupción?

Mecanismo por el cual otros componentes (ej. módulos E/S) pueden interrumpir la normal ejecución de la CPU.

3. ¿Cómo pueden tratarse múltiples interrupciones?

Pueden ocurrir n interrupciones al mismo tiempo.
Solución1: Inhabilitar interrupciones mientras se trata una, las que ocurran quedan pendientes y se tratarán luego de que termine la interrupcion actual. Solución elegante, pero no tiene en cuenta las prioridades o criticidad de las interrupciones (por ejemplo, las comunicaciones, que deben ser atendidas rápidamente para hacer lugar a la recepción de nuevos datos y evitar pérdida.
Solución 2: Manejar prioridades; una interrupción de prioridad más alta puede interrumpir a una de más baja prioridad.

4. ¿Qué es la memoria cache y qué posición ocupa en la jerarquía de memoria?

La memoria cache es una memoria de alta velocidad, cara, pequeña, rápida, invisible para el SO, que se aproxima mucho más que la memoria principal a la velocidad de los registros de la CPU. En la jerarquía de memoria se encuentra por encima de la memoria principal y por debajo de los registro de la CPU. Explotando *el principio de proximidad o proximidad de referencias* (durante la ejecucion de un programa, las referencias de memoria del procesador -tanto a instrucciones como a datos- tienden a agruparse. Debido a los bucles iterativos, rutinas, etc. Durante un período largo de tiempo, estas agrupaciones van cambiando, pero en un período corto, el procesador esta trabajando principalmente con un grupos fijos de referencias a memoria. Esto motiva al diseño de la cache.) se pueden copiar desde la memoria principal bloques de memoria en la memoria cache para que la CPU la acceda desde allí y de esta forma tener un mejor rendimiento.

5. ¿Qué relación hay entre el costo de memoria, su velocidad, su tamaño y la jerarquía de memoria?

Capacidad, Velocidad, Costo son interdependientes. A mayor velocidad, mayor costo, menor capacidad. A menor velocidad, menor costo, mayor capacidad.

6. ¿Por qué cree Ud. que se desarrollo la técnica de DMA? ¿Por qué no continuar sólo con el tratamiento de interrupciones?

Para mejorar el rendimiento, de esta forma, la CPU no necesita encargarse de la transferencia de datos desde el dispositivo a la memoria principal, sino que dicha tarea es delegada al modulo DMA quien realiza la tarea con una mínima intervención por parte de la CPU, la cual queda libre para ejecutar otros procesos.

7. Describa brevemente las tres técnicas para llevar a cabo operaciones de I/O.

-E-S programada: el módulo de E-S no lanza interrupciones sobre la CPU, sino que la CPU chequea periódicamente si se ha completado la operación de E-S para transferir los datos hacia/desde la memoria principal. Consume mucho tiempo (espera activa), ocupa la CPU en forma innecesaria.

-E-S dirigida por interrupciones: la CPU envía señal de E-S al módulo de E-S correspondiente y continúa con otra tarea. El módulo de E-S (en paralelo) realiza la operación de E-S y cuando ésta esta lista, envía la señal de interrupción a la CPU; ésta interrumpe -temporariamente- la ejecución del proceso actual, realiza la transferencia de datos y reanuda la ejecución del proceso interrumpido. Elimina espera innecesaria de la CPU, cada palabra a transferir del módulo E-S a la memoria principal debe pasar por la CPU.

-DMA: DMA puede ser un módulo separado, implementado sobre el bus del sistema o bien puede estar incorporado dentro de un módulo de E-S. Cuando la CPU necesita realizar una operación de E-S, señala al DMA indicando:

-Tipo de operación (read / write)
-Dirección del dispositivo de E-S
-Dirección inicial de memoria principal
-Número de palabras a escribir / leer

la CPU continúa con otra tarea. A partir de allí (en paralelo) el módulo DMA hace la operación de E-S en forma independiente, transfiere los datos y cuando la operación ha finalizado, envía señal de interrupción a la CPU. De esta forma, los datos no pasan a través de la CPU para ser transferidos desde/hacia la memoria principal.

Respuestas Orientadoras e incompletas (en algunos casos) a las preguntas de revisión. Favor tomarlas como orientadoras.

Clase 2 – Revisión

1. ¿Cómo funciona una pila o stack? ¿Cuál es su punto de acceso, su límite y su base? ¿Es una lista de tipo LIFO o FIFO?

Una pila es un conjunto ordenado de elementos, en donde el último elemento en entrar al conjunto es el primero en salir, se puede implementar como una lista de tipo LIFO (last-in, first-out). Su punto de acceso es a través de la cima (desde allí se pueden agregar o quitar elementos)... El uso habitual es:

```
push a
push b
push c
...
...
pop c
pop b
pop a
```

2. ¿Qué utilidad puede tener la utilización de una estructura de tipo pila o stack? ¿Qué implicaciones en cuanto a la programación tiene este tipo de estructura?

Tiene gran aplicación en la ejecución de procesos, por ejemplo, en un sistema multiprogramado, la cpu puede estar ejecutando el proceso A, supongamos que éste realiza una operación de E-S, la cpu puede "apilar" toda la información del proceso A, interrumpirlo, poner en ejecución el proceso B y si luego que quiere retomar la ejecución del proceso A en el punto en el que había quedado, se puede "desapilar" la información del proceso A. Las implicancias en cuanto a la programación estan indicadas en el punto anterior, en cuanto al orden de "apilamiento" que hay que respetar, si apilo los datos de los registros a,b,c debo desapilarlos en el orden inverso (c,b,a).

3. ¿Qué relación hay entre una pila o stack y las operaciones PUSH y POP?

Push -> apilar, poner
Pop -> desapilar, sacar

4. ¿Cuáles son los objetivos de un SO?

1-SO como interfaz de usuario...
2-Eficiencia (uso eficiente de los recursos del computador)...
3-Evolución (permitir agregar nuevas funciones sin afectar a los servicios que brinda)...

5. Una vez que el SO está en funcionamiento, ¿El núcleo o kernel siempre permanece cargado en memoria? O bien el mismo se carga y descarga de la memoria según lo requiera la administración de memoria del SO.

Siempre permanece cargado en memoria.

6. ¿Cuáles eran los principales problemas del procesamiento en Serie de los primeros SO's?

-Planificación: uso de computadores costosos como monousuarios, monotarea.
-Tiempo de Preparación: cargar compilador, cargar programa, guardar programa compilado, cargar programa compilado, montar y desmontar cintas, etc.

7. En el procesamiento en Serie, ¿había algún tipo de aislación (isolation) entre el usuario y el hardware utilizado?

No.

8. En el procesamiento batch o por lotes, ¿había algún tipo de aislación (isolation) entre el usuario y el hardware utilizado?

Si, el monitor.

9. ¿De qué manera el usuario podía interactuar con el Monitor?

A través del conjunto de instrucciones JCL (job control language) que se acompañaba con cada trabajo a ejecutar.

10. En un sistema de procesamiento batch o por lotes, ¿Qué sucede cuando un programa del usuario ejecuta una instrucción privilegiada (asumiendo que el hardware tiene la facilidad de contar con este tipo de instrucciones)? ¿Cuál cree Ud. que es la razón de ello?

Si un programa de usuario pretende ejecutar una instrucción privilegiada se producirá un error, puesto que este tipo de instrucciones no esta permitido para un programa de usuario. Solo el monitor puede ejecutar instrucciones privilegiadas puesto que éstas tienen mayor prioridad que las instrucciones del usuario, si los programas de usuario pudieran ejecutar estas instrucciones habría -al menos- dos problemas: 1. si un programa de usuario acaparara todo el tiempo de cpu ejecutando instrucciones privilegiadas (de mayor prioridad), el monitor no podría tomar el control. 2. se perdería la aislación / abstracción del hardware que provee el monitor, la interacción con el hardware debe hacerse a través del monitor y no directamente desde los programas de usuario.

11. Las instrucciones privilegiadas incluyen también operaciones de I/O, entonces, ¿Cómo es posible que un programa de usuario pueda realizar operaciones de I/O?

Porque las hace a través del monitor y no directamente.

12. ¿Qué areas de memoria están disponibles para un programa de usuario cuando éste se ejecuta en “modo usuario” (user mode)?

Solo puede acceder a su propia area de datos y aquellas a las que se le haya otorgado acceso.
No puede acceder a areas de memoria del monitor o de otro proceso.

13. ¿Qué areas de memoria están disponibles para el monitor teniendo en cuenta que éste se ejecuta en “modo núcleo” (kernel mode)?

El monitor puede acceder a cualquier area de memoria.

14. En un sistema de tiempo compartido o time sharing, ¿el sistema de procesamiento batch o por lotes es reemplazado por la terminal? O bien son sistemas en donde pueden convivir tanto el procesamiento batch como procesos interactivos.

Son sistemas en donde pueden convivir tanto el procesamiento batch como procesos interactivos. La terminal es solo un proceso más.

Respuestas Orientadoras e incompletas (en algunos casos) a las preguntas de revisión. Favor tomarlas como orientadoras.

Clase 3 – Revisión

1. ¿Cuáles son las características de un proceso?

-Programa en ejecución
-"espíritu animado" de un programa
-entidad asignada a CPU y ejecutada por ésta
-instancia de programa en ejecución
-unidad de actividad caracterizada por la ejecución de una secuencia de instrucciones, un estado y un conjunto de recursos asociados
-entidad formada por n elementos (codigo del programa, conjunto de datos, PCB (process control block))
-entidad que describe un comportamiento o taza, listado de la secuencia de instrucciones del proceso

2. ¿Cuáles son los componentes de un proceso?

-Programa ejecutable
-Datos asociados al programa (variables, espacio de trabajo, buffers, etc.)
-Contexto de Ejecución: info. necesaria para que el SO pueda administrar el proceso, contenido de los registros del procesador, PC, registros de datos, prioridad, etc.

3. Teniendo en cuenta las dos preguntas anteriores y sumado al hecho de la existencia de sistemas con multiprogramación, ¿Qué problemas podrán causar n procesos ejecutándose al mismo tiempo en una misma CPU?

-Sincronización Incorrecta: mal manejo de interrupciones, diseño incorrecto de mecanismo de señalización.
-Fallos de Exclusión mutua: problemas de concurrencia: mecanismo de exclusión para que solo uno por vez pueda tomar el recurso.
-Funcionamiento no determinista del programa: la salida debe depender de la entrada y no de las actividades realizadas por otros programas.
-Interbloqueos: dos o más programas suspendidos, uno a la espera del otro (abrazo mortal), por ejemplo con asignación de dos dispositivos de E-S, forma imprevista de asignación y liberación de recursos.

4. ¿Qué ventajas cree Ud. que acarrea el hecho de trabajar con memoria virtual?

Permite a los programas direccionar memoria desde el punto de vista lógico, sin importar la cantidad de memoria principal física disponible, permite n procesos concurrentes. Las referencias se manejan por medio de una dirección virtual (a través de un sistema de paginación de memoria, se referencia como un número de página y un desplazamiento) en donde el sistema de paginación proporciona una proyección a una dirección real o física de la memoria principal.
Las páginas de un proceso en ejecución se guardan en el disco y sólo se tienen en memoria las mínimas requeridas para la ejecución del proceso, cuando se referencia una dirección que no esta en memoria principal, el hardware de gestión de memoria lo detecta y permite que la página que falta se cargue; a esto se lo llama área de memoria virtual.
La CPU y el SO dotan al usuario de un "CPU virtual" que accede a memoria virtual intercalando un traductor de direcciones (memory management unit, MMU, mapper) entre la CPU y la memoria principal.

5. ¿Qué diferencia hay entre un hilo (thread) y un proceso?

Un proceso es una unidad de asignación de recursos y de protección, la cual puede ser implementada bajo un modelo de un único hilo o bajo un modelo multihilo.

Cada hilo tiene su información contextual (BCP), su pila o stack, pero comparte el BCP del proceso al que pertenece el hilo y tiene acceso al espacio de direcciones del proceso común a todos los hilos, ver Fig. 4.2 Pag. 160.
Cada hilo tiene: su estado de ejecución, contexto, pila, espacio de direcciones para variables locales, acceso a memoria y recursos del proceso, compartido por todos los hilos de su mismo proceso.

6. ¿Qué diferencia hay entre la arquitectura micro-núcleo (microkernel) y la arquitectura multi-hilo (multithreading)?

Una Arquitectura micronúcleo (microkernel) se refiere a una decisión de diseño que se debe tomar en todo SO moderno, la idea es asignar sólo unas pocas funciones escenciales al núcleo del SO que permanecerá en memoria, tales como: -manejo de espacio de direcciones de memoria, comunicación entre procesos, planificación básica de procesos. Esta decisión de diseño está en contraposición con la aquitectura monolítica. La arquitectura multihilo (multithreading) se refiere a la habilidad de un SO para soportar múltiples y concurrentes flujos de ejecución dentro de un mismo proceso, tal como es el caso de windows y los unix modernos (linux, solaris, etc). Sin embargo, los antiguos unix soportaban n procesos concurrentes pero un solo hilo de ejecución por cada proceso. Se puede o no hacer uso de multihilo dentro de una arquitectura microkernel o monolítica, dependiendo si el SO soporta o no multihilos.

7. Responda Verdadero o Falso:
   En un sistema de multiprocesamiento simétrico (SMP):
   * existen n procesadores cada uno con su propia memoria (F)
   * existen 1 solo procesadores que comparte su memoria con n hilos o threads (F)
   * existen n procesadores que permiten ejecutar distinto set de instrucciones (F)
   * existen n procesadores que permiten ejecutar el mismo set instrucciones (V)
   * un procesador para poder ejecutar una instrucción debe esperar a que los otros procesadores no estén ejecutando ninguna instrucción (F)
8. Suponga que un usuario UNIX está usando la aplicación *vi* (editor) modificando un archivo de texto. Enumere las capas que se encuentran entre este usuario y el hardware de su computador.

1. Unix commands and Libraries (vi)
2. System call interface
3. Kernel
4. Hardware

9. ¿Por qué razón Ud. cree que los diseñadores han optado por un modelo por capas? ¿Qué ventajas cree Ud. que trae el programar de esta forma? ¿Podemos utilizar esta idea en la construcción de software aplicativo?

El desarrollo de software por capas requiere de capas de software con un objetivo específico e interfases bien definidas, toda capa de software debe estar *especificada*. Esto facilita el trabajo en grupo, en donde desarrolladores se concentren en determinada capa, desconociendo el resto, bajando el nivel de complejidad y acelerando la curva de aprendizaje. Las capas pueden ser provistas por distintos proveedores, pueden corregirse, actualizarse (hacia su interior) y hasta incluso ampliarse, sin interferir con el resto (encapsulación). Esta forma de desarrollo puede aplicarse a cualquier tipo de software.

10. Linux no utiliza la arquitectura de micro-núcleo (microkernel), describa brevemente cuál es la idea de su implementación en contraposición con la arquitectura de un UNIX tradicional.

Linux no tiene un diseño microkernel sino una aquitectura modular, se trata de una colección de módulos que se cargan en tiempo de ejecución (runtime) y se pueden vincular y desvincular (también en runtime) del kernel a través de los comandos *insmod , rmmod*. Los módulos son apilab les (stackable) de forma jerárquica. De forma tal, que un modulo puede servir como librería de otro módulo cliente. La idea es que un módulo pueda implementar el código común a n módulos, de esta

forma, se reduce la cantidad de código que el SO debe mantener en memoria, no hay duplicación de código, facilita el desarrollo de módulo más específicos, permite definir las dependencias entre módulos y el kernel esta seguro de contar con todos los módulos requeridos.

Respuestas Orientadoras e incompletas (en algunos casos) a las preguntas de revisión de las clases 4,5,6,7. Favor de tomarlas como orientadoras.

Clase 4 – Revisión – Unidad II – Procesos (hasta modelo de 5 estados)

1. Enumere 4 oraciones que definan qué es un proceso.

Programa en ejecución.
Instancia de un programa ejecutado en un computador.
"Espíritu animado" de un programa.
Entidad asignada a una CPU y ejecutada por ésta.
Unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual y un conjunto de recursos del sistema asociados.

2. ¿Qué es la traza de un proceso?

Esta asociado con el comportamiento del proceso, es la secuencia de instrucciones ejecutada por el proceso.

3. ¿Por qué razón el modelo de dos estados no es apropiado para describir lo que realmente ocurre con los procesos?

Porque este modelo asume que todos los procesos en estado "not running" pueden pasar a estado a "running" una vez seleccionados por el dispatcher y en la realidad esto no es así, puede haber procesos que queden en estado "not running" porque estan esperando la finalización de una operación de entrada/salida.

4. ¿A qué nos referimos con “process spawning”?

"Process spawning" es cuando un proceso crea otro proceso, a través de la operación fork(). Esto también se denomina creación de procesos "pesados" en contraposición a la creación de procesos "livianos" (threads/hilos).

5. ¿A qué nos referimos con “swapping”?

Se refiere a mover todo o parte de un proceso que se encuentra actualmente en memoria principal hacia memoria secundaria. El area de swap es un area de intercambio entre la memoria principal y la secundaria.

6. ¿Qué diferencia hay entre un proceso en estado “Blocked” y otro en estado “Blocked/Suspend”?

El lugar de residencia. Ambos procesos estan bloqueados, solo cambia su ubicación, el proceso "blocked" se encuentra ocupando memoria principal, mientras que el proceso "blocked/suspend" se encuentra en memoria secundaria. Este tipo de estados se aplica en SO que utilizan memoria virtual, permitiendo realizar swapping entre los distintos procesos a ejecutar.

7. ¿ A qué nos referimos con “time overrun”?

Para muchos eventos el SO establece un tiempo límite, cuando se excede el tiempo límite de espera de un evento que se supone debiera haber ocurrido y éste no ocurre, se termina el proceso debido a un "time overrun".

8. ¿En dónde se encuentra un proceso suspendido?

En memoria secundaria.

9. Describa los 5 estados posibles de un proceso.

Running: proceso en ejecución.
Ready: listo para ejecutar cuando le den la oportunidad.
Blocked/Waiting: proceso que no puede ejecutarse hasta que algún evento ocurra, por ejemplo, la finzalización de una operación de entrada/salida.
New: proceso recien creado pero aún no incluído dentro del pool de procesos ejecutables.
Exit: un proceso que ha sido sacado (por el SO) del pool de procesos ejecutables

10. ¿Por qué razones puede suspenderse un proceso?

-Para permitir realizar swapping del proceso y liberar memoria principal.
-Debido a que el SO ha detectado algún interbloqueo con otros procesos.
-Por decisión del usuario, para hacer debug del proceso o porque el mismo esta haciendo uso de algún recurso.
-Por temporización del proceso, ejemplo un proceso hace un simple chequeo cada 30 segundos y luego queda en estado suspendido, así sucesivamente hasta que le toca nuevamente hacer el chequeo.
-En una situación de spawning el proceso padre puede decidir suspenderse a la espera de la finalización de la ejecución del proceso hijo (coordinación, ejemplo un shell o intérprete de comandos) o también puede decidir suspender la ejecución del proceso hijo para inspeccionarlo o modificarlo.

Clase 5 – Revisión – Unidad II – Procesos (Estructuras para gestionar procesos – modos de ejecución – modos de ejecución del SO – creación de procesos Unix/Linux)

11. ¿Por qué es necesario para el SO mantener estructuras de control de los procesos?

Para que el SO pueda seguir la traza de los procesos, para poder administrarlos debe mantener el registro de estado de cada proceso y su informacion de contexto, puesto que la ejecución de los procesos es intercalada/interrumpida con el process switcher y los otros procesos en ejecucion. La cpu "se presta" entre los distintos procesos, incluyendo los procesos del propio SO.

12. ¿Qué categorías de información hay en un bloque de control de proceso?

Identificación de Proceso, Info. de estado cpu, Info. Control de Proceso.

13. ¿Qué es la imagen de un proceso?

Fig.3.16 pag.142. Imagen=bloque de control de proceso (PCB)+pila o stack usuario+espacio de direcciones de usuario+pila núcleo+espacio compartido de direcciones

14. ¿Por qué razón se realiza un cambio de proceso (process switching)?

Pag.138. se realizar porque ocurre una interrupción (causa externa al proc. en ejecucion), ocurre un trap (causa interna al proc. en ejecución), se realiza una llamada al SO (system call).

15. ¿Por qué razón se realiza un cambio de modo?

Porque ha ocurrido una interrupción, se cambia de modo usuario a kernel (para ejecutar instrucciones privilegiadas) o viceversa (para ejecutar instruccion no privilegidas, en modo usuario), se debe guardar el contexto de ejecucion del proceso.

16. ¿Qué diferencia hay entre “cambio de proceso” y “cambio de modo”?

No son conceptos equivalentes, un cambio de modo afecta al tipo de instrucciones que se pueden ejecutar, pero esto no implica un cambio de proceso. Se puede hacer un cambio de modo dentro del mismo programa, esto esta en relación con los modos de ejecucion que tiene un SO Fig.3.15(b).

17. ¿Qué diferencia hay entre “trap” e interrupción?

Son conceptos similares, pero no no idénticos, un trap es una forma de interrupción pero que se origina dentro del propio proceso interrumpido, asociado a una condicion de error o excepción. Mientras que una interrupción es algo externo al proceso en ejecución, por ejemplo, puede ser que se haya agotado la rodaja de tiempo asignada para la ejecucion del mismo (time slice) entonces se produce una interrupción de reloj (clock).

18. En un entorno Unix, Describa: proceso swaper, proceso 0, proceso 1 y proceso init.

Pag.144. Proceso swapper es sinónimo de proceso 0, es mas bien una estructura de datos que se carga en memoria cuando el sistema arranca. Luego este proceso ejecuta al proceso 1 o proceso init y éste es el padre de todos los procesos.

19. En un entorno Unix, ¿Qué es un proceso zombie?

Es un proceso que ya no esta en ejecución, ya sea por una terminación normal o anormal o porque

fue abortado; pero aún queda su información registrada en el sistema para que pueda consultarla el padre. Obsérvese el código del shell, cuando el proceso padre hace un fork() (crea un hijo), y luego hace una llamada a la función waitpid() (si el hijo termina antes de la llamada a waitpid() por parte del padre, el hijo pasa a estado zombie) y luego el padre necesita que no "desaparezca su rastro" para recuperar el status de finalización del proceso hijo. Una vez que el padre ha recuperado la información de finalización del proceso hijo a través de waitpid(), el hijo deja de estar en estado zombie y pasa al estado terminado [Tanenbaum, Sistemas Operativos Modernos, 3ra. ed., Pag. 744].

20. En un entorno Unix, ¿Para qué sirve la llamada al sistema (system call) fork()? ¿Cómo funciona?

Pag.148. fork() sirve para crear un proceso, el SO realiza las siguientes funciones:
1. Solicita una nueva entrada en tabla de procesos para el nuevo proceso 2. Asigna ID proceso 3. Copia imagen del proceso padre con excepción de las regiones de memoria compartidas 4. Incrementa en uno el contador de ficheros en posesión del padre porque ahora el hijo también puede acceder a los ficheros 5. El nuevo proceso se pone en estado listo para ejecutar (ready to run) 6. Devuelve el id de proceso hijo al padre, devuelve 0 al proceso hijo.

Luego de esto el SO tiene 3 opciones: 1. continuar con la ejecución de padre, volviendo al modo usuario, luego de la instrucción fork() 2. transferir el control al proceso hijo, continuar después de la instrucción fork() 3. transferir el control a otro proceso, dejando a padre e hijo en estado “ready to run”.

Clase 6 – Revisión – Unidad II – Procesos (Hilos – Relación Hilos vs Procesos)

1. ¿Qué diferencia hay entre “monohilo” y “multihilo”?

Fig.4.1 (izquierda) Monohilo = 1 proceso = 1 hilo, por ejemplo MS-DOS; mientras que multihilo implica que el SO da soporte a n hilos en ejecución en un mismo proceso, por ejemplo: Windows, Solaris, Fig.4.1 (derecha).

2. Defina qué es un hilo o thread.

Un hilo es una unidad de ejecución que se activa, tiene su propio pc y consume cpu, llamado también thread o proceso ligero/liviano. Mientras que un proceso o tarea es aquel quien tiene la propiedad o tutela de los recursos requeridos para la ejecución de dicho proceso o tarea, la cual puede estar formada por un único hilo o múltiples hilos de ejecución.

3. ¿Por qué un cambio de contexto entre procesos es más “pesado o lento” que un cambio de contexto entre hilos?

Porque para cambiar entre procesos se requiere la intervención del SO para que éste guarde el contexto de ejecución de proceso actual, lo descargue, recupere el contexto de ejecución del nuevo proceso a ejecutar y lo ponga en ejecución. Mientras que un hilo o thread comparte gran parte del contexto del proceso que lo contiene, por lo tanto se puede cambiar de un hilo a otro dentro del mismo proceso sin necesidad de intervención del SO (tampoco requiere un cambio de modo) y sólo salvando y restaurando el contexto propio del thread, véase Fig. 4.2. Los hilos comparten el bloque de control de proceso, solo hay que salvar/restaurar el bloque de control de hilo.

4. Volviendo al concepto de “process spawning” (clase inicial de procesos) ¿Qué sería más eficiente? ¿Llamar a la función fork() o crear un hilo para la atención de cada cliente?

Es más eficiente crear un hilo de atención para cada cliente, porque es más rápido crear un hilo que hacer un fork().

5. ¿Qué sucede con los hilos del proceso X cuando el SO suspende al proceso X?

En un ambiente ULT (User Level Thread), el SO no hace gestión a nivel de hilos (es más, no sabe de su existencia), toda la planificación es a nivel de proceso, por lo tanto, cuando se suspende un proceso, todos los hilos de ese proceso se “suspenden”, no podrían estar en ejecución mientras el proceso que los contiene queda suspendido; el entrecomillado se debe a que esto no implica que los

propios hilos se reconozcan en estado de suspendido, obsérvese la Fig. 4.7 un proceso puede quedar bloquedado y sin embargo, sus hilos continúan en estado ready y running. En un ambiente KLT el SO realiza una gestión a nivel hilo, no obstante la pregunta no se ajusta demasiado a esta idea puesto que aquí son hilos en ejecución dentro del kernel y no dentro de un proceso de usuario; algo similar ocurre en un ambiente ULT/KLT si el hilo ULT está asociado a un KLT el hilo ULT es como que hereda la gestión del hilo KLT, pero el proceso de usuario que lo contiene bien podría quedar suspendido, en tal caso, la bibliografía no aclara que sucede con dicho hilo, pero seguramente no podrá estar en ejecución más allá del status propio del hilo (similar a lo comentado en el ambiente ULT). La pregunta es más aplicable a un ambiente ULT que a los restantes.

6. ¿En un ambiente ULT (user level thread), quién está a cargo de la planificación de los hilos?

El SO no está a cargo de su planificación, ni conoce de su existencia. La planificación está a cargo del programador, quien por lo general, utilizará una librería o biblioteca de software para gestionar los hilos dentro del programa, un ejemplo de ello es la biblioteca pthread (posix threads).

7. ¿En un ambiente KLT (kernel level thread), quién está a cargo de la planificación de los hilos?

El SO.

8. Existen 4 relaciones posibles entre hilos y procesos (1:1, M:1, 1:M, M:N) ¿Cuál de todas ellas está relacionada con los SO Distribuidos?

1:M en donde un hilo puede migrar de un entorno de proceso a otro, permitiendo que los hilos puedan moverse entre distintos sistemas, ejemplo: SO distribuidos como RA (clouds), Esmerald.

9. ¿Por qué se dice que ULT (user level thread) no saca provecho de la multiprogramación?

Porque muchas llamadas al sistema (system calls) son bloqueantes, cuando un hilo hace una llamada bloqueante, todo el proceso queda bloqueado, porque la gestión en un ambiente ULT es a nivel de proceso, no de hilo, en un momento dado sólo puede haber un único hilo en ejecución (si la gestión es a nivel de proceso, el SO no puede asignar el mismo proceso a más de una CPU al mismo tiempo); en este caso, ULT no está sacando provecho de la multiprogramación.

10. ¿Cuáles son las soluciones posibles al problema planteado en la pregunta anterior? ¿Qué entiende por “jacketing”?

Se plantea la posibilidad de escribir un proceso por cada thread, pero ello no es solución, puesto que perdemos la principal ventaja de los threads. La otra posibilidad es utilizar una técnica llamada “jacketing” la cual hace un envoltorio (wrap) de una llamada bloqueante para “transformarla” en una llamada no bloqueante: por ejemplo, en el caso de una operación de I/O el hilo verifica primero si el dispositivo está ocupado (esto es una llamada a nivel de la aplicación, no del sistema), si el dispositivo está ocupado, seguramente la llamada de I/O al SO provocaría el bloqueo del proceso, entonces no realiza la llamada al SO sino que suspende (a nivel de la aplicación del usuario) el thread y pasa el control a otro thread para tener la oportunidad de preguntar más tarde por el estado del dispositivo (si le da que está desocupado, ahí sí hace la llamada al SO, lo cual no implicaría un bloqueo muy largo).

Clase 7 – Revisión – Unidad II – Procesos (SMP – Micronúcleo)

1. Según la taxonomía de Flynn ¿Cuáles son los 4 tipos de sistemas que existen?

SISD monoprocesador; mientras que los sistemas que soportan procesamiento en paralelo serían: MISD, SIMD, MIMD. De éstos, el que más no interesa es el MIMD, puesto que de allí derivan los clusters y los SMP.

2. ¿Un SMP es un MIMD con memoria compartida (fuertemente acoplada) en el cual el núcleo se ejecuta en CPU determinado?

No. Un SMP es un MIMD con memoria compartida en el cual el núcleo se ejecuta en cualquier CPU.

3. ¿En qué se diferencia un SMP de un Cluster?

En cuanto a la memoria. Un SMP tiene una memoria fuertemente acoplada, es compartida por n CPU's que se encuentran sobre el mismo bus, en la misma motherboard, en un mismo computador Fig. 4.9, Pag.175. Mientras que un Cluster tiene una memoria débilmente acoplada o distribuida, puesto que se trata de n computadores distintos c-u con una o más CPU's y con sus propias memorias.

4. ¿Por lo general, el problema de la coherencia de cachés debe ser resuelto por el SO?

No, es resuelto por hardware.

5. ¿En un SMP si falla una CPU entonces falla todo el sistema?

No, porque el kernel puede ejecutarse en cualquier cpu, el sistema debería detectar la falla y reajustarse para ejecutarse en una cpu menos, debe ser tolerante a este tipo de fallos.

6. ¿Por qué se dice que un núcleo o kernel SMP debe ser "reentrante"?

Porque implica que un mismo código pueda estar ejecutándose al mismo tiempo en más de una cpu con un conjunto de datos distintos; debe ser un código lo suficientemente depurado y probado para evitar interbloqueo u operaciones inválidas.

7. ¿Qué tipo de algoritmos se debe utilizar en cuanto a sincronización en un SMP?

El problema es que puede haber n CPU's accediendo al mismo tiempo a un único conjunto de recursos, entre ellos la memoria principal, por ejemplo. Esto implica que se deben utilizar algoritmos tales como exclusión mutua y cerrojos o candados.

8. ¿Micronúcleo implica SO pequeño?

NO. Es sólo una cuestión de diseño, se pueden hacer SO pequeños y grandes (en cuanto a la funcionalidad que éstos ofrezcan) utilizando este tipo de diseños.

9. ¿Monolítico implica SO grande?

NO. Idem anterior, por lo general, los SO monolíticos son los SO más primitivos (por ende, los más pequeños), pero ello no implica (aunque no parece una buena idea) que no pueda desarrollarse un SO relativamente grande bajo este diseño; seguramente tendría serios problemas de mantenimiento y para asegurar su fiabilidad.

10. ¿SO por capas es igual que monolítico?

NO. La principal diferencia entre uno y otro es su organización interna. Un SO por capas es un SO estructurado, es un diseño más maduro. No obstante, un SO por capas puede compilarse en un único bloque de código que corra todo en un mismo espacio de direcciones al estilo de un SO monolítico, pero no obstante ello, su estructura interna permitiría un mejor mantenimiento. También podría implementarse esta idea de SO por capas en una arquitectura no monolítica, usando un micronúcleo en donde existan distintas capas de servicios c-u con su objetivo, su responsabilidad y su interfase bien definidas. O también podrían ser una serie de módulos a ser cargados en tiempo de ejecución y cada uno de ellos con su espacio de direcciones, comunicados entre sí a través de mensajes como es el caso de Linux.

11. ¿El Micronúcleo corre en modo kernel o núcleo, al igual que el resto del SO?

No. El micronúcleo corre en modo kernel, pero el resto del SO corre en modo usuario.

12. ¿La implementación de IPC (inter process communication) está bajo la responsabilidad del micronúcleo o es algo externo a él?

IPC esta bajo la responsabilidad del micronúcleo, pero cabe aclarar que sólo en cuanto a su forma más elemental de comunicación entre procesos, como es el caso del pasaje de mensajes (con cabecera: remitente, receptor; cuerpo: datos, punteros,etc.; puerto: cola de mensajes por proceso).

13. ¿Qué relación hay entre la arquitectura client-server y el micronúcleo?

Se puede pensar a un micronúcleo como un conjunto de procesos servidores que dan servicio a un conjunto de procesos clientes; en este sentido, hay una relación directa entre ambos, sería el caso de una arquitectura client-server dentro de un mismo servidor. Los procesos servidores validan los mensajes recibidos de los clientes y los direccionan internamente hacia/desde el hardware y los

distintos componentes del SO.

14. ¿Por qué razón se dice que hay un potencial problema de rendimiento en la implementación de un SO con micronúcleo? ¿Cuál sería una posible solución?

Porque el envío, recepción y decodificación de mensajes entre procesos es más lento que hacer una llamada al SO (system call) como podría hacerse en un SO bajo una arquitectura no micronúcleo. Una posible solución está en la reducción del tamaño del micronúcleo, cuanto más pequeño, menos tardarán el traslado y decodificación de mensajes, logrando un rendimiento similar y hasta incluso mejor que en un SO por capas tradicional.

15. Explique por qué un SO con micronúcleo facilita la implementación de Sistemas Operativos Distribuidos

Porque si cada mensaje que se envía dentro del SO incluye un ID de servicio solicitado y este ID de servicio es único dentro del sistema (para todas las cpu's existentes en todas las máquinas que posea el SO), se puede obtener una imagen única del servicio a nivel de micronúcleo, facilitando la implementación distribuida del SO.

16. Acorde con la implementación de hilos en las nuevas versiones de Linux, ¿Qué ventajas obtengo cuando los hilos de usuarios están asociados con hilos del núcleo y todos comparten el mismo id de grupo de proceso?

La asignación de recursos se hace por grupo de procesos por lo tanto, si un hilo a nivel de usuario se asocia con un hilo a nivel de kernel y éste tiene asociado n recursos, ambos tendrían acceso a los mismos recursos, podrían compartirlos, evitando un cambio de contexto cuando el planificador cambia entre procesos del mismo grupo.

17. En Unix/Linux, ¿Qué es un proceso zombie?

Idem 5.19.

Respuestas Orientadoras e incompletas (en algunos casos) a las preguntas de revisión. Favor tomarlas como orientadoras.

Revisión – Unidad III – Sincronización entre Procesos

1. ¿Podría aplicarse la exclusión mutua sobre una sección crítica?

Si, es lo que debe hacerse; si se aplicara una exclusión mutua sobre una sección no crítica, ésta no tendría mayor sentido.

2. Unir con flechas:

Inanición	mutual exclusion
Condición de Carrera	livelock
Exclusión Mutua	deadlock
Interbloqueo	race condition
Circulo Vicioso	startvation

Inanición--> startvation
Condición de Carrera-->race condition
Exclusión Mutua--> mutual exclusion
Interbloqueo-->deadlock
Circulo Vicioso-->livelock

3. ¿Puedo coordinar n procesos teniendo en cuenta las velocidades de ejecución de cada proceso?

No, no puede predecirse la velocidad relativa de ejecución de los proceos.

4. ¿Qué relación Ud. Encuentra entre la dificultad de encontrar errores y la condición de carrera?

La dificultad reside en que los resultados obtenidos -cuando hay una condición de carrera- depende del orden de ejecución de las intrucciones de n procesos (tal vez, con n hilos cada uno) y ésto a su vez esta determinado por el planificador de procesos y otros factores, con lo cual, no es fácilmente reproducible, se rompe con el modelo determinístico al cual debería responder la ejecución de procesos.

5. ¿Qué tipo de problemas puede provocar la implementación de la exclusión mutua?

Interbloqueo (deadlock) e Inanición (startvation)

6. ¿Qué sucede si un proceso queda indefinidamente dentro de una sección crítica?

El resto de los procesos -que también ejecutan esta sección crítica- quedarán bloqueados intentando entrar en ella.

7. Responda V o F: La espera activa es una buena solución de software, puesto que no consume tiempo de CPU.

Falso. La espera activa consume tiempo de cpu.

8. Responda V o F: El algoritmo de Dekker es mejor que el algoritmo de Peterson, porque tiene menor carga de procesamiento.

Falso. El algoritmo de Dekker es mas complejo, menos elegante, pero no puede afirmarse que tenga una menor carga de procesamiento.

9. ¿La deshabilitación de interrupciones es una técnica de hardware que da soporte a la exclusión mutua en un SMP?

Falso, porque en un SMP hay n cpu's que pueden estar ejecutando n procesos al mismo tiempo y un SMP no posee un mecanismo de interrupción de n cpu's que comparten la misma memoria.

10. Responda V o F: Deshabilitar interrupciones en un SMP implica evitar el entrelazado de procesos.

Falso, idem anterior.

11. Responda V o F: Deshabilitar interrupciones en un computador con una sola CPU implica evitar el entrelazado de procesos.

Verdadero.

12. ¿Las soluciones de hardware son mejores que las soluciones de software, en cuanto al problema de la exclusión mutua?

No, tanto las soluciones de software como las de hardware son insuficientes para tratar el problema de la exclusión mutua, se requiere soporte del SO.

13. ¿Las instrucciones de CPU atómicas surgen como una mejora a la deshabilitación de interrupciones?

Si, mas que nada para permitir implementar un mecanismo de exclusión mutua por hardware en un SMP. Si bien permiten mejorar la eficiencia de la cpu permitiendo el entrelazado de procesos, la mejora se ve disminuida debido a que las operaciones atómicas requieren de espera activa.

14. Responda V o F: Tanto las soluciones de software como las instrucciones atómicas de hardware evitan la espera activa.

Falso, en ambos casos hay espera activa.

15. Responda V o F: Las instrucciones de CPU atómicas permiten implementar algoritmos de exclusión mutua de forma más eficiente que las soluciones de software.

Falso. Idem anterior.

16. ¿Por qué razón Ud. Cree que el SO es el que puede dar el mejor soporte a la exclusión mutua?

Porque el SO es quien controla al dispatcher, a la ejecución de todos los procesos del computador, tiene una posición privilegiada para tener una visión global en cuanto a todos los procesos y facilitar mecanismos de exclusión entre ellos.

17. Responda V o F: Según Dijkstra, la exclusión mutua no puede resolverse usando semáforos.

Falso.

18. Responda V o F: semSignal(s) es una instrucción atómica, no interrumpible, que envía una señal al semáforos.

Verdadero.

19. Responda V o F: semWait(s) incrementa s.

Falso, semWait(s) decrementa s.

20. Responda V o F: semSignal(s) decrementa s.

Falso, semSignal(s) incrementa s.

21. Responda V o F: Una de las ventajas de los semáforos es que su valor se puede cambiar sin necesidad de usar semWait() o semSignal().

Falso, no se puede manipular el valor de un semáforo fuera de semSignal() y semWait().

22. Responda V o F: Un semáforo binario es más rápido y potente que un semáforo no binario.

Falso, solo son más faciles de usar, pero ello no implica que sean más potentes.

23. ¿Qué es un proceso productor?

Proceso que genera algún tipo de dato, poniéndolo en un buffer del que luego será extraído por un proceso consumidor.

24. ¿Qué es un proceso consumidor?

Proceso que extrae datos del buffer cargado previamente por el productor.

25. ¿Por qué razón se desarrollaron los monitores?

Debido a la dificultad y a la probabilidad de error que implica el uso de semáforos distribuidos en distintas partes del sistema. No es fácil seguir la secuencia de ejecución de semWait() y semSignal() en distintas partes del sistema para los distintos semáforos.

26. ¿Qué relación hay entre los monitores y la OOP?

Dos características comunes de los monitores y la OOP: 1. variables locales, solo accesibles a través

de procedimientos del monitor y no por otros procedimientos externos (encapsulación); 2. Un proceso entra en un monitor invocando el uso de sus procedimientos (se accede al interior del objeto solo a través de sus métodos).

27. ¿Qué es una variable de condición?

Son variables al interior del monitor que permiten la sincronización entre distintos procesos: cwait(c) bloquea el proceso esperando por la condicion c; csignal(c) retoma la ejecución de cualquier proceso bloqueado a la espera de la condicion c.

28. En cuanto a cola de mensajes, ¿Qué tipo de primitivas receive() existen?

Las categorías más comunes son: bloqueante (queda el proceso bloqueado, en espera no activa, hasta que llegue algún mensaje), no bloqueante (no queda bloqueado esperando, devuelve algun valor indicando la llegada o no de un mensaje). También existe el receive() como comprobación de llegada de mensaje (al estilo de poll() o select()). Otra clasificación puede ser receive() explícito (se indica de qué proceso se espera recibir un mensaje) e implícito (recibe mensaje de cualquier proceso).

En cuanto a los trabajos prácticos:

1. ¿Es posible aplicarle cierto time-out a una operación bloqueante read()?

Si, a través del uso de las funciones poll(), select() o bien usando la señal de alarma.

2. ¿Las señales se lanzan como eventos sincrónicos o asincrónicos?

Las señales pueden ser lanzadas en forma sincrónica o asincrónica. Generalmente, en forma sincrónica cuando están vinculadas a acciones dentro y bajo el control del programa. En forma asincrónica cuando se lanzan en respuesta a eventos que están fuera del control del programa que las recibe (ejemplo, el proceso A responde asincrónicamente a la señal SIGUSR1 que fue enviada por el proceso B).

3. ¿Qué sucede si un programa entra en loop y comienza a lanzar en forma repetitiva una misma señal a otro proceso?

Supongamos que el proceso A entra en loop enviado la señal SIGUSR1 al proceso B en forma repetitiva, en tal caso, el SO, encolará una sola vez la señal SIGUSR1 para el proceso B.

4. ¿Una señal de interrupción tiene mayor prioridad que una señal de usuario?

No, todas las señales tienen la misma prioridad, no hay señales más prioritarias que otras.

5. ¿Qué cuidados debe tener el programador que utiliza la función pause()?

El cuidado es que no se produza una señal en forma temprana, antes de lo previsto por el programador, antes de que se ejecute pause(), porque, si esta fuese la única señal generada, el proceso quedará bloqueado indefinidamente, ya que no arribarán más señales a este proceso, impidiendo la finalización de la función pause().

6. ¿La llamada a la función sleep() implica una espera activa del proceso que la invocó?

No, sleep() bloquea el proceso actual, no es una espera activa.

7. En un contexto de multithreading, ¿Tiene sentido escribir sentencias luego de invocar a la función pthread_exit()?

No, estas sentencias no serán ejecutadas.

8. ¿Qué sucede cuando un thread invoca a la función pthread_join()?

Si el thread t1 invoca la función join(t2) implica que t1 quedará bloqueado hasta que finalice t2; si t2 ya estaba finalizado al momento de invocar el join(), join() no hace nada y t1 continua con su ejecución.

9. ¿Qué tipo de atributo debe tener un thread sobre el cual se ejecuta una operación de join?

El thread debe ser de tipo "joineable" (atributo PTHREAD_CREATE_JOINABLE). Puesto que un thread de tipo "joineable" requiere de una serie de controles, por defecto, todo thread es "no joineable" mientras no se declare lo contrario.

10. ¿Qué diferencia hay entre un semáforo y una variable de condición?

Un semáforo actúa de forma distinta a una variable de condicion. Por ejemplo, un semáforo binario o mutex simplemente es algo de tipo "pasa" o "no pasa", incrementando o decrementando el valor del semaforo, permitiendo la implementación de una exclusión mutua. Una variable de condición se usa en combinación con mutex (una especie de semáforo binario, como se explicó anteriormente), una vez que se ingresó en la sección crítica, se pregunta por una determinada condición y en tal caso, se señala una variable de condición (csignal(c)) ; se supone que hay otro proceso esperando por la señalización de esa condicion (cwait(c)). El csginal(c) provoca el bloqueo del proceso actual y la liberación del mutex actual, se adquiere el mutex y se desbloquea el proceso que estaba esperando por la condicion c (cwait(c)).

11. ¿Para qué sirve el comando "time"? ¿Cómo se usa?

El comando unix time sirve para contabilizar los recursos utilizados por un proceso
Forma de uso: time [opciones] <proceso> Ejemplo: $ time ./tp81
En el ejemplo anterior se obtendría una salida que indicaría (expresado en horas, minutos, segundos) el Real Time (tiempo real, wall clock) transcurrido en la ejecución del programa, el User Time (tiempo que el proceso corrió en modo usuario, no privilegiado) y el System Time (tiempo que el proceso corrió en modo kernel o núcleo o privilegiado).

12. Supongamos que tengo la sección crítica C y hay dos funciones f1() y f2() que acceden a dicha sección crítica. La función f1() tarda unos 200 milisegundos, f2() tarda unos 500 milisegundos; entonces ejecuto f1() y luego f2(); al comienzo de f2() pongo una demora de unos 300 milisegundos. Ambas funciones se lanzan con pthread_create(). Responda:

a) Ud. es el Analista a cargo del desarrollo de la aplicación X y un programador vino con este planteo para un proceso que Ud. le encargó ¿Aprueba Ud. este diseño?

Falso, no puede aprobarse este tipo de sincronización basado en supuestos tiempos de duración de los procesos.

b) ¿Funciona el diseño planteado en el punto 12?

Eventualmente podría funcionar, pero no se puede asegurar un funcionamiento libre de errores ya que esta supeditado al tiempo real que dure la ejecución de cada proceso, lo cual puede variar por múltiples factores (planificador, tiempos de I/O, carga actual del sistema, etc.).

c) ¿Qué otro diseño podría aplicarse al proceso del punto 12?

La forma correcta de diseñar esto sería implementar un mutex para las funciones f1() y f2() sin importar los tiempos de ejecución de los procesos, sin implementar demoras que impacten en el rendimiento de este proceso. Sería un diseño sólido y de mejor rendimiento que el propuesto.

13. ¿Por qué razón Ud. cree que las prácticas proponen devolver valores distintos en la función main() para las distintas situaciones de error detectadas?

Se pueden capturar los valores de retorno en el shell del SO y saber con exactitud que tipo de error se ha dado en la aplicación. Esto facilita la ubicación del error y su corrección.

14. ¿Qué es una señal nula? ¿Para qué sirve?

La señal nula es aquella cuyo valor es 0 (nulo decimal) y generalmente se utiliza para determinar si un proceso esta en este momento en ejecución o no, si la función kill() devuelve -1 y errno = = ESRCH indica que el proceso en cuestión no existe.

15. ¿Qué significa que una función no es re-entrante?

Una función no re-entrante es una función que utiliza datos que estan fuera del stack, por ejemplo una variable estática o global o dinámicamente asignada (usando malloc(), calloc(), realloc()) la cual es accedida desde el código de esta función (apuntando siempre a la misma dirección) y que habiendo dos o más llamadas concurrentes a dicha función pudiera haber una interferencia en su funcionamiento. La interferencia se produce sobre el dato que esta fuera del stack (su copia privada de datos) y sobre el cual no hay un mecanismo de exclusión mutua implementado.

16. ¿Una variable de tipo volatile implica que su valor vuelve a su valor por defecto luego de cierto tiempo o bajo determinada condición?

No, una variable debe declararse como volatile si la misma es accedida en forma asincrónica desde una o más funciones; esto inhibe ciertas optimizaciones del compilador y permite que las actualizaciones de la variable se hagan de forma atómica.

17. ¿Es posible cambiar la imagen del proceso actual por la de otro proceso?

Si, es lo hace el shell cuando combina la función fork() con la función exec*().

18. ¿Qué significa que un programa no se comporte acorde con un "Modelo Determinista"?

Que, dada una misma entrada, no siempre se obtiene la misma salida.

19. ¿Qué es un coproceso?

Se trata de dos procesos que colaboran de forma muy estrecha, como es el caso de la implementación de pipes half-duplex entre procesos emparentados, en donde el proceso padre escribe en el stdin del proceso hijo y lee del stdout del proceso hijo.

20. Tengo un programa interactivo (requiere que el operador tipee datos por teclado y obtenga salidas por pantalla), ¿Puedo -a través de otro proceso- reemplazar al operador (obviamente suponiendo un comportamiento determinístico)?

Si, a través del uso de pipes sin nombre (unnamed pipes) entre proceso emparentados, generamos un proceso padre que ejecute a otro proceso hijo usando fork() + exec*() y luego el proceso padre será quien simule la interacción humana.

21. Desde el punto del vista del SO, ¿Cómo se identifica cada recurso de tipo IPC y Cómo podemos consultarlo y/o administrarlo?

Se identifica a través de un número entero positivo, este número es de tipo key_t y esta definido dentro del archivo de cabecera sys/types.h . Si dos o más programas pretenden compartir este recurso IPC, los procesos deberán conocer este número para poder accederlo de forma unívoca. El comando ipcs permite la administración (por fuera de las aplicaciones) de los recursos IPC actualmente existentes en el kernel del SO.

22. De la misma forma que un SO tiene su API (system calls), cualquier aplicación que desarrollemos también podría proveer (hacia el exterior) su propia API, ¿En qué archivo se especificaría esta API?

La api externa de la aplicacion se indica en los archivos de cabecera (headers) *.h. Puesto que, si la API luego se distribuirá como una librería, quien pretenda usarla debe conocer su interfase y la misma esta descripta en cada uno de los prototipos de las funciones a utilizar -desde el exterior- dentro del archivo .h y su forma de uso debe indicarse en el manual del sistema. Si una función no será ejecutada desde el exterior, sino que su reuso será interno, simplemente puede definirse su prototipo dentro de los archivos .c que tienen su implementacion (se debe quitar su prototipo del archivo de cabecera .h para que no sea "visible" desde el exterior, puesto que C no tiene los cualificadores de control de acceso de los lenguajes orientados a objetos, tales como: friend, public, protected, private, etc. Para C todas las funciones son public).

23. ¿Qué otra razón -además de la indicada en el TP VII Cola de Mensajes- se le ocurre a Ud. que un Analista puede tener para definir una API implementada en la librería X para ser reusada en las aplicaciones A,B,C,D?

Existen varias razones, definir una API implica definir la interfase de una capa de software con la cual interacturán las aplicaciones A,B,C,D; de esta forma, el Analista puede implementar cambios hacia el interior de la capa de software, mientras que no altere la API previamente definida, las aplicaciones seguirán funcionando como siempre, sin necesidad de cambio. Facilita el training del personal, es cuestion de aprender a utilizar esta API para poder dar soporte a las aplicaciones A,B,C,D sin necesidad de conocer los detalles de implementación de esta capa de software. Si hubiese algún error dentro de esta capa de software, será cuestión de modificar una llamada de esta API y todas las aplicaciones asimilarán el cambio. ...

24. Responda Verdadero o Falso y justifique: "La memoria compartida es la forma más lenta de IPC”.

Falso, se considera la forma más rápida de IPC porque no se requiere que los datos de comunicación entre procesos requieran ser copiados a areas de memoria locales a los procesos que se comunican.

25. Enumere algunos ejemplos de funciones bloqueantes vinculadas con IPC.

msgrcv() intenta recibir un mensaje de la cola de mensajes, si ésta esta vacía el proceso quedará bloqueado. Otro ejemplo puede ser la función semop() usada para incrementar/decrementar los valores de un semaforo, si un proceso pretente decrementar el semaforo actualmente usado por otro usuario, éste quedará bloqueado hasta que el semafóro sea liberado.

26. Responda Verdadero o Falso y justifique: “El programa cliente que se conecta a un area de memoria compartida debe especificar (en la conexión) el tamaño de la misma”.

Falso. Razon por la cual la API de la librería myipc tiene dos llamadas distintas: una para crear la memoria (proceso server) y otra para conectarse a la memoria compartida (proceso cliente) y en este último caso, no se requiere indicar el tamaño de la memoria compartida.

27. Responda Verdadero o Falso y justifique: “Una de las principales ventajas de la memoria compartida es que nos libera de la preocupación de la compartición de datos, puesto que implementa automáticamente mecanismos de exclusión mutua”.

Falso. La memoria compartida sólo permite el acceso controlado (a través de los permisos que indique el proceso owner de dicha memoria compartida) a un area de memoria comun, mantenida como un recurso centralizado en el kernel del SO, pero no implementa ningún tipo de exclusión mutua sobre la misma.

28. ¿Qué es lo que hace esta sentencia C: `for(i=0;i<10;i++,p++) *p *= 1.1;` ?

Si p es un puntero a un arreglo de números de tipo double o float, este código lo que hace es incrementar en un 10 % los primeros 10 valores contenidos en dicho arreglo. Luego de esta instrucción, p estará apuntado al elemento 11; si el arreglo solo tiene 10 elementos, p estará apuntando más allá de los límites del arreglo.

29. ¿Qué relación hay entre el estado "zombie" de un proceso y la llamada al sistema waitpid() ?

Un proceso *h* queda en estado "zombie" (una especie de estado suspendido) cuando *h* termina y su proceso padre *p* aun no lo estaba esperando (a través de la ejecución de la llamada al sistema waitpid(), por ejemeplo); cuando por fin el proceso *p* llama a waitpid(), el proceso *h* deja el estado "zombie" y termina [Tanenbaum, Sistemas Operativos Modernos, 3ra. Ed, Pag 744].

30. ¿Cuántas llamadas pendientes de alarma puede tener un proceso?

Una sola [Tanenbaum, Sistemas Operativos Modernos, 3ra. Ed, Pag 745].

31. ¿Qué sucede si se solapan dos llamadas de alarm() en un proceso? Por ejemplo, supongamos que un proceso hace una llamada a alarm() con 10 segundos y 3 segundos más tarde, se hace otra llamada a alarm() con 30 segundos.

Como todo proceso no puede tener mas de una alarma pendiente, lo que sucede es que la segunda llamada a alarm() cancela a la primer llamada y provocará que se genere una señal de alarma 20 segundos después de la segunda llamada a alarm() [Tanenbaum, Sistemas Operativos Modernos, 3ra. Ed, Pag 745].

En cuanto a Concurrencia, Interbloqueo e Inanición:

1. Supongamos que dos procesos (A,B) que se ejecutan concurrentemente, requieren los recursos R1 y R2. Respnda Verdadero o Falso:
   a. Siempre -sin excepción- se producirá el interbloqueo

Falso, pueden existir varios caminos seguros en las trayectorias de los proceso A y B.

   b. Es factible que exista una o más trayectorias (en la ejecución concurrente de A y B) que eviten el interbloqueo

Verdadero.

c. El interbloqueo se evita cuando los procesos entran en la "región fatal"

Falso, el interbloqueo se produce cuando se ingresa en una región fatal.

d. Evitar el interbloqueo implica encontrar "un camino seguro"

Verdadero, se evita el interbloqueo no entrado a una región fatal, manteniendo al sistema es un estado seguro.

e. "Un camino seguro" es un camino que no pasa por una "región fatal"

Verdadero.

2. Responda Verdadero o Falso: El interbloqueo siempre ocurre por competencia de recursos.

Falso. Tambien puede ocurrir por comunicación entre procesos.

3. ¿Cuáles son las condiciones necesarias (aunque no suficientes) para la ocurrencia del interbloqueo?

Exclusión mutua. Retención y Espera (un proceso tiene recursos asignados mientras espera por otros). Sin Expropiación (no se le puede forzar la expropiación de un recurso a un proceso que lo posee).

4. ¿Para qué sirve un "Grafo de Asignación de Recursos"?

Permite graficar interbloqueos.

5. ¿Qué forma tiene un "Grafo de Asignación de Recursos" cuando estamos en presencia de una "espera circular"?

Son grafos cerrados, circulares, cíclicos, en donde todos los procesos tienen asignados recursos y al menos uno de ellos pretende la asignación de un recurso tomado por otro proceso, no habiendo más unidades disponibles de ese recurso para satisfacer su demanda.

6. ¿Cuáles son las estrategias utilizadas para el tratamiento del problema del interbloqueo? ¿Qué tratan de lograr estas estrategias?

Las estrategias tratan de encontrar un camino seguro en la asignación de recursos a procesos, evitando, de esta forma, el interbloqueo.

Las estrategias son: Prevención, Predicción, Detección.

7. Teniendo en cuenta el grado de concurrencia entre los procesos que permiten las estrategias para el tratamiento del interbloqueo ¿Cuál es la estrategia más conservadora, la menos conservadora y la más moderada?

La más conservadora es la Prevención, la más moderada es la Predicción y la menos conservadora es la Detección.

8. Supongamos que no puedo determinar -por anticipado- los recursos que requerirá cada proceso, en este caso, ¿Puedo utilizar la estrategia de "Predicción del Interbloqueo"?

Si no se puede determinar por anticipado todos los recursos que requerirá cada proceso entonces no puede aplicarse Prevención ni tampoco puede aplicarse Predicción a través del Algoritmo del Banquero (uso de matrices de necesidad y asignación de recursos, permite anticipar la posibilidad de interbloqueo).

En 1965 Dijstra ideó el algoritmo del Banquero, si al mismo lo aplicamos sobre un solo recurso –y solo en este caso- podemos decir que podría aplicarse aunque no supiéramos de antemano el requerimiento de este recurso por parte de cada proceso[1]. Cuando generalizamos el Algoritmo del Banquero para manejar varios recursos entre varios procesos, se requiere conocer por anticipado (antes de su ejecución) la cantidad máxima requerida de cada recurso por parte de cada proceso.[2] Debido a esta limitación, este algoritmo en la práctica es muy poco utilizado para evitar interbloqueos.

9. ¿Cómo funciona el "Algoritmo del Banquero" y en qué tipo de estrategia se usa?

Usa la estrategia de Predicción. El sistema posee un estado (la asignación actual de recursos a

[1] "Sistemas Operativos Modernos", 3ra.Ed. en Español, Tanenbaum, Pag.451-452.
[2] Idem anterior, Pag. 452-454.

procesos), el cual puede ser un estado seguro (al menos puede otorgarse una petición de asignación de recurso sin ocasionar un interbloqueo) o no seguro. El algoritmo requiere 2 vectores uno indicando los recursos existentes y la cantidad de cada uno y otro vector indicando la cantidad de disponible de cada recurso, atento a la asignación actual de recursos. La asignación actual de recursos se implementa en una matriz de recursos y procesos (matriz de asignación); por otro lado, hay otra matriz identica que guarda la necesidad de recurso por parte de cada proceso (matriz de neccesidad); el algoritmo hace la diferencia entre ambas matrices y teniendo en cuenta la disponibilidad de recurso, cuando un proceso solicita un conjunto de recursos, suponiendo que éstos se conceden, se actualizan las matrices y el sistema puede determinar si ello configura un estado seguro.

10. ¿Cuál es la estrategia que requiere de un algoritmo que se ejecute periódicamente para detectar la "condición de espera circular"?

Detección.

Respuestas Orientadoras e incompletas (en algunos casos) a las preguntas de revisión. Favor tomarlas como orientadoras.

Revisión – Unidad IV – Planificación de Procesos

1. ¿Cuál es el objetivo de la planificación?

Asignar procesos que serán ejecutados por la CPU

2. Unir con flechas:

Planif. Largo Plazo	Swapping
Planif. Mediano Plazo	Decidir ejecutar un proceso en estado listo (ready)
Planif. Corto Plazo	Decidir ejecutar un proceso bloqueado por petición E-S (I/O)
Planif. E-S (I/O)	Creación de nuevo proceso

Planif. Largo Plazo-->Creación de nuevo proceso
Planif. Mediano Plazo-->Swapping
Planif. Corto Plazo-->Decidir ejecutar un proceso en estado listo (ready)
Planif. E-S (I/O)--> Decidir ejecutar un proceso bloqueado por petición E-S (I/O)

3. Responda V o F: Es mas frecuente que un SO deba tomar decisiones de Planificación a Largo Plazo que tomar decisiones a Corto Plazo.

Falso. Las decisiones a corto plazo son mas frecuentes.

4. ¿Qué es el "grado de multiprogramación" de un sistema?

El grado de multiprogramación de un sistema es la cantidad de procesos que actualmente están cargados en memoria (en cualquier estado) y eventualmente pueden ser ejecutados.

5. ¿Por qué razones un proceso -actualmente en ejecución- podría ser expulsado para que luego se opte por ejecutar otro proceso?

Las razones de expulsar un proceso actualmente en ejecución son: cuando se crea un nuevo proceso, cuando ocurre una interrupción que permite pasar a "listo" a un proceso que estaba bloqueado o bien cuando hay una interrupción de reloj (al proceso se le termina su rodaja de tiempo).

6. ¿A qué nos referimos cuando decimos que un SO debe tener "previsibilidad"?

Nos referimos a que cualquier trabajo debe ejecutarse aproximadamente en el mismo tiempo a pesar de la carga del sistema.

7. Resulta que Juan ha instalado un SO en su computador y ha notado que a medida que incrementa su carga de trabajo (cantidad de procesos en ejecución) se degrada rápidamente el tiempo de respuesta de sus aplicaciones interactivas. Sin embargo, posee un programa para medir el rendimiento de su computador y éste indica un uso eficiente de CPU. ¿Qué tipo de criterio han tenido los diseñadores del SO instalado por Juan en cuanto al algoritmo de Planificación que éste utiliza?

Orientado al Sistema

8. Ud. ha conseguido el empleo de sus sueños, trabaja para una importante empresa y participa en el desarrollo de un nuevo SO, más precisamente en el equipo que desarrolla el planificador de este SO. El planificador hace un manejo de prioridades por proceso. Haciendo pruebas, han notado que los procesos de menor prioridad sufren inanición ¿Qué cambios haría Ud. en el planificador para solucionar este problema?

Que el planificador cambie dinámicamente las prioridades de los procesos a ejecutar según su antiguedad o histórico de ejecución

9. ¿Qué función cumple la "función de selección"?

Determina el proceso a ejecutar

10. ¿Qué significa utilizar un modo de decisión "Non-preemptive"?

Non-preemptive=Sin Expulsión, una vez que el proceso esta en ejecución, éste continua hasta que

termina o bien queda bloqueado por una operación de Entrada/Salida o una llamada al sistema.

11. ¿Qué significa utilizar un modo de decisión "Preemptive"

Preemptive=Con Expulsión, el proceso que se esta ejecutando puede ser interrumpido y pasar al estado Listo, puede ser expulsado, por los motivos (explicados en el punto 5).

12. Responda V o F: Un modo de decisión "Non-preemptive" implica una mayor sobrecarga de CPU y mejor servicio al resto de los procesos.

Falso. Implica menor sobrecarga y peor servicio al resto de los procesos.

13. Responda V o F: Un modo de decisión "Preemptive" implica una menor sobrecarga de CPU y peor servicio al resto de los procesos.

Falso. Implica una mayor sobre carga y un mejor servicio a los demás procesos.

14. Se podría decir que: "A mayor sobrecarga de CPU mejor servicio al resto de los procesos" ¿Hasta que punto podríamos afirmar esto?

Se puede afirmar esto mientras que no se exceda el nivel optimo de multiprogramación (ver concepto control de carga (load control) Unidad V)

15. ¿Qué entiende por "proceso limitado por E-S (I/O)"?

Un “proceso limitado por E/S” es aquel que pasa más tiempo utilizando los dispositivos de E/S que el procesador.

16. ¿Qué entiende por "proceso limitado por CPU"?

Un “proceso limitado por CPU” es aquel que pasa más tiempo usando la CPU que los dispositivos de E/S.

17. ¿Cuáles serían las razones por cuales decimos que la política FCFS no es viable en un sistema uniprocesador, pero sin embargo, podría aplicarse en un SMP?

FCFS no es viable en un sistema uniprocesador porque si se encola un proceso largo y otro corto, éste último deberá esperar demasiado tiempo para su ejecución. Sin embargo, podría aplicarse en un SMP tradicional, en donde por lo general, hay una única cola de procesos a ejecutar (arquitectura multiservidor) y habiendo n cpu's, el proceso más corto tendría una menor probabilidad de sufrir inanición. Esto esta respaldado por los trabajos de Saver C et. al. : a mayor número de cpu's menos importante es la sofisticación del algoritmo de planificación.

18. ¿Qué impacto tiene el tamaño de la "rodaja de tiempo" (quatum) usado en la política de "Turno Rotatorio" (Round Robin) en el rendimiento del SO?

Muy chico implica mayor sobrecarga de cpu y muy grande implica que funcione como FCFS.

19. ¿Por qué razón se desarrolló la política "Turno Rotatorio Virtual" (Virtual Round Robin)?

Debido a que RR penaliza los procesos limitados por E/S, ya que éstos -por lo general- no pueden usar el 100% de la rodaja tiempo que le fuera asignado.

20. ¿Cuál es el requisito común que tienen las siguientes políticas para ser implementadas: "Primero el proceso más corto" (Shortest Process Next, SPN), "Menor Tiempo Restante" (Shortest Remaining Time, SRT), "Primero el de mayor tasa de respuesta" (Highest Response Ratio Next, HRRN)?.

Necesitan conocer el “tiempo de servicio” de los procesos.

21. Responda V o F: La política "Retroalimentación" (Feedback) es una política no expulsiva.

Falso. La política Feedback es expulsiva, ya que expulsa a aquellos procesos cuya rodaja de tiempo (quantum) se ha acabado.

22. Responda V o F: La política "Retroalimentación" (Feedback) utiliza un manejo de prioridades dinámicas.

Verdadero. Cuando un proceso es expulsado, su prioridad se modifica (es dinámica).

23. Responda V o F: Como conclusión de esta unidad podemos decir que la política "Turno Rotatorio Virtual" (Virtual Round Robin) es la mejor de todas.

Falso. Si bien la política de Turno Rotatorio Virtual es muy buena, la decisión de que política es mejor depende del rendimiento relativo del sistema y ello depende de n factores (tiempo de servicio

de los procesos, eficiencia del algoritmo de planificación, mecanismo de cambio de contexto, demandas de E/S, rendimiento de los modulos de E/S).

24. Responda V o F: Un SMP es un multiprocesador débilmente acoplado.

Falso. Un SMP es un multiprocesador fuertemente acoplado ya que los procesadores comparten una única memoria principal.

25. ¿A qué nos referimos con la "granularidad" del sistema?

Nos referimos a la frecuencia de sincronización entre los procesos del sistema

26. ¿Qué tipo de "granularidad" tiene una aplicación multi-hilo (multithreading)?

granularidad de grano medio.

27. Responda V o F: Una asignación estática de procesos en un SMP implica una cola de procesos a ejecutar por CPU.

Verdadero.

28. Responda V o F: Una asignación dinámica de procesos en un SMP implica una única cola de procesos a ejecutar para todas las CPU's del sistema.

Verdadero.

29. ¿Que desventaja tiene una Arquitectura "Maestro-Esclavo" (Master-Slave)?

Si falla el Maestro entonces falla todo y el Maestro puede ser un "cuello de botella" para el sistema

30. ¿Que desventaja tiene una Arquitectura de "Camaradas" (Peer)?

Mas complicado de implementar, requiere sincronización en la competencia por recursos

31. Responda V o F: En una Arquitectura "Maestro-Esclavo" (Master-Slave) el planificador puede ejecutarse en cualquier CPU.

Falso. El planificador se ejecuta solamente en la CPU Maestro.

32. Responda V o F: En una Arquitectura de "Camaradas" (Peer) el planificador puede ejecutarse en cualquier CPU.

Verdadero

33. ¿Es posible estar en presencia de un SMP en donde las CPU's sean monoprogramadas?

Sí, es posible, en el caso de las aplicaciones de grano medio (app. formada por n hilos).

34. ¿Es posible estar en presencia de un SMP en donde las CPU's sean multiprogramadas?

Sí, es posible, en el caso de las aplicaciones de grano grueso, muy grueso (menor nivel de sincronización) e independientes (no hay sincronización explícita entre procesos)

35. ¿Es posible estar en presencia de un uniprocesador en donde las CPU's sean monoprogramadas?

No es posible porque si es uniprocesador no puede haber más de una cpu.

36. ¿Es posible estar en presencia de un uniprocesador en donde las CPU's sean multiprogramadas?

No es posible porque si es uniprocesador no puede haber más de una cpu.

37. Responda V o F: Una aplicacion multi-hilo (multithreading) siempre tendrá mejor rendimiento sobre un SMP con CPU's multiprogramadas.

Falso, porque es una app. de grano medio y en tal caso, no está claro si las CPU's deben ser multiprogramadas.

38. Responda V o F: Una aplicacion multi-hilo (multithreading) siempre tendrá mejor rendimiento sobre un SMP con CPU's monoprogramadas.

Falso, porque es una app. de grano medio y en tal caso, no está claro si las CPU's deben ser monoprogramadas.

39. Responda V o F: La utilización de sofisticados algoritmos de planificación, con alto nivel de sobrecarga, en un SMP siempre mejorará su rendimiento.

Falso, según los trabajos de Saver C et. al.

40. ¿Qué es una Arquitectura de cola multiservidor?

Usado en un SMP, cuando los procesos no se asignan a una única CPU, sino que hay una única cola

de procesos a ejecutar para todas las CPU's o n colas por prioridad.

41. ¿Qué probaron los estudios de Saver C. et. al.?

A mayor número de CPU's menor importancia de la sofisticación del algoritmo de planificación.

42. Supóngase que Ud. está en la situación descripta en la pregunta 8), ya ha solucionado dicho problema. Este nuevo SO puede ejecutarse tanto en equipos uniprocesador como equipos multiprocesador fuertemente acoplados. Se produce una discusión dentro del equipo que desarrolla el planificador, el dilema es: ¿El planificador debe planificar a nivel de procesos o debe planificar a nivel de hilos (threads)? ¿Qué implicancias tendría -en cuanto al desarrollo de aplicaciones para esta nueva plataforma- el optar por una u otra opción?

Cuando el SO se ejecuta en un uniprocesador puede utilizar un planificador a nivel de procesos, mientras que cuando éste se ejecuta en un SMP convendría utilizar un planificador a nivel de hilos para explotar el paralelismo real que ofrecen n CPU's. La implicancia sería que los desarrolladores deberían utilizar hilos en sus aplicaciones para explotar al máximo las capacidades de un sistema SMP.

43. ¿Qué es un Sistema en Tiempo Real? ¿Qué diferencia tiene con un Sistema que no es en Tiempo Real?

Sistema cuya corrección depende no sólo del resultado de las operaciones sino también del momento en el que se producen los resultados. Debe cumplir con tareas las cuales pueden tener plazos en los cuales deben cumplirse. Las principales diferencias: un SOTR tiene tiempos de respuesta a eventos externos muy pequeños con respecto a un SO, lo mismo en cuanto a los tiempos de servicio de una interrupción, un SOTR distingue entre tareas duras y suaves (algo que no sucede en un SO normal), un fallo en un SOTR puede significar una catástrofe mientras que en un SO normal puede significar un reboot del sistema o una degradación en los tiempos de respuesta....

44. Responda V o F: Dentro de un Sistema en Tiempo Real, lo fundamental, es su planificador a Largo Plazo.

Falso. Lo fundamental es su planificador a corto plazo.

45. Responda V o F: Ante un fallo en Sistema en Tiempo Real, lo reiniciamos y listo.

Falso. El SOTR debe tener tolerancia a fallos, debe contar con distintos modos de fallos...

46. Responda V o F: Aplicar la política de "Turno Rotatorio" (Round Robin) en un Sistema de Tiempo Real se considera una opción viable.

Falso, se tardaría mucho tiempo en atender el pedido de ejecución de un proceso en tiempo real.

47. Responda V o F: En Sistema de Tiempo Real, independientemente de la política a aplicar, ésta debería ser no expulsiva.

Falso. Un enfoque no expulsivo no es aceptable en Sistemas de Tiempo Real.

48. Responda V o F: Los Sistemas de Tiempo Real, rara vez utilizan interrupciones de reloj (clock interrupts).

Falso. Los algoritmos de planificación por lo general utilizan interrupciones de reloj (clock) como puntos de expulsión.

49. ¿Por qué razón es posible que un Sistema de Tiempo Real utilice una política de Planificación por Plazos (en vez de una basada en prioridades), a pesar de que ésta requiera contar con mayor información en cuanto a la tarea a planificar?

Porque una planificación por prioridades no garantiza la finalización o el inicio de una tarea en el momento más adecuado.

50. Responda V o F: La mayor preocupación de un Sistema de Tiempo Real es cumplir con las tareas de tiempo real suaves (soft).

Falso. La mayor preocupación es satisfacer los plazos de las tareas duras (hard) y luego el de las tareas blandas (soft).

51. ¿Es posible utilizar una planificación de Tasa Monótona en un Sistema de Tiempo Real para la planificación de tareas aperiódicas?

No, la planificación de Tasa Monótona se utiliza para tareas periódicas, puesto que esta basada en prioridades y ello no es apropiado para tareas aperiódicas que deben comenzar o terminar en un plazo determinado.

52. ¿Qué es el fenómeno de "Inversión de Prioridad"? ¿Con qué tipo de planificaciones esta relacionada? ¿Cuáles son sus posibles soluciones?

El fenómeno ocurre cuando una tarea de mayor prioridad esta forzada a esperar por una tarea de menor prioridad. Ocurre con planificaciones expulsivas basadas en prioridad. Las soluciones posibles son la herencia de prioridad (...) o el techo de prioridad (....).

Respuestas Orientadoras e incompletas (en algunos casos) a las preguntas de revisión. Favor tomarlas como orientadoras.

Revisión – Unidad V – Gestión de Memoria

1. ¿Qué es "Gestionar Memoria"?

Dividir la memoria física principal del sistema entre el SO y los procesos de usuario. Contar en memoria con procesos no ociosos: asegurar un flujo razonable de procesos listos para consumir tiempo de cpu.

2. ¿Por qué el SO debe poder reubicar un proceso en memoria?

Porque no se sabe de antemano en qué dirección de memoria va a estar un proceso al momento de su ejecución.

3. Responda V o F: La reubicación implica que debe hacerse una traducción de las referencias que están dentro del código del programa a direcciones de memoria físicas.

Verdadero.

4. Responda V o F: La reubicación facilita la implementación de la protección de memoria.

Falso. La reubicación dificulta, incrementa la complejidad de cualquier mecanismo de protección de memoria.

5. Responda V o F: El SO es quien detecta y aborta un proceso que referencia -sin permiso- posiciones de memoria fuera de su espacio de direcciones.

Falso. Es el hardware quien detecta un fallo en la protección de memoria de un proceso que será abortado por la CPU.

6. Describa la técnica de overlays.

La tecnica consiste en mantener solo una porción del programa en memoria principal y luego ir cargando y descargando los restantes modulos de la aplicación bajo demanda. Por ejemplo, una aplicación podría implementarse como un modulo principal y n modulos adicionales. El modulo principal podría estar formado por un conjunto de funciones o librería a ser utilizada en todo el sistema, más el menú principal del sistema, más datos o recursos globales a todo el sistema. Luego cada módulo adicional se correspondería con cada uno de los modulos aplicativos (clientes, proveedores, compras, ventas, etc.). Algunos linkeditores daban soporte para la linkedición de aplicaciones a través de esta técnica (ejemplo: Blinker).

7. En los tiempos del SO DOS era muy común que un programador debiera usar una técnica de overlays para lograr la ejecución de programas "grandes", ¿Por qué cree Ud. que esto era requerido?

Existen varias razones: 1. El SO DOS no tenia soporte para Memoria Virtual, por lo tanto, no podía ejecutar procesos que tuvieran un tamaño mayor que la memoria principal. 2. DOS no usaba ni paginación ni segmentación ni una combinación de ambos; reservaba un area de memoria para el sistema operativo y el resto de la memoria quedaba libre para el usuario, en donde podía cargarse el programa a ejecutar (puesto que no tenia soporte de multriprogramación, se ejecutaba un programa por vez), hacía una asignación fija de memoria, la memoria máxima que manejaba el sistema era de 640 KB (aunque el computador tuviese mucha más memoria), a ello habia que restarle lo usado por el kernel del SO; el límite estaba dado por direccionamiento, el sistema no podía representar direcciones de memoria por encima de los 640 KB, limitado por el espacio reservado a las direcciones de memoria. Es de notar, que las CPU's sobre las cuales se implementó y se usó habitualemente este sistema operativo, ya contaban con soporte para memoria virtual. Hubo software de terceros que dieron soporte a memoria extendida para intentar superar esta limitación del sistema.

8. ¿Qué entiende por "fragmentación interna"?

Se refiere al espacio desperdiciado en el interior de un bloque. Por ejemplo, suponga que dividimos

a la memoria en n bloques de longitud m y ubicamos allí al proceso p, este proceso podrá ocupar 3 bloques de longitud m más otro bloque adicional con un resto de x bytes, en donde x <= m; la fragmentación interna sería igual a m - x.

9. ¿Qué entiende por "fragmentación externa"?

Continuando con el ejemplo del punto anterior, supongamos que ubicamos un proceso p que ocupa 3 bloques, un proceso q de 2 bloques y un proceso w de 3 bloques. Si luego termina la ejecución del proceso q, quedarán 2 bloques libres en la memoria (entre medio de p y w), si luego se pretende ubicar en memoria a un nuevo proceso z que ocupa 3 bloques, si la ubicación debe hacerse en forma contigua, implica que no se pueden reutilizarse el "hueco" de 2 bloques que nos dejo el proceso q, a este hueco lo llamamos fragmentación externa.

10. Responda V o F: El particionamiento fijo tiene como ventaja que los procesos pequeños usen eficientemente el espacio ocupado en memoria.

Falso. Los procesos pequeños tendrán una alta fragmentación interna.

11. Tanto en el particionamiento fijo como en el particionamiento dinámico se habla de particiones de igual y de distinto tamaño, entonces, ¿Cuál es la diferencia entre ambos?

La diferencia es que, en el particionamiento fijo se reserva espacio para el SO y el resto de la memoria se divide en porciones fijas; mientras que en el particionamiento dinámico la memoria se va asignado a demanda de los procesos en bloques de cantidad y tamaño variable.

12. ¿Cuál es el objetivo de un "algoritmo de ubicación" (placement algorithm)?

Decidir que bloque de memoria asignar.

13. ¿Qué diferencia hay entre "marco" (frame) y "página" (page)?

Marco se refiere a las porciones (pequeñas) en que se divide la memoria física principal.
Pagina se refiere a las porciones (pequeñas) en que se divide a los procesos.
Las páginas se asignan a marcos disponibles. Un marco contiene una pagina.

14. La reubicación implica lidiar con distintos tipos de direcciones, ¿Qué tipos de direcciones de memoria Ud. conoce?

Direccion Física = dirección de memoria física dentro de la memoria principal
Dirección Base o Registro Base = dirección de memoria física inicial de un proceso
Dirección Final o Registro Valla = dirección de memoria física final de un proceso
Dirección Relativa = desplazamiento a partir de otra dirección base
Dirección Absoluta = Dirección Base o Registro Base + Dirección Relativa
Dirección Lógica = es igual que la dirección física, referencia a una posición determinada de la memoria, pero en este caso, es independiente de la actual asignación puntual en memoria.

15. Complete la Frase:

"La traducción de direcciones de memoria se hace en tiempo de ejecución de aquellos procesos que se encuentran en estado ejecutando (running), en el registro base de la CPU se carga la dirección inicial del proceso en memoria, en el registro "valla"(bound register) de la CPU se carga la dirección final del proceso en memoria. Para traducir una dirección relativa en una dirección absoluta, se toma el contenido del registro base y se suma la dirección relativa; el resultado de la suma se compara con el registro "valla" (bound register); si está dentro de los límites del proceso entonces se ejecuta la instrucción y sino la CPU genera una interrupción, asegurando de esta forma la protección de memoria."

Con alguno de los siguientes términos (que pueden usarse 0, 1 o n veces):
dirección intermedia del proceso en memoria - bloqueado (blocked) - tiempo de compilación - listo (ready) - - - - - suspendido (suspend) - - registro intermedio - dirección absoluta - dirección física - dirección intermedia - - - el SO -

13. Responda V o F: La paginación permite cargar un proceso aunque no existan marcos libres

en forma contigua.

Verdadero.

14. ¿Cómo es posible que un proceso pueda ejecutarse aunque no se encuentre en forma contigua en la memoria?

Porque el SO mantiene una tabla de paginas por proceso y las referencias a memoria son traducidas por la CPU de direcciones lógicas (número de página + desplazamiento) a direcciones físicas (número de marco + desplazamiento).

15. Indique la opción correcta. El SO mantiene una tabla de páginas por proceso, en dicha tabla: a) se accede por número de marco (frame) y se obtiene la página; b) se accede por número de página y se obtiene el número de marco.

a) Incorrecto b) Correcto.

16. ¿Por qué razón se dice que usando paginación hay un mínimo de fragmentación interna a pesar de que la paginación utiliza páginas y marcos de longitud fija?

Porque los marcos y páginas son muy pequeños.

17. Defina, aclare, diferencie los siguientes conceptos: Particionamiento, Paginación, Segmentación.

Particionamiento es una técnica antigua (previo a la memoria virtual), no usada actualmente, para dividir a la memoria física principal en bloques de longitud fija o variable (cabe aclarar que estos bloques tenían un tamaño muy superior a una pagina o marco).

Paginacion es una forma de particionamiento tanto de la memoria física principal como de los procesos en pequeñas porciones de longitud fija llamadas paginas (en cuanto a los procesos) y marcos (en cuanto a la memoria).

Segmentacion es similar a la paginación, pero en este caso los programas y sus datos se dividen en n segmentos que pueden tener distinta longitud, esto es más acorde a como se estructuran los programas más que la memoria física principal. Los modulos de codigo que conforman los programas son más bien "segmentables" que "paginables", puesto que cada programa se conforma de n modulos que pueden tener distinta longitud.

18. ¿Qué entiende por "thrasing"?

Trashing o trasiego se da cuanto el sistema pierde mas tiempo haciendo swapping que ejecutando instrucciones. Esto ocurre cuando hay muchos procesos en ejecucion y los fallos de páginas son muy frecuentes (en estos casos, se dice que esta fallando el control de carga).

19. ¿Por qué usar memoria virtual?

Porque permite independizar al programador de los límites de memoria.

Porque permite la ejecucion de programas con un tamaño mayor que la memoria principal.

Porque los programadores ya no tendrán que hacer overlays.

Porque permite ahorrar memoria cargando solo unos pocas porciones del proceso que seran usadas en un corto plazo (acorde con el principio de proximidad).

...

20. ¿Por qué razón debo traducir las referencias a memoria en tiempo de ejecución (runtime)?¿Por qué no hacerlo antes y evitar esa sobrecarga?

Porque no es posible saber de antemano en dónde quedará ubicado el proceso. Un proceso puede comenzar su ejecucion en la direccion 100 luego bloquearse, swappearse a disco y luego volver a cargarse para continuar su ejecucion a partir de la dirección 2000, hasta que no esté nuevamente ubicado en memoria y listo para continuar su ejecución no se sabe exactamente cuál será su ubicación en memoria.

21. ¿Por que razón cree Ud. que se requiere soporte de hardware para la traducción de referencias? ¿Por qué no se encarga de eso el SO?

Porque si lo hiciera el SO sería demasiado lenta la ejecución de procesos.

22. ¿Que entiende por "conjunto residente" (resident set)?

Es el conjunto de paginas que actualmente tiene un proceso en memoria. ...

23. ¿Por qué razón (vinculada con el manejo de memoria) un proceso puede quedar en estado bloqueado?

Un proceso en ejecucion puede hacer referencia a una página que no esta dentro de su conjunto residente, ello provocará un fallo de pagina, entonces el SO puede bloquear el proceso, cargar de la memoria secundaria la pagina requerida y luego volver a poner en ejecución al proceso que había dejado bloqueado.

24. ¿Qué relación hay entre "el principio de proximidad/vecindad" (locality principle) y el concepto de "thrasing"?

A medida que el conjunto residente de un proceso se hace más pequeño, acorde con el principio de proximidad, la posibilidad de fallo de página se incrementa y en la medida de que tengamos muchos procesos en memoria con conjuntos residentes demasiado pequeños, el sistema puede caer en trashing debido a la gran cantidad de fallos de pagina.

25. Responda V o F: Una implementación eficiente de memoria virtual no requiere de soporte de hardware para paginación y segmentación.

Falso. Memoria Virtual requiere de soporte de HW para paginacion y/o segmentación.

26. Responda V o F: En un SO con soporte para memoria virtual, parte de las tablas de páginas de un proceso, así como también parte del código del proceso en ejecución residen en memoria virtual.

Verdadero.

27. ¿Por qué razón Ud. cree que se divide la tabla de páginas en dos niveles jerárquicos? ¿Por qué complejizar el código y requerir de dos accesos a la tabla de páginas para traducir una dirección?

Para evitar tener que lidiar con un solo nivel jerárquico que implique un número muy grande de páginas (crecen proporcionalmente con el espacio de direcciones de memoria virtual) que difícilmente pueda mantenerse por completo en memoria principal, buscar una única entrada allí puede implicar varios accesos. Utilizando una jerarquía de varios niveles facilita el mantener en memoria ciertas porciones de la tabla de página en memoria, la busqueda en cada nivel es rápida (puesto que implica un número menor de entradas) y desde allí se apunta al siguiente nivel, dirigiendo la búsqueda hacia el valor buscado. Esta situación es análoga al planteo de implementar un índice exhaustivo (una entrada por cada valor de clave) o un índice no exhaustivo (una entrada por vencidad de clave o grupo de valores de clave).

...

28. ¿Qué relación hay entre el "control de carga" y los conceptos de "swapping" y "thrashing"?

El control de carga determina el número de procesos residentes en memoria principal (nivel de multiprogramación); muy pocos procesos en memoria puede ocasionar que -en un momento dado- todos los procesos estén bloqueados y se gaste mucho tiempo en swapping; demasiados procesos en memoria implica menos memoria principal para proceso, lo cual, puede llevar a un tamaño inapropiado del conjunto residente, provocando fallos de página muy frecuentes y ello llevará al trashing o trasiego (se pierde más tiempo haciendo swapping que ejecutando instrucciones).

29. ¿Por qué razón se utilizan las "tablas de páginas invertidas" (Inverted Table Pages)?

Debido a que el sistema tiene que lidiar con un número de páginas muy grande, se puede acelerar la búsqueda utilizando una ténica de hashing (al igual que ocurre con los archivos de acceso directo, en donde, en promedio, ubicar el registro destino esta en el orden de magnitud O(1.5)); se aplica una función de hashing que recibe como argumento el número de página a buscar y ésta devuelve la dirección en la tabla invertida, se accede a la tabla invertida, si es la pagina buscada se obtiene el marco (frame destino); si no es la página buscada puede tratarse de una página sinónima y requiere del puntero chain para acceder a otra entrada de la tabla invertida y ver si se trata ahora de la página buscada; así hasta encontrar el marco o bien generar un fallo para cargar otra porcion de la tabla

invertida a memoria principal.

30. ¿Para que sirve la función de dispersión (hash function)? ¿Cómo funciona? ¿Por qué existe un atributo llamado "chain" dentro de la tabla de páginas invertidas?

Idem anterior. Debido a como funciona un algoritmo de hashing. Idem a lo que sucede con archivos de acceso directo. La funcion de dispersión transforma clave a buscar (en este caso número de página) en un desplazamiento dentro del archivo de datos (en este caso, desplazamiento dentro de la tabla invertida). La funcion de dispersion puede implementarse de varias formas, un algoritmo posible es el del cubo, en donde la clave a buscar se divide por el tamaño del archivo y se obtiene el resto de la división como desplazamiento. El problema es que todos los algoritmos de dispersion pueden provocar claves sinónimas (2 o más paginas distintas que da como resultado el mismo desplazamiento), por ello se requiere de algún algoritmo para el manejo de claves sinónimas, una posibilidad es hacer una lista enlazada de registros con claves sinónimas, para ello se necesita del puntero chain.

...

31. ¿Cuál es el objetivo del "buffer de traducción anticipada" (Translation Lookaside Buffer (TLB))? ¿En dónde se encuentra? ¿Qué relación tiene con " el principio de proximidad/vecindad" (locality principle)?

Toda referencia a memoria virtual implica dos accesos físicos a memoria principal (1. en la tabla de páginas 2. armar la dirección física y accederla en memoria principal) el buffer de traducción anticipada (TLB) tiene como objetivo acelerar este proceso, poniendo las entradas de la tabla de páginas más recientemente usadas en una memoria de alta velocidad (cache), acorde con el principio de proximidad, se supone que éstas páginas tienen una mayor probabilidad de ser usadas en el corto plazo. De esta forma, primero se busca en el TLB y luego (si hay fallo de TLB) se busca en la tabla de páginas.

32. ¿Qué complicaciones hay vinculadas con el tamaño de la página?

Implica un dilema entre fragmentación interna Vs cantidad de entradas en la tabla de páginas. A menor tamaño de la página, menor fragmentación interna, pero ello también implica mayor cantidad de páginas requeridas.

33. Para discutir: Cuando se combinan Segmentación y Paginación, la secuencia de traducción es: Dirección Virtual - Segmentación - Paginación - Dirección Física ¿Por qué cree Ud. que esto es así? ¿Por qué no hacer: Dirección Virtual - Paginación - Segmentación - Dirección Física?

Se considera que ello es así por una cuestión de facilidad de implementación, puesto que los procesos están estructurados en términos de modulos de software y éstos son más bien "segmentables" que "paginables". Los módulos no son de longitud fija (como las páginas) sino que un proceso puede estar formado por n modulos de distintos tamaños cada uno. Por otro lado, el llevar todo en términos de páginas, que son bloques de igual longitud, facilita el desarrollo de algoritmos más sofisticados para manipularlas. Esto último sería más complicado si debiéramos trabajar con segmentos.

...

34. Responda V o F: Cualquier proceso puede accer al espacio de memoria del Dispatcher.

Falso. Fallo de protección de memoria.

35. Cuando se diseña un SO y se decide implementar soporte para memoria virtual, se deben implementar una serie de políticas relacionadas con: ¿cuándo traer una página a la memoria? ¿dónde ponerla? ¿cuando reemplazarla por otra?. ¿A qué políticas nos referimos?

¿cuándo traer una página a la memoria? --> Politica de recuperación
¿dónde ponerla? --> Politica de reubicación
¿cuando reemplazarla por otra? --> Politica de reemplazo

36. ¿Cuál es el principal objetivo de las políticas descriptas en el punto anterior?

Deben ser consideradas en el diseño de la gestión de memoria de todo SO, el objetivo principal es el rendimiento, minimizar la ocurrencia de fallos de página.

37. Si existe un "algoritmo de reemplazo optimo" (Optimal Replacement Algorithm) ¿Para qué lidiar con otros algoritmos? ¿Por qué no usar este y listo?

Debido a que el algoritmo óptimo es un algoritmo teórico (no hay implementaciones del mismo), sirve como medio de comparación para otros algoritmos.

38. ¿Qué diferencia hay entre "una estrategia de reemplazo de alcance local" (local scope replacement strategy) y una "estrategia de reemplazo de alcance global" (global scope replacement strategy)?

Ante una fallo de página, se puede optar por una estrategia de reemplazo de alcance local (selecciona páginas a reemplazar dentro de las páginas del proceso que genero el fallo) o bien de alcance global (se consideran todas las páginas en memoria principal que no estén bloqueadas, sin importar a qué proceso pertenecen).

39. ¿Cuándo debo aplicar una "política de suspensión" (suspension policy)?

Cuando debo reducir el grado de multiprogramación.

40. ¿Cuándo debo aplicar una "política de limpieza" (cleaning policy)?

Cuando debo decidir cuándo se debe escribir en disco una página que ha sido modificada.

Respuestas Orientadoras e incompletas (en algunos casos) a las preguntas de revisión. Favor tomarlas como orientadoras.

Revisión – Unidad VI – Entrada - Salida

1. ¿Cuál es la dificultad que presentan los dispositivos de I/O al momento de desarrollar software para darles soporte dentro del SO?

La gran diversidad de dispositivos de E-S antenta contra uno de los objetivos de diseño del SO en cuanto a su sistema de E-S: la generalidad. Solucion: Estructura modular genérica para implementar los drivers de los dispositivos, en la parte superior de la jerarquía los drives más genéricos y en la parte inferior los más específicos.

2. ¿Cuáles son las principales diferencias entre los dispositivos de I/O?

Velocidad de transferencia de datos, aplicación, complejidad de control, unidad de transferencia, representación de datos y condiciones de error.

3. Ordene los siguientes dispositivos de menor a mayor (en cuanto a tasa de transferencias de datos): placa ethernet de 100Mbps - Scanner - Teclado

Teclado - Scanner - placa ethernet de 100Mbps

4. ¿Qué tipos de unidad de transferencia podemos encontrar en los dispositivos de I/O?

Caracteres y bloques.

5. ¿Cuáles son las técnicas utilizadas para realizar una operación de I/O?

E/S Programada, E/S por interrupciones y DMA (Direct Memory Access).

6. En cuanto a la evolución histórica que ha tenido el tratamiento de las operaciones de I/O,¿Cuál o cuáles fueron los avances luego de DMA?

- Los modulos de E-S se transformaron en procesadores de E-S en los cuales la CPU delega la ejecución de un programa de I/O y solo se interrumpe cuando termina el programa de E-S.
- A los procesadores de E-S se les agregó memoria propia, con lo cual se convirtieron en pequeños computadores de E-S que permiten control un dispositivo de E-S con mínima intervención de la CPU.

7. ¿Cuáles son las posibles configuraciones de DMA?

- Compartiendo el mismo bus, CPU, modulos de E-S y DMA
- Integración DMA y funciones E-S
- Integración DMA y modulos de E-S usando un bus de E-S

8. ¿Por qué hablamos de "diseño jerárquico" o "diseño jerárquico por capas" en cuanto al desarrollo de software vinculado con la implementación de dispositivos dentro de un SO?

Este diseño es la solución implementada al problema de la diversidad de dispositivos, de esta forma se implementaron los drivers de los dispositivos, organizados jerárquicamente, en los niveles superiores el codigo más generico y en los inferiores el codigo más específico. Se llama también jerárquico por capas, porque cada nivel de la jerarquía puede considerarse una capa de software, puesto que tiene un objetivo determinado y una interfase de comunicación entre su capa superior e inferior.

9. ¿Cuándo podemos (y cuándo no) afirmar que el "SO x tiene una interfaz standard con los dispositivos"?

Cuando utiliza la misma API de comunicación entre los distintos dispositivos.

10. Responda V o F: La interacción con los dispositivos se hace directamente desde las aplicaciones del usuario.

Falso. La interacción con los dispositivos se hace a través del SO y no directamente desde las aplicaciones.

11. Responda V o F: El SO es un intermediario entre los dispositivos y las aplicaciones del usuario.

Verdadero.

12. ¿Qué diferencia hay entre un dispositivo local y un puerto de comunicaciones?

La diferencia está en el módulo de E/S lógica, que en el caso de un puerto de comunicaciones, se reemplaza por una arquitectura de comunicaciones de n capas (por ejemplo, TCP/IP).

13. ¿Cuál es el objetivo de hacer "I/O buffering"?

-Permitir la paginación en memoria virtual: supongamos que un programa requiere 512 de un dispositivo a partir de la dirección de memoria 1000, esto implica que las direcciones 1000 a 1511 deben permanecen en memoria hasta completarase la operacion de E-S, el programa queda bloqueado, se impediría su expulsión.
-Incrementar la eficiencia del SO y rendimiento de los procesos.

14. ¿Que relación hay entre hacer I/O usando buffers y la expulsión de un proceso en ejecución (swapping)?

Que el hecho de hacer E-S usando buffers permite la expulsión del proceso que quedará bloqueado hasta que se complete la operación de E-S; situación que no podría darse si no existieran los buffers (ver rta. anterior).

15. ¿Qué tipos de "buffering" existen?

Buffer único, buffer doble y buffer circular.

16. Cuando usamos simple (single) o doble (double) buffers, ¿Dónde se encuentran estos buffers?

En memoria principal, bajo supervisión del SO, una vez que se termina la transferencia, estos datos se mueven al espacio de direcciones del usuario (memoria del proceso del usuario).

17. Responda V o F: Se transfieren datos del dispositivo a un buffer asociado dentro del SO e inmediatamente se transfieren de este buffer al espacio de direcciones del usuario.

Verdadero.

18. Responda V o F: Es posible hacer buffering dentro del núcleo (kernel) del SO como también dentro del programa del usuario

Verdadero, ejemplo: buffer circular.

19. ¿Cuándo se pierde la eficiencia de un buffer?

Cuando hay un pico de demanda en donde la mayoría de los buffers se llenan.

20. ¿Qué efecto provoca la técnica de buffering sobre los picos de demanda de I/O?

Tienden a amortiguarlos.

21. Realice un cuadro con los tres tipos de I/O usando buffers (buffer único, doble, circular), indicando sus ventajas y desventajas.

Tipo de Buffer	Ventajas	Desventajas
Unico	-buffering por líneas o por bytes -el proceso del usuario puede ser expulsado -el proceso de usuario puede estar procesando bloque mientras se está leyendo el bloque siguiente	-si la operación de E-S se hace sobre el mismo dispositivo de swapping, prácticamente no tiene sentido -El SO debe seguir la cuenta de buffers asignados a procesos de usuarios.
Doble	-Permite que el proceso del usuario lea datos del buffer mientras el SO llena el otro buffer -Es posible mantener el dispositivo trabajando a toda su velocidad	-Mas complejo que buffer unico -En una operacion byte a byte no hay una ventaja adicional con respecto a un buffer unico de doble longitud

	-Es posible que el proceso no deba esperar (quedar bloqueado) la finalización de la operación de E-S	-No suficiente para un proceso que realice ráfagas rápidas de E-S
Circular	-Permite amortiguar (mejor que el buffer doble) los picos de demanda de E-S -Mejora la performance con respecto a buffer único	-Complejo -Si el tiempo de servicio de un dispositivo de E-S es más lento que la demanda media del proceso, no se podrá sostener el ritmo por más buffers que usemos, todos se terminarán llenado y el proceso deberá esperar por la finalización de la operación de E-S.

22. ¿Qué relación aproximada de velocidad hay entre la memoria principal y el disco?

El disco es 4 veces más lento que la memoria principal.

23. ¿Cuáles son los parámetros de rendimiento del disco?

Tiempo de búsqueda, retardo rotacional o latencia, tiempo de acceso (tiempo de búsqueda + retardo rotacional) y tiempo de transferencia.

24. Responda V o F: Tiempo de Acceso (access time) = tiempo de búsqueda (seek time) + retardo rotacional (rotational delay)

Verdadero.

25. Responda V o F: retardo rotacional (rotational delay) = latencia (rotational latency)

Verdadero.

26. Unir con flechas:
    a) tiempo que tarda la cabeza lectora/escritora en alcanzar el comienzo del sector deseado
    b) tiempo que tarda la cabeza lectora/escritora en posicionarse sobre la pista deseada
    c) tiempo que se tarda en transferir los datos
    d) rotational delay
    e) seek time
    f) transfer time

a)->d)
b)->e)
c)->f)

27. Responda V o F: El tiempo de ocupación del dispositivo X incluye al tiempo de espera por el canal de I/O del dispositivo X.

Verdadero.

28. ¿Cuál es la peor política de planificación de disco que se puede aplicar para un sistema de base de datos?

La política de Prioridad.

29. ¿Cuál podría ser una buena política para un sistema de procesamiento de transacciones?

La política de LIFO (Ultimo en entrar, primero en salir), ya que aprovecha el principio de proximidad y el movimiento del brazo es mínimo.

30. ¿Por qué usar discos redundantes (RAID redudant array of independent disks) si éstos implicarían una mayor probabilidad de fallo?

Porque éstos otorgan una mayor tasa de E-S y de transferencia. La mayor probabilidad de fallos se contrarresta con información de paridad (redundancia) que permita su recupero.

31. ¿Cuáles son las 3 características de un RAID?

1. Conjunto de discos tratados por el SO como un único dispositivo.
2. Datos distribuidos a lo largo del vector de discos
3. La redundancia se utiliza como información de paridad para garantizar su recuperación en caso de falla.

32. ¿Por qué se dice que RAID0 no es un verdadero RAID?

Porque no incluye redundancia (no cumple con punto 3.)

33. ¿Por qué se dice que un fallo en un RAID0 es catastrófico?

Porque al no incluir redundancia no hay posibilidad de recupero ante un fallo.

34. ¿Qué beneficio ofrece implementar un RAID0?

Permite hacer peticiones de E-S en paralelo cuando los bloques se encuentran en distintos discos.

35. Responda V o F: RAID1 es no aplicable porque implica grabar dos veces lo mismo y por lo tanto, gastar el doble de tiempo.

Falso. Grabar dos veces lo mismo NO implica el doble de tiempo, porque esto se realiza en paralelo, ya que un mismo bloque se encuentra siempre en dos discos distintos.

36. Responda V o F: RAID1 es un verdadero RAID porque cumple con las 3 características de un RAID.

Falso. RAID1 no incluye redundancia (no cumple con punto 3.).

37. ¿Cómo se implementa el control de paridad en un RAID1?

No hay dato de paridad, simplemente se duplican los datos a grabar en dos discos distintos.

38. Responda V o F: En un RAID2 se usa el código Hamming como control de paridad.

Verdadero.

39. Complete la frase. En un RAID2, si el disco1 se encuentra en la pista 44, entonces el disco3 se encuentra en la pista **44**

40. ¿Cuántos discos redundantes tengo en un RAID3?

Uno.

41. Supongamos que tengo un RAID3 con 5 discos, en la posicion n-ésima tengo los siguientes valores para los cuatro discos de datos: 44, 38, 25, 30; el valor para el disco de paridad es 13. ¿Cómo obtuve el dato de paridad?

Valor de paridad = 44 xor 38 xor 25 xor 30 = 13

42. Volviendo al ejemplo anterior, supongamos que se rompe el disco 3 de datos ¿Cómo obtengo el valor 25, que en esta situación, sería desconocido?

Valor del disco3 = 44 xor 38 xor 30 xor 13 = 25

43. Volviendo al ejemplo anterior, ¿Qué pasa si se rompen dos discos de datos? ¿Cómo recupero los datos?

No puedo recuperar los datos perdidos si se rompen dos discos.

44. Responda V o F: Un RAID4 penaliza las escrituras puesto que también involucra al disco de paridad y ello puede provocar un cuello de botella.

Verdadero.

45. ¿De qué forma el RAID5 evita el cuello de botella que podría ocasionarse en el RAID4?

Distribuyendo las bandas de paridad en todos los discos en forma rotatoria, de esta forma, la información de paridad esta distribuida, evita el cuello de botella en el disco de paridad y al mismo tiempo, permite grabación en paralelo.

46. Responda V o F: Un RAID6 tengo la mitad de datos redundantes con respecto a RAID5.

Falso. Tengo el doble de redundancia con respecto a RAID 5.

47. ¿Cuál es el costo que debe pagar un RAID6 para lograr "sobrevivir" ante la rotura simultanea de dos discos?

Penalizar doblemente a la escritura, se requiere comprar N+2 discos, se requieren hacer 2 calculos de paridad diferentes.

48. Responda V o F: Un RAID6 penaliza menos la escritura que un RAID5.

Falso. Penaliza el doble a la escritura.

49. Responda V o F: Para armar un RAID6 de N discos necesito agregar 1 disco más para control de paridad.

Falso. Se requieren N+2 discos.

50. Responda V o F: En UNIX/Linux los dispositivos se representan como archivos especiales sobre los cuales se puede leer y escribir.

Verdadero.

51. ¿Que tipo de I/O existe en UNIX/Linux?

Con buffer (Buffered) y Sin Buffer (UnBuffered).

52. Responda V o F: Un proceso en UNIX/Linux que esta realizando una operación de I/O sin hacer uso de buffers, es un proceso que no puede ser swapped.

Verdadero. El proceso no puede ser expulsado, puesto que se requiere de su area de memoria en donde guardar/leer los datos de la operación de I/O.

53. Responda V o F: Una Terminal RS-232 es un dispositivo orientado a bloques.

Falso. Es un dispositivo orientado a caracteres.

54. Responda V o F: Una Terminal RS-232 implica una comunicación en paralelo.

Falso. Es una comunicación en serie.

55. Responda V o F: En modo canónico la combinación de teclas CTRL+M debe ser manejado de forma especial.

Verdadero. Ctrl+M representa un retorno de carro (carriage return).

56. Responda V o F: En modo no canónico la combinación de teclas CTRL+M debe ser manejado de forma especial.

Falso. En modo no canónico los caracteres no son interpretados, se trata simplemente de un flujo de bytes.

Respuestas Orientadoras e incompletas (en algunos casos) a las preguntas de revisión. Favor tomarlas como orientadoras.

Revisión – Unidad VII – Sistema de Archivos

1. ¿Por qué se dice que los archivos son una abstracción asociada al almacenamiento secundario?

La abstracción es un recurso valioso para manejar la complejidad. Detrás de una abstracción puede ocultarse cierto grado de complejidad. La mayoría de los usuarios manejan el concepto abstracto "archivo o fichero", en la mayoría de las aplicaciones el concepto central es el "archivo" como una colección de datos, persistente, compartible entre procesos y con una estructura propia. Esta es la abstracción presentada al usuario que oculta la complejidad de implementación de esta colección de datos grabada en n bloques de un dispositivo de almacenamiento secundario.

2. Responda V o F: Todo SO debe organizar sus archivos bajo una estructura jerárquica.

Falso. Existen muchos SO's (CP/M, VMS, MVS, etc.) que presentan una abstracción de su sistema de archivos -que, si bien esta organizada bajo la abstracción "archivo"- en forma no jerárquica.

3. Responda V o F: El concepto de archivo esta asociado con la persistencia de datos.

Verdadero.

4. ¿Cuál es la función principal de un Sistema Gestor de Ficheros?

Proporcionar servicios a usuarios y aplicaciones para el uso de ficheros.

5. ¿Cuáles son las operaciones típicas que se aplican sobre ficheros?

crear() ..., borrar() ..., abrir() ..., cerrar() ..., leer() ..., escribir() ....

6. Defina los siguientes conceptos: Campo, Registro, Fichero, Base de Datos.

Campo: elemento básico, valor único, de longitud fija o variable, de un tipo de dato determinado.
Registro: conjunto de campos relacionados que se pueden tratar como la representación de una unidad, de longitud fija o variable.
Fichero: Colección de registros, de campos similares, con restricciones en cuanto al control de acceso (permisos), entidad única desde el punto de vista del usuario y las aplicaciones, referenciada por nombre.
Base de Datos: Colección de datos relacionados, formada por uno o más ficheros, diseñada para ser usada por una o más aplicaciones, se requiere de un software adicional para su manipulación: SGBD (sistema gestor de base de datos) separado del SO.

7. ¿Qué ventajas le provee al desarrollador de aplicaciones el trabajar con un SO que posea un Sistema de Gestión de Ficheros?

Le permite concentrarse en el problema a resolver en vez de ponerse a escribir código para lidiar con los dispositivos, el espacio ocupado y libre en disco, la planificación de los dispositivos, etc. El SGF provee una interfase uniforme (E-S lógica) que le permite al usuario y a las aplicaciones acceder a los registros de los archivos en forma lógica, ocultando los detalles de implementación.

8. Responda V o F: El Sistema Gestor de Ficheros no se ocupa de la administración de permisos sobre los ficheros.

Falso. Dentro del soporte para la compartición de archivos (entre distintos usuarios) que debe proporcionar el SGF se encuentran los derechos de acceso.

9. Responda V o F: Los drivers tienen un nivel de abstracción mayor que la I/O lógica.

Falso. Los drivers son los de menor nivel de abstracción, puesto que interactúan con el hardware (en términos de señales, registros de cpu, canales, etc.); mientras que la E-S lógica interactúa con los usuarios y las aplicaciones (en términos de archivos, registros lógicos).

10. Responda V o F: Los programas del usuario interactúan directamente con los drivers.

Falso. Los programas interactúan con la E-S lógica.

11. Responda V o F: El sistema básico de I/O trabaja a nivel de bloques.

Verdadero. Realiza la E-S física, manejo de buffers, etc.

12. Responda V o F: El supervisor de I/O básico trabaja con las políticas de planificación de I/O.

Verdadero. Inicia y termina las peticiones de E-S, planificación de E-S, colas, estados de ficheros, asignación de buffers.

13. Responda V o F: Los datos básicos de un fichero son mantenidos por el sistema de I/O lógica.

Verdadero. El sistema de E-S lógica trabaja con los usuarios y aplicaciones permitiendo el acceso a archivos y registros lógicos; así como también mantiene los datos básicos de los ficheros. Pag.552.

14. ¿Qué es un método de acceso? ¿Para qué sirve?

Es el nivel de la arquitectura del sistema de ficheros más cercano al usuario. Sirve para proporcionar una interfaz estándar entre las aplicaciones, sistemas de ficheros y dispositivos de almacenamiento. Es la forma de manipular la estructura física del archivo, cómo leer, cómo grabar un registro lógico; existen distintas formas de hacerlo según el método de acceso utilizado. Esta vinculado con las distintas estructuras de los archivos.

15. ¿A qué nos referimos con la "organización de un fichero"?

Idem anterior. Estructura lógica de los registros determinados por su forma de acceso.

16. ¿Qué relación hay entre la "organización de un fichero" y los "métodos de acceso"?

La organización de un fichero se refiere a su estructura física mientras que el método de acceso se refiere al algoritmo que debe aplicarse -acorde con la estructura física del archivo- para manipular los registros del archivo. Por ejemplo, si un archivo tiene una organización de tipo Hash, el método de acceso debe ser a través de un valor clave el cual será traducido a una dirección de registro lógico utilizando para ello una función de dispersión. ...

17. Responda V o F: En cuanto a la administración de ficheros, el SO trabaja a nivel de bloques físicos (registros físicos) mientras que las aplicaciones de usuario trabajan a nivel de registros lógicos.

Verdadero.

18. ¿Qué diferencia hay entre registro físico y registro lógico?

Registro físico: son la unidad de transferencia entre el dispositivo de almacenamiento y la memoria principal. Un dispositivo de almacenamiento secundario lee y graba "de a registros físicos" o bloques físicos.

Registro lógico: es el registro de un archivo desde el punto de vista del programador, formado por n campos que describen a una entidad determinada (artículo, proveedor, cliente, etc.). Un registro físico -según su tamaño- podrá contener 0, 1 o n registros lógicos más algunos bytes de desperdicio (fragmentación interna).

19. Describa la organización de archivo de tipo pila (stack). ¿Qué características tiene?

Es la organización más sencilla, el orden de los registros equivale al orden de llegada, los registros pueden ser de longitud variable, con igual cantidad de campos o distinta cantidad, lo cual implica el uso de delimitadores de registros/campos y/o campos autodescriptivos. Acceder a un registro determinado implica una búsqueda exhaustiva, organización inadecuada para la mayoria de las aplicaciones, generalmente se usa como una forma de almacenamiento previo a su proceso.

20. Responda V o F: En una organización de tipo pila puedo procesar registros en forma directa.

Falso. El acceso requiere de una búsqueda exhaustiva, secuencial.

21. Describa la organización de archivo de tipo secuencial. ¿Qué características tiene?

Registros de longitud fija, igual cantidad de campos, no autodescriptivos, posicionales, conjunto de 1 o más campos que forman clave primaria, almacenamiento según clave primaria, procesamiento batch, ideal para almacenamiento en cinta o disco, el orden físico de los registros coincide con el orden lógico. Generalmente se actualizan mezclándose con otro fichero de tipo pila, obteniéndose un nuevo archivo secuencial actualizado.

22. Responda V o F: En una organización de tipo secuencial el o los campos clave no tienen ninguna importancia.

Falso. Ya que los registros se guardan físicamente según la clave.

23. Responda V o F: Un archivo secuencial indexado, al igual que un archivo secuencial se encuentra ordenado por uno o más campos clave.

Verdadero. Ambas organizaciones mantienen la misma estructura, pero el secuencial indexado agrega dos estrcturas adicionales: un archivo índice que permite acceso aleatorio por clave y un archivo de desbordamiento.

24. ¿Qué ventajas adicionales provee la organización secuencial indexada por sobre la organización secuencial?

Permite acceso aleatorio por clave.

25. Responda V o F: Una archivo secuencial indexado ocupa menos espacio en disco que un archivo secuencial.

Falso. Puesto que requiere dos archivos adicionales.

26. ¿Qué es un índice?

Es una archivo adicional al archivo de datos. Tiene la estructura de un archivo secuencial, con -al menos- dos campos: el campo clave y un puntero a una dirección de registro dentro del archivo de datos (también este puntero puede utilizarse como indirección a otro nivel dentro del mismo índice).

27. ¿Qué es un índice exhaustivo?

Es un índice que tiene una entrada por cada valor de clave.

28. ¿Qué es un índice no exhaustivo?

Es un índice que tiene una entrada por cada grupo de valores de clave (permite apuntar hacia la "vecindad" de la clave buscada). Pueden incluirse sólo algunas claves en el índice (para reducir su tamaño) y luego apuntar hacia otro nivel o bien hacia la vecindad de la clave buscada en el archivo de datos y continuar con una búsqueda secuencial a partir de allí.

29. Responda V o F: Para mejorar el rendimiento de un archivo secuencial indexado voy a crear una clave por cada atributo.

Falso. Porque a medida que se agregan índices se penalizan las grabaciones, puesto que, además de actualizar los datos del registro también se deben actualizar los índices.

30. Responda V o F: Un archivo secuencial indexado puede contener registros de longitud variable.

Falso. Todos los registros deben ser de longitud fija.

31. ¿Por qué hay necesidad de un "archivo de desborde" (overflow file) en un archivo secuencial indexado?

Para enlazar (y mantener en forma ordenada por el campo clave) los nuevos registros que se agregan en el archivo.

32. ¿Cómo funciona un archivo de acceso directo?

Tiene una estructura similar a un archivo secuencial, pero sin mantener un orden por clave primaria. Este tipo de archivos es accesible en forma aleatoria, a través del número de registro lógico (0 .. N-1). Generalmente se utiliza combinado con una tecnica de hashing, utilizando una funcion de dispersión para lograr un acceso a través de clave primaria.

33. ¿Cómo funciona un archivo hash (hashed file)?

Idem anterior. Stallings asume que un archivo de acceso directo es sinónimo de un archivo hash. En este caso se implementa una función hash (o función de dispersión) que transforma una clave primaria en dirección de registro lógico. Son archivos muy rápidos para acceder por clave primaria, no requieren de ninguna estructura de datos adicional al archivo de datos. El problema habitual es que debe implementarse una forma de dar soporte a las claves sinónimas (dadas dos claves primarias distintas resulta que ambas, luego de aplicarles la función hash, apuntan al mismo registro lógico).

34. ¿Cuáles son las posibles implementaciones de un archivo hash?

Siguiendo con el punto anterior. Existen dos implementaciones posibles: hashing lineal y desbordamiento encadenado. En el primer caso, ante una colisión, se continúa dentro del archivo de datos secuencialmente hasta encontrar un lugar libre donde almacenar el registro (esto tiene el efecto adverso de poder generar más colisiones en forma indirecta). En el segundo caso, se agrega al archivo de datos un campo adicional que sera un puntero hacia el primer registro colisionado. En todos los registros que no tengan colisiones, este campo estará vacío. Entonces, cuando ocurre una colisión, los registros colisionados se mantienen -a partir de un area de desborde (en un archivo separado o a partir del registro n + 1)- como una lista simplemente enlazada.

35. ¿Qué es un directorio? ¿Qué contiene?

Es una proyección entre los usuarios, las aplicaciones y los ficheros. Mantiene información sobre cada archivo del sistema, incluye información básica (nombre,tipo,organización), información de direccionamiento (volumen/disco,direccion física inicial,tamaño,tamaño asignado), información de control de acceso (dueño,claves de acceso,permisos), información de uso (fecha creación,usuario creador,fecha ultima lectura,usuario ultimo lector,fecha ultima actualizacion,usuario ultimo actualizador,fecha ultima copia de seguridad,uso actual (info.sobre el uso actual, que proceso tiene este archivo abierto, si esta bloqueado, si esta actualizado en memoria y aun no en disco,etc.)).

36. ¿Qué operaciones pueden aplicarse sobre un directorio?

buscar(), crear fichero(), borrar fichero(), listar directorio(),actualizar directorio().

37. ¿Cuál es la información de control asociada a un directorio?

Información de control de acceso (dueño,claves de acceso,permisos).

38. ¿Quién es el dueño (owner) de un archivo y qué operaciones puede hacer sobre un archivo de su propiedad?

El dueño del fichero es su propietario, su creador y puede hacer cualquier tipo de operación sobre un archivo de su propiedad.

39. ¿Cómo se identifican unívocamente los archivos dentro de un sistema de ficheros jerárquico?

A través del nombre de camino (path, conjunto de directorios hasta el directorio que contiene al archivo) + nombre del fichero. Ejemplo, el archivo myfile se encuentra en el directorio /home/grchere por lo tanto, unívocamente el archivo se identifica como /home/grchere/myfile.

40. ¿Qué es el directorio actual (current directory) o directorio de trabajo (working directory)?

Permite hacer referencia en forma relativa a un archivo o directorio. El directorio actual o directorio de trabajo es un nombre de camino (path) asociado con el usuario interactivo actual o con un proceso actualmente en ejecucion. En forma interactiva, en una consola unix/linux, el usuario puede conocer su directorio actual utilizando el comando "pwd" quien muestra en pantalla el directorio actual.

41. ¿Que implicancias tiene el hecho de que un archivo sea compartido por n procesos ejecutados por n usuarios?

El SO debe aplicar algun tipo de disciplina sobre el fichero utilizando algún tipo bloqueo (a nivel del fichero completo o a nivel de registros individuales) para implementar una exclusión mutua entre distintos procesos que pretenden acceder al mismo fichero.

42. ¿Qué clases de permisos sobre ficheros Ud. conoce?

ninguno ..., conocimiento ..., ejecucion ..., lectura ..., adición ..., actualización ..., cambio de protección ..., borrado ... .

43. ¿Qué clases de usuarios Ud. conoce sobre los cuales se pueden aplicar distintos accesos?

Usuario específico, Grupo de usuarios y Todos los usuarios.

44. ¿Qué tipo de bloqueos (locks) pueden aplicarse sobre un archivo para garantizar exclusión mutua?

Bloqueo a nivel de archivo o bloqueo de registros individuales.

45. El Sistema de Ficheros debe gestionar el espacio de almacenamiento secundario. Un bloque físico es una unidad de I/O de almacenamiento secundario, existen 3 enfoques en cuanto al tipo de bloques físicos. Indique cuáles son estos 3 enfoques y cuáles de ellos son incompatibles con las implementaciones tradicionales de memoria virtual.

1.Bloques fijos: ...
2.Bloques expandidos de longitud variable: ...
3.Bloques no expandidos de longitud variable: ...

2 y 3 son incompatibles con las implementaciones tradicionales de memoria virtual.

46. Al asignar bloques a un fichero, hay 2 enfoques: el estático y el dinámico. Describa brevemente cada uno de ellos y qué relación guardan en cuanto a la cantidad de bloques a asignar a un nuevo archivo.

La asignación estática de bloques implica pre-asignación, es decir, que los usuarios, las aplicaciones deberían indicar de antemano cuánto espacio ocupará un fichero, lo cual podría llevar a un malgasto de espacio por sobre-estimación.
La asignación dinámica asigna bloques a medida que lo necesita el fichero.
En ambos casos, el dilema común se encuentra en estimar el tamaño de porción apropiado, ¿qué cantidad de bloques asigno a un fichero?, es una tensión entre la eficiencia del fichero Vs eficiencia del sistema. Tener espacio contiguo incrementa el rendimiento del fichero (pero puede ocasionar que el SO agote los espacios contiguos requeridos para otras asignaciones), muchas porciones pequeñas incrementa el tamaño de las tablas necesarias para gestionar la asignación, porciones de tamaño fijo simplifica la re-asignación de espacio, porciones variables o pequeñas de tamaño fijo minimiza el espacio malgastado.

47. Existen 3 métodos de asignación de ficheros (file allocation): Asignación Contigua, Encadenada o Enlazada e Indexada. Describa brevemente cada uno de ellos.

1.Asignación Contigua: Pre-asignación.Se asigna un conjunto contiguo de bloques y existe una tabla de asignación de ficheros que contiene el nombre del fichero, el bloque inicial y la longitud. Se producirá fragmentación externa. ....
2. Asignación Encadenada: Con o sin pre-asignación. Se asignan bloques individuales, donde cada uno contiene un puntero al siguiente elemento (fichero=lista simplemente enlazada de bloques), es decir, no están contiguos. Tabla de asignación de ficheros contiene nombre de fichero y puntero al bloque inicial de la lista. No hay fragmentación externa. Asignación sencilla. ....
3. Asignación Indexada: La tabla de asignación de ficheros contiene nombre de fichero y puntero al bloque indice, el cual a su vez, contiene los bloques donde se encuentra el fichero. Elimina la fragmentación externa y aprovecha el principio de proximidad. Permite usar porciones de tamaño fijo o variable....

48. ¿Qué es "la tabla de asignación de ficheros" (file allocation table)?

La tabla de asignación de ficheros es una estructura de datos (tabla) que guarda la “traza” de las porciones de memoria secundaria asignadas a cada fichero.

49. ¿Por qué razón se produce "fragmentación externa" al utilizar asignación contigua?

Porque cuando se borra un archivo quedará un "hueco" de bloques libres, si dicho hueco se encuentra entre dos archivos, ese espacio no puede ser reutilizado eficientemente a menos que se vuelva a asignar un nuevo archivo de exactamente la misma cantidad de bloques que el archivo borrado. Esto puede llegar al extremo de que un sistema tenga miles de bloques libres en "huecos" pequeños generados entre archivos y no pueda asignar espacio contiguo de un nuevo archivo aunque éste no supere el espacio libre. En tal caso, el disco necesitará de un proceso de defragmentación para "reacomodar" todos los archivos a partir del comienzo del disco, dejando los bloques libre al final.

50. ¿Por qué razón la asignación encadenada o enlazada no tendría buen rendimiento sobre un archivo de acceso directo?

Porque ir al i-esimo bloque implica recorrer la lista enlazada hacia adelante a partir de un bloque inicial y así sucesivamente para cada petición. Obtener el i-esimo bloque puede implicar la lectura de muchos bloques.

51. ¿Por qué razón se necesita hacer una compactación o una consolidación del disco luego de cierto tiempo de uso?

Para transformar el espacio libre en espacio libre contiguo y al añadir nuevos ficheros a éstos se les podría otorgar bloques contiguos favoreciendo el principio de proximidad.

52. ¿Existe alguna diferencia dentro de la tabla de asignación archivos cuando usamos una asignación indexada utilizando porciones fijas que cuando usamos porciones variables?

No, en ambos casos la FAT solo guarda el puntero al bloque indice. Si cambia la estructura del bloque indice que en el caso de las porciones variables se debe agregar además del número de bloque, su longitud.

53. Gestionar el espacio en disco implica gestionar el espacio libre y ocupado. La tabla de bits es un método posible para gestionar espacio libre. Indique cuál es su principal desventaja.

La principal desventaja es que el tamaño de la tabla de bits crece proporcionalmente al tamaño del disco, ocupando mucho espacio y haciendo las búsquedas muy lentas, incluso aunque la misma se cargue en memoria principal. La mayoría de los SO que usan esta técnica utilizan estructuras de datos auxiliares en memoria (a modo de resumen de la tabla de bits) para acelerar la ubicación de espacio libre.

54. En UNIX/Linux, una estructura de tipo inode esta asociado a un único archivo, entonces ¿Por qué se dice que puede haber muchos nombres de archivos asociados a un mismo inode?

Debido a que en Unix (al igual que en Windows) se puede enlazar una o más entradas del directorio a un archivo físico determinado. Cada i-node representa un único archivo físico, pero sobre un archivo físico puede haber n enlaces. Cada enlace tendrá una entrada el directorio que estará apuntando a un mismo i-nodo. Fig. 12.14 Pag. 579 (podría haber alguna entrada del directorio apuntando al mismo i-nodo).

55. Responda V o F: En UNIX/Linux los directorios son estructuras especiales, distintas a un archivo normal.

Falso. Un directorio es un archivo normal cuyo contenido es una lista de nombres (que representan archivos u otros directorios) más un número que indica un desplazamiento dentro de la tabla de nodos-i.