Examen 1er parcial de Sistemas Operativos Avanzados (7/5/2014)

1) Para las 3 generaciones de métodos de virtualización, responda a los siguientes aspectos:

	1ra Generación	2da Generación	3ra Generación
Independencia del hardware anfitrión *(¿se requiere que el HW del sistema anfitrión sea uno específico?)*			
Independencia del software alojado *(¿se requiere que el SO alojado sea uno específico?)*			
Inter-plataforma *(¿es posible que el sistema alojado sea a nivel de SO y a nivel de arquitectura hw distinto?)*			
Limitaciones de rendimiento *(Ante la posible pérdida de rendimiento que se observa en el sistema virtualizado ¿a qué se debe?)*			

La visión de las diversas soluciones de virtualización como sucesivas generaciones que van mejorando las propuestas previamente planteadas proviene de la órbita de Xen y, en mi opinión, es sesgada y bastante discutible.

Consúltese la primera cuestión del examen de junio de 2015 para más información sobre las distintas alternativas a la hora de virtualizar un sistema y las ventajas y desventajas de cada una de ellas.

2) *Explique en qué se diferencia un núcleo expulsivo de uno no expulsivo y describa las ventajas y desventajas de cada tipo de núcleo.*

En un núcleo no expulsivo, si un proceso que está ejecutando un evento síncrono en modo sistema (ya sea una llamada al sistema o el tratamiento de una excepción) debe ser expulsado (bien sea porque se ha desbloqueado un proceso más prioritario o porque ha agotado su turno de ejecución), este cambio de contexto involuntario se difiere hasta que el proceso complete el tratamiento del evento síncrono (si es que antes no realiza por su cuenta un cambio de contexto voluntario). En contraposición, en un núcleo expulsivo, el proceso es expulsado dejando sin completar el tratamiento del evento síncrono.

Un núcleo expulsivo proporciona un mejor tiempo de respuesta para los procesos. Así, en un núcleo no expulsivo, si se desbloquea un proceso más prioritario que el que está actualmente en ejecución y éste acaba de comenzar el tratamiento de un evento asíncrono, la activación del proceso más prioritario se retrasará hasta que se complete el tratamiento, lo que empeora significativamente el tiempo de respuesta del sistema.

Por su parte, un núcleo expulsivo presenta mayor fiabilidad y más eficiencia. Un núcleo expulsivo, al permitir la ejecución concurrente de llamadas al sistema, presenta muchos más problemas de sincronización que un núcleo no expulsivo requiriendo implementar y aplicar mecanismos de sincronización para afrontar estos problemas. Estas circunstancias hacen que el núcleo no expulsivo sea intrínsicamente más fiable, al no presentar tantos problemas de sincronización, y más eficiente, ya que no incurre en la sobrecarga asociada al uso de estos mecanismos de sincronización adicionales.

3) *Responda a las siguientes cuestiones sobre la planificación en un multiprocesador:*

(a) *¿Qué esquema usan, y por qué motivo, los SS.OO. actuales: una cola de procesos listos por procesador o una global para todo el sistema?*

El esquema con una única cola global presenta dos deficiencias. Por un lado, dado que todos los procesadores deben acceder a esa cola para todas las operaciones vinculadas con la planificación de procesos y estas operaciones hay que hacerlas en exclusión mutua, se presenta un problema de congestión en el acceso a dicha cola que aumenta según crece la escala del sistema. Por otro lado, puesto que los procesadores se auto-planifican seleccionando procesos de la cola común, no es fácil incorporar soporte a la afinidad, que establece que es mejor que un proceso ejecute sus sucesivas ráfagas en el mismo procesador para posibilitar de esta forma una mejor reutilización de la información almacenada en la jerarquía de cachés del sistema. En este esquema, para dar soporte a la afinidad, a la hora de tomar las decisiones de planificación relativas a un determinado procesador, se le da un cierto *bonus* a la prioridad de un proceso si éste ejecutó su última ráfaga en ese procesador.

Un esquema con una cola por procesador mitiga ambos problemas. Exceptuando en las situaciones de migración de procesos que se realizan para equilibrar la carga en el sistema, cada procesador accede únicamente a su cola de procesos eliminando el problema de la congestión. En cuanto al soporte a la afinidad, dado que cuando se crea un proceso se le asigna un procesador y continúa ejecutando en el mismo a no ser que sea necesaria una operación de migración por desequilibrio de carga en el sistema, se consigue de forma automática.

(b) *¿Cómo influye la jerarquía presente en un multiprocesador (Simultaneous MultiThreading, Chip-level multiprocessing, NUMA,...) en la planificación?*

En un multiprocesador jerárquico los diversos procesadores comparten recursos dependiendo de su ubicación dentro de la jerarquía lo que conlleva la extensión del concepto de afinidad a toda la jerarquía:

- Los procesadores lógicos del mismo *core* comparten la caché de nivel 1 por lo que si un proceso ejecutó su última ráfaga en un procesador y no puede usarlo para su próxima ráfaga, sacará ventaja de la caché de nivel 1 si ejecuta en otro procesador del mismo *core* en vez de en uno asociado a otro *core*.
- Los *cores* del mismo paquete comparten la caché de nivel 2 por lo que si un proceso ejecutó su última ráfaga en un procesador y no puede usarlo para su próxima ráfaga, sacará ventaja de la caché de nivel 2 si ejecuta en otro procesador del mismo paquete en vez de en uno asociado a otro paquete.
- Los procesadores que pertenecen al mismo nodo NUMA comparten la memoria local del nodo por lo que si un proceso ejecutó su última ráfaga en un procesador y no puede usarlo para su próxima ráfaga, sacará ventaja de la memoria local del nodo si ejecuta en otro procesador del mismo nodo en vez de en uno asociado a otro nddo.

4) (a) *Indique qué técnica de gestión de memoria se usa habitualmente para implementar el servicio* `fork` *eficientemente y descríbala explicando qué acciones de gestión de memoria conlleva dicha técnica en la llamada* `fork` *y cuáles cuando el proceso padre y el hijo prosiguen su ejecución después de esa llamada.*

Se utiliza la técnica del *copy-on-write* (COW). Esta técnica se usa para gestionar las regiones de carácter privado del mapa de un proceso, puesto que con una región de este tipo, cada proceso que la utiliza tiene que trabajar con su propia copia de la misma. Gracias a esta técnica, en vez de realizar inicialmente una copia completa de la región para cada proceso que la usa, se establece que se comparte el contenido inicial, realizándose el duplicado de cada página de la región por demanda, cuando el proceso intente modificarla. Dado que es muy frecuente que un proceso no modifique todas las páginas de una región de ese tipo, se obtiene un ahorro significativo al realizar este duplicado por demanda.

La llamada *fork* crea un proceso que es un duplicado del proceso que invoca la llamada. En principio, la implementación de esta llamada implicaría tener que duplicar el mapa del proceso. Sin embargo, gracias al uso del COW, se optimiza considerablemente la implementación de esta operación. Basta con marcar

como COW tanto en el mapa del proceso padre como en el hijo cada región de carácter privado. De esta forma, en vez de tener que duplicar el mapa del proceso, sólo hay que duplicar su tabla de regiones y su tabla de páginas, marcando en esta última como COW las entradas correspondientes a las páginas de una región privada. Para conseguir que el sistema operativo tome control cuando un proceso intenta modificar una página de tipo COW, se elimina el permiso de escritura de la entradas de la tabla de páginas correspondientes a páginas de esta región de manera que se produce una excepción (a veces, denominada fallo de protección) al escribir en la misma.

Cuando después del `fork` el padre o el hijo escriban en una dirección incluida en una página de una región privada, se producirá un fallo de protección, en cuyo tratamiento, el sistema operativo, después de comprobar en la tabla de regiones que dicha región tiene permiso de escritura, busca un marco libre y realiza un duplicado de la página, actualizando la entrada de la tabla de páginas asociada para que haga referencia a ese marco y reasignando el permiso de escritura.

(b) *Teniendo en cuenta que en la mayoría de los casos después del* `fork` *un proceso hijo ejecuta un pequeño fragmento de código y, a continuación, realiza un* `exec` *y dado el uso de la técnica identificada previamente, argumente razonadamente qué proceso debería ejecutar antes (el padre o el hijo) para mejorar el rendimiento del sistema.*

Si ejecutase primero el padre, durante su turno de ejecución causaría varios fallos de protección sobre las páginas marcadas como COW realizándose el duplicado de las mismas. En el momento que le toca ejecuta el hijo rápidamente invoca el servicio `exec` que destruye el mapa de memoria del hijo y crea un nuevo mapa asociado al ejecutable especificado en dicho servicio. Se puede considerar, por tanto, que las operaciones de duplicado realizadas durante el turno de ejecución del padre han sido innecesarias y no se hubieran producido en caso de que el hijo hubiese ejecutado antes.