1 Con respecto al apoyo hardware al SO:

a.- Explica los modos de direccionamiento de memoria de la arquitectura IA-32: modelos de memoria flat, segmented y real-address mode.

- Memoria flat: en este modelo de gestión de memoria, se divide toda la memoria principal en palabras y se direcciona por palabras. En IA-32 se pueden direccionar 2^{32} palabras de un byte.
- Segmented: en este modelo la memoria se divide en segmentos de cierta longitud y para direccionar se emplean dos datos: el número de segmento y el *offset*. IA-32 permite identificar aprox. 2^{14} segmentos de 2^{32} bytes.
- real-address mode: es un modo de gestión implementado por razones de retrocompatibilidad con el procesador 8086 y que divide la memoria en segmentos de 64KB, aproximadamente 2^{20} bytes en total.

b.- Describe los pasos, hardware y software, que se tienen que llevar a cabo para la resolución de una llamada al sistema.

Primeramente, el proceso mediante una instrucción genera una interrupción específica.

- Hardware: la señal de interrupción llega al procesador, que termina su ciclo de ejecución de intrucción y entonces comprueba la señal, la reconoce, guarda el contexto del programa en ejecución (PSW y PC) en la pila de control del núcleo, activa modo kernel y activa el manejador de interrupciones pasándole el código de la interrupción. Dado que es una llamada al sistema, el manejador de interrupciones carga en el PC la dirección de la primera instrucción del manejador de llamadas al sistema, que se comporta en realidad como una RSI.
- El manejador de llamadas al sistema salva los registros que necesite, revisa qué tipo de llamada se está realizando (siguiendo las convenciones marcadas en cada arquitectura) y activa la secuencia de instrucciones correspondiente que ejecuta la llamada al sistema, y devuelve el control al programa o llama al planificador a corto plazo (en caso de que el proceso se vaya a quedar en estado bloqueado). Se restauran los registros y se restaura PC y PSW y el procesador continúa. En el caso en que se llame al planificador, este se encarga de realizar el cambio de contexto.

2 Con respecto a virtualización:

a.- ¿Cuál es la diferencia entre los dos tipos de enfoque de virtualización explicados en clase: hipervisor tipo1 e hipervisor tipoo2?

La principal diferencia es que el hipervisor tipo 1 permite alojar diferentes SO huéspedes directamente sobre el hardware (sin necesidad de un SO anfitrión), es decir, el hipervisor se ejecuta directamente sobre el hardware anfitrión. Mientras que el hipervisor tipo 2 requiere un sistema operativo anfitrión que lo ejecute como si se tratase de un programa más, y dentro del software del hipervisor se realiza la virtualización. Por lo tanto este segundo hipervisor interpone más capas de software entre la máquina huésped y el hardware que ejecutará dicha máquina.

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?

El modelo de hebras permite por una parte explotar el paralelismo, permitiendo que el código de un proceso se ejecute en paralelo y teniendo un acceso más rápido y sin necesidad de cambio de contexto a los datos que comparten diferentes hebras. Además permite modularizar los procesos, ya que cada hebra puede cambiar de contexto con hebras del mismo proceso sin ser esta una operación costosa. También permite que una hebra bloqueada por E/S no detenga el proceso entero (en ciertas implementaciones). A esto se le añade el procesamiento asíncrono (p.ej.: copias de seguridad periódicas) y la posibiliad de trabajar en primer o segundo plano.

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?

El principal problema es que si todas las hebras de usuario de un proceso vienen a estar representadas por una única hebra de kernel, la llamada bloqueante bloquea al proceso.

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 de usuario y el núcleo planifica procesos y no hebras.

El nivel de paralelismo alcanzado se limita a la posibiilidad de un cambio rápido entre las distintas hebras cuando el proceso se encuentra en ejecución. Por la implementación que se ha usado, no es posible asignar hebras a distintos procesadores o que una hebra se ejecute mientras otra espera E/S, pues las operaciones que requieren llamada al sistema bloqueante bloquean el proceso en su conjunto.

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

Linux no implementa estructuras de datos separadas para procesos y hebras. En su lugar, lo que se hace es que se incluye en el PCB de cada proceso su memoria virtual. Si dos procesos comparten el mismo espacio de memoria virtual, son tratados como dos hebras de un solo proceso, pero no se define una estructura especial para las hebras. Para crear un nuevo proceso se emplea la llamada al sistema `clone()`, que a diferencia de `fork()`, permite que el proceso hijo y el proceso padre compartan el mismo espacio de direcciones, esto es, sean vistos como hebras por el núcleo de Linux. También es posible en Linux implementar hebras a nivel de usuario usando bilbiotecas.

5 En un SO con política de planificación apropiativa, enumere las distintas partes del SO que deben comprobar la posibilidad de desplazar al proceso que se está ejecutando actualmente, 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 mayor penalización a los procesos limitados por E/S frente a los limitados por CPU?¿Por qué?

El algoritmo FCFS (*First Come First Served*) es el que más penaliza a los procesos de mucha E/S y a los procesos cortos, pues los procesos más largos monopolizan los recursos del sistema (tener en cuenta que los procesos cortos ocupan un menor tiempo en CPU). Esto sucede debido a que un proceso limitado por CPU probablemente ocupará el procesador hasta que termine mientras que un proceso limitado por E/S tendrá que volver al final de la cola cada vez que complete una operación E/S.

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

7 Con respecto al núleo de Linux visto en clase:

a.- Describe los pasos que ejecuta el núcleo en la función `do_exit()`.

Los pasos que realiza `do_exit()` son:

1.- Activa el flag `PF_EXITING` de `task_struct`.

2.- Para cada recurso que esté usando el proceso, decrementa el contador que indica el nº de procesos que lo usan. Si el contador llegase a 0, destruye ese recurso (lo libera).

3.- El valor que se pase a `exit()` como argumento se almacena en el campo `exit_code` de `task_struct`. Esta es la información de terminación.

4.- Se manda una señal al proceso padre indicando la finalización (`SIGCHLD`).

5.- Si el proceso tiene hijos, dependiendo de las característas del grupo de procesos, se cambia el padre al proceso `init (PID=1)` o a otro proceso del grupo.

6.- Se establece `task_struct.exit_state = EXIT_ZOMBIE`.

7.- Se llama a `schedule()`.

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

El planificador de Linux utiliza la clase CFS para decidir el orden de ejecución de los procesos, esta clase almacena información sobre los tiempos consumidos por los procesos y los usa para calcular el valror `vruntime`: *virtual runtime*, que se calcula teniendo en cuenta la prioridad, el tiempo de uso de la CPU y el peso del proceso (este peso depende de la prioridad). Cuando `schedule()` es llamado, carga el proceso con menor vruntime. El valor de vruntime se calcula cada cierto tiempo por el propio planificador, cuando llega un nuevo proceso o cuando el proceso actual se bloquea.