1.-Con respecto al apoyo hardware al SO:

a) Explica los modos de direccionamiento de memoria de la arquitectura IA32: modelos de memoria flat, segmented y real-address mode

Los programas no acceden directamente a la memoria física sino indirectamente usando los modelos de memoria:
Modelos de memoria flat: Es un espacio lineal con direcciones consecutivas en el rango [0,2^32-1]. Permite direccionar con granularidad de un byte.
Segmented memory model: Los programas ven el espacio de memoria como un grupo de espacios independientes llamados segmentos.
Real-Address mode memory model: Implementación de segmentación con limitaciones en el tamaño de los segmentos, 64KB, y en el espacio de memoria final accesible, 2^20 bytes.

b) Describe los pasos, hardware y software, que se llevan a cabo para la resolución de una llamada al sistema
Si se produce una llamada al sistema, en primer lugar, se busca el número asociado a dicha llamada en el sistema, el procesador cambia el bit de modo a modo kernel y se invoca al manejador de llamadas del sistema, que salva los registros del procesador en la pila del sistema para luego poder reanudar la ejecución del programa. A continuación, el procesador carga en el PC la primera instrucción de dicha llamada y se dispone a procesarla. Hasta aquí todo lo realiza el hardware.
A partir de aquí todo lo realiza el software. Pueden ocurrir dos cosas, que la llamada sea bloqueante o que no lo sea. En el primer caso, el
proceso se bloquea a espera de un determinado evento. En el segundo caso, devuelve un valor que determina la condición de finalización de dicha llamada, y se restauran los valores salvados en la pila para poder seguir ejecutando el proceso por donde lo había dejado.

Otra opción:

Si se produce una llamada al sistema, en primer lugar, se busca el número asociado a dicha llamada en el sistema, el procesador cambia el bit de modo a modo kernel y se invoca al manejador de llamadas del sistema, que salva los registros del procesador en la pila del sistema para luego poder reanudar la ejecución del programa. A continuación, el procesador carga en el PC la primera instrucción de dicha llamada y se dispone a procesarla.

Pueden ocurrir dos cosas, que la llamada sea bloqueante o que no lo sea. En el primer caso, el proceso se bloquea a espera de un determinado evento. En el segundo caso, devuelve un valor que determina la condición de finalización de dicha llamada, y se restauran los valores salvados en la pila para poder seguir ejecutando el proceso por donde lo había dejado.

La primera parte lo realiza el hardware, y la segunda, el software.

2.- Con respecto a la virtualización:

a) ¿Cual es la diferencia entre los dos enfoques de virtualización explicados en clase hipervisor tipo 1 e hipervisor tipo 2?

Tipo 1, native o bare-metal: Se ejecutan directamente en el host HW para proporcionar VM a los SO invitados. Ej. Xen y Vmware ESX/ESXi.

Tipo 2, hosted: Se ejecutan sobre un SO convencional como el resto de programas y el SO invitado se ejecuta sobre la abstracción proporcionada por el hipervisor. Ej. VMware Workstation y VirtualBox .

Los hipervisores tipo 1 obtienen mejor rendimiento que los tipo 2 ya que disponen de todos los recursos para las VM y los múltiples niveles de abstracción entre el SO invitado y el HW real no permiten alto rendimiento de la máquina virtual. Los hipervisores tipo 2 permiten realizar virtualización sin tener que dedicar toda la máquina a dicho fin. Ejemplo, prueba de kernels o puesta a punto de servidores.

b) **Describe en qué consiste la técnica de virtualización denominada virtualización asistida por hardware. Básese para la explicación en la arquitectura x86 y los rings del procesador.**

La Virtualización Asistida por Hardware, como su propio nombre indica, son extensiones introducidas en la arquitectura del procesador x86 para facilitar las tareas de virtualización al software corriendo sobre el sistema. Si cuatro son los niveles de privilegio o "anillos" de ejecución en esta arquitectura, desde el 0 o de mayor privilegio, que se destina a las operaciones del kernel de SO, al 3, con privilegios menores que es el utilizado por los procesos de usuario, en esta nueva arquitectura se introduce un anillo interior o ring -1 que será el que un hypervisor o Virtual Machine Monitor usará para aislar todas las capas superiores de software de las operaciones de virtualización.

3.- Responda a las siguientes cuestiones sobre el concepto de hebra:

a)¿Qué ventajas proporciona el modelo de hebras frente al modelo de proceso tradicional?

- Se reduce el tiempo de cambio de contexto entre hebras, así como el tiempo de creación y terminación de hebras.
- Mientras una hebra de una tarea está bloqueada, otra hebra de la misma tarea puede ejecutarse, siempre que el kernel sea mutithreading.
- Usan los sistemas multiprocesador de manera eficiente y transparente, a mayor número de procesadores, mayor rendimiento.
- La comunicación entre hebras de una misma tarea se realiza a través del espacio de direcciones asociado a la tarea por lo que no necesitan utilizar los mecanismos del núcleo.

b) ¿Cuál es el inconveniente en la implementación de hebras de usuario a la hora de que se realice una llamada al sistema bloqueante por parte del programa?

Si nos encontramos con hebras en biblioteca de usuario (user-level threads ó ULT), tendremos que si se produce una llamada al sistema bloqueante por parte de una hebra, el resto de hebras del mismo proceso se bloquean también.

c) Justifique el grado de paralelismo real alcanzado por una aplicación con varias hebras, teniendo en cuenta que los programas utilizan una biblioteca de hebras a nivel usuario y el núcleo no planifica hebras sino procesos.

Como el kernel planifica procesos, solo puede asignar un proceso con todos sus hilos correspondientes a un solo procesador. Entonces, los hilos de un mismo proceso nunca podrán ejecutarse en paralelo. No

obstante, si la aplicación consta de varios procesos, entonces lograremos paralelismo a nivel global de la aplicación, pero no desde el punto de vista de los hilos de sus procesos.

4.- ¿Cómo implementa Linux el concepto de hebra? Explíquelo utilizando el PCB de Linux (struct task_struct) y llamada al sistema clone().

Desde el punto de vista del kernel, no hay distinción entre hebra y procesos. Linux implementa el concepto de hebra como un proceso sin más, que simplemente comparte recursos con otros procesos. Cada hebra tiene su propia task_struct, que es una estructura que engloba el concepto de proceso y hilo. La llamada al sistema clone() crea un nuevo proceso o hebra dependiendo del parámetro flags. A diferencia de fork, esta llamada permite al proceso hijo compartir partes de su contexto de ejecución con el proceso invocador, tales como el espacio de memoria, la tabla de descriptores de fichero y la tabla de manejadores de señal.
A veces, es útil que el kernel realice operaciones en segundo planto, para lo cual se crean hebras kernel- Las hebras kernel no tienen un espacio de direcciones y por lo tanto su puntero mm es NULL. Estas hebras son ejecutadas únicamente en el espacio del kernel. Son planificadas y pueden ser expropiadas. Terminan cuando realizan una operación do_exit o cuando otra parte del kernel provoca su finalización.

5.- En un SO con una política de planificación apropiativa, enumere las distintas partes del SO que deben comprobar la posibilidad de desplazar al proceso que actualmente se está ejecutando y proponga un pseudocódigo que describa cómo se realizaría dicha comprobación en cada parte.

Hay un total de 4 situaciones:

1. El proceso finaliza su ejecución. En este caso el planificador (schedule()) es llamado dentro de la rutina de núcleo que finaliza un proceso (sys_exit() o do_exit()).

2. El proceso realiza una llamada bloqueante. En este caso, después de una llamada a E/S o una llamada wait(), se llama a schedule().

3. El proceso agota su rodaja de tiempo (es interrumpido por la señal de reloj). La llamada a schedule() ocurre durante la interrupción de RSI_reloj().

4. Se añade un proceso nuevo a la cola de Listos. Una vez que el SO ha creado el proceso nuevo o un proceso ha recibido la señal que lo desbloquea o se haya movido a memoria principal (en resumen, pasa a cola de Listos), se llama a schedule(), esto suele hacerlo el planificador a largo o medio plazo, excepto en el caso de que se desbloquee un proceso

6.- Con respecto a la planificación de procesos responda las siguientes cuestiones:

a) ¿Qué algoritmo de planificación provoca una mayor penalización a los procesos limitados por E/S frente a los procesos limitados por CPU? ¿Por qué?

El FCFS, ya que con procesos largos que no incluyan E/S, el proceso se ejecuta de una sola vez. Sin embargo, en los procesos cortos que tengan E/S, el proceso se bloqueará por cada una de las E/S que tenga, aumentando con creces el tiempo de espera y la pnealización con

él.

b) Describa los factores a considerar a la hora de diseñar un algoritmo de planificación basado en colas múltiples con retroalimentación. En particular, justifique como asociaría los conceptos de quantum y prioridades a su diseño.

La cola de listos se divide en varias colas y cada proceso es asignado permanentemente a una cola concreta
P. ej. Dos colas: Procesos interactivos y procesos batch. Cada cola puede tener su propio algoritmo de planificación P. ej. Interactivos con RR y procesos batch con FCFS. Un proceso se puede mover entre las
distintas colas. Para implementar un algoritmo de prioridades, optamos por la asignación de prioridades fijas, pues no existe problema de inversión de prioridad al tener retroalimentación. En cuanto al quantum,
tendremos en cuenta que no puede ser muy grande (se convertiría en FCFS), ni muy corto (pues emplearíamos mucha CPU en múltiples cambios de contexto).

7.- Con respecto al núcleo de Linux visto en clase:
a) Describa los pasos que ejecuta el núcleo de Linux en la función do_exit().

En primer lugar, se activa el flag PF_EXITING de task_struct. Para cada recurso que esté utilizando el proceso, se decrementa el contador correspondiente que indica el número de procesos que lo están utilizando. Si contador==0, se realiza la operación de destrucción oportuna sobre el recurso. Despúes, el valor que se pasa como argumento a exit() se almacena en el campo exit_code de task_struct (esta es la información de terminación para que el padre pueda hacer un wait() o waitpid() y recogerla). Luego, se manda una señal al padre indicando la finalización de su hijo. Si el proceso aún tiene hijos, se estable como padre de dichos hijos al proceso init con PID = 1 aunque esto depende de las características del grupo de proceso al que pertenezca el proceso. Para finalizar, se establece el campo exit_state de task_strcut a EXIT_ZOMBIE y se llama a schedule() para que el planificador elija un nuevo proceso a ejecutar.
Nota: do_exit() nunca retorna ya que es el último código que ejecuta un proceso.

b) Describa como se comporta el planificador de Linux, schedule(), para los procesos planificados mediante la clase de planificación CFS(Complete Fair Scheduling).
-Actuación de schedule ():

1. Determina la actual runqueue y establece el puntero previo a la task_struct del proceso actual.
2. Actualiza estadísticas y limpia el flag TIF_NEED_RESCHED.
3. Si el proceso actual va a un estado TASK_INTERRUPTIBLE y ha recibido la señal que esperaba, se establece su estado a TASK_RUNNING.
4. Se llama a pick_next_task de la clase de planificación a la que pertenezca el proceso actual para que se seleccione el siguiente proceso a ejecutar; y se establece next con el puntero a la task_struct de dicho proceso seleccionado.
5. Si hay cambio en la asignación de CPU, se realiza el cambio de contexto llamando a context_switch() (Nuestro dispatcher()).