Examen 1er parcial de Sistemas Operativos Avanzados (12/6/2015)

1) *Explique qué dificultades técnicas presenta la virtualización de la arquitectura x86 y qué dos soluciones existen (full vs. para) para superarlas identificando las ventajas de cada una.*

Dado que en un sistema de virtualización el sistema operativo alojado no puede ejecutar en el nivel de máximo privilegio, ya que si así fuera un sistema operativo alojado podria afectar a otro sistema coexistente, es necesario que todas las instrucciones que se comporten de manera diferente dependiendo del nivel de privilegio en que ejecuten sean emuladas por el hipervisor en vez de ser ejecutadas directamente por el sistema operativo alojado. En algunos procesadores esas instrucciones "conflictivas" causan un *trap* cuando no se ejecutan en modo privilegiado, permitiendo que el hipervisor pueda tomar el control y emularlas adecuadamente (el clásico ciclo de *trap-and-emulate* característico de muchos sistemas de virtualización). Sin embargo, esto no ocurre en la arquitectura x86 donde existen instrucciones, como, por ejemplo, POPF, cuyo modo de operación es distinto dependiendo del nivel en que se ejecutan, pero no causan un *trap*, impidiendo que el hipervisor pueda tomar control para emularlas. Ante este problema, que ha hecho incluso afirmar con frecuencia que esta arquitectura no es virtualizable, se han planteado dos tipos de soluciones: virtualización completa (*full virtualization*), como, por ejemplo, VMware, frente a *paravirtualización*, como, por ejemplo, Xen.

La primera solución plantea incorporar un mecanismo de traducción dinámica que sustituye en tiempo de ejecución esas instrucciones conflictivas por llamadas al hipervisor.

La solución de paravirtualización conlleva modificar el código del sistema operativo alojado (es decir, crear una distribución específica del sistema operativo adaptada a un cierto hipervisor) sustituyendo las instrucciones conflictivas por llamadas al hipervisor.

La virtualización completa proporciona una solución transparente, que no requiere modificar el código del sistema operativo y que se puede aplicar, por tanto, a cualquier sistema operativo, aunque no se disponga de su código fuente. La paravirtualización no ofrece una solución transparente pero presenta menor sobrecarga ya que no requiere ese proceso de traducción dinámica.

2) *Se quiere diseñar un módulo del SO que se encargue de gestionar un dispositivo que accede a un conjunto de sensores y planifica operaciones de lectura sobre los mismos que usan interrupciones. Dadas las siguientes actividades, indicar en qué tipo de contexto de ejecución las situaríais (rutina de evento síncrono, rutina de interrupción, rutina de interrupción software de sistema, rutina de interrupción software de proceso, proceso de núcleo, actividad en modo usuario), explicando brevemente por qué:*

1. *Actividad realizada sobre el dispositivo, para que planifique una operación de lectura bloqueante sobre un determinado sensor*
2. *Recogida del valor medido por el sensor*
3. *Interpretación del valor medido por el sensor (que implica una labor de cierta complejidad)*
4. *Terminación involuntaria de un proceso que está en medio de la recogida de un valor medido por el sensor*
5. *Implementación de una operación de lectura asíncrona sobre un determinado sensor*

1. La operación de bloqueo se debe llevar a cabo en el contexto de un evento síncrono. Por tanto, estará incluida en la llamada al sistema de lectura del dispositivo:

```
cola_t cola_disp;
read(void *dir) {
   programa_dispositivo();
   bloquear(&cola_disp);
   ..................
}
```

2. Al tratarse de un dispositivo gestionado mediante interrupciones, la recogida del valor medido debe realizarse en el contexto de la rutina de interrupción del mismo:

```
buffer_t buffer_disp;
cola_t cola_disp;
read(void *dir) {
   programa_dispositivo();
   bloquear(&cola_disp);
   *dir = extraer(&buffer_disp);
}
int_disp(){
   dato d = leer_disp();
   insertar(d, &buffer_disp);
   desbloquear(&cola_disp);
}
```

3. Dado que no se debe realizar un procesamiento largo dentro de una rutina de interrupción puesto que haría que el sistema tuviera peor tiempo de respuesta (todas las interrupciones de menor prioridad deberían esperar hasta que se realizase ese procesamiento complejo) e incluso que se perdiera alguna interrupción, hay que delegar esta operación a otro contexto, presentándose dos opciones.

   (a) Realizar el procesamiento en el contexto de la llamada al sistema:

```
buffer_t buffer_disp;
cola_t cola_disp;
read(void *dir) {
   programa_dispositivo();
   bloquear(&cola_disp);
   *dir = procesamiento(extraer(&buffer_disp));
}
int_disp(){
   dato d = leer_disp();
   insertar(d, &buffer_disp);
   desbloquear(&cola_disp);
}
```

   (b) Llevarlo a cabo en el contexto de una interrupción software de sistema que no retendrá a las interrupciones hardware ya que ejecuta con un nivel de prioridad menor:

```
buffer_t buffer_disp, buffer_disp_procesados;
cola_t cola_disp;
read(void *dir) {
   programa_dispositivo();
   bloquear(&cola_disp);
   *dir = extraer(&buffer_disp_procesados);
}
int_disp(){
   dato d = leer_disp();
   insertar(d, &buffer_disp);
   activar_int_SW_disp();
}
```

```
int_SW_disp(){
   dato_procesado dp = procesamiento(extraer(&buffer_disp));
   insertar(dp, &buffer_disp_procesados);
   desbloquear(&cola_disp)
}
```

4. Para evitar los problemas de sincronización asociados a la terminación involuntaria de un proceso, se puede usar una interrupción sofware dirigida a ese proceso de manera que el mismo pueda terminar su ejecución de una manera ordenada.
5. Una operación de lectura asíncrona devuelve el control inmediatamente presentando el problema de cómo copiar los datos del buffer del dispositivo al especificado por el programa, dado que esta operación de copia la debe llevar a cabo el propio proceso siendo impredecible qué está haciendo éste cuando se dispone del dato. Una posible solución es enviar al proceso una interrupción software de manera que la copia se realice en el contexto de la misma. A continuación, se muestra esta solución incluida en la primera opción explicada en el tercer apartado:

```
buffer_t buffer_disp;
void *d;
proc_t proc;
read(void *dir) {
   d = dir;
   proc = proc_actual;
   programa_dispositivo();
}
int_disp(){
   dato d = leer_disp();
   insertar(d, &buffer_disp);
   activar_int_SW_proceso(proc);
}
int_SW_proceso() {
   *d = procesamiento(extraer(&buffer_disp));
}
```

3) (a) *Describa cómo se implementa el concepto de afinidad a un procesador en un sistema que usa una cola única de procesos listos y cómo en un sistema con una cola por procesador explicando qué solución es más efectiva en lo que se refiere a respetar dicha afinidad.* ***(b)*** *A continuación, extienda la cuestión previa a un sistema multiprocesador jerárquico donde la afinidad afecta a toda la jerarquía.*

(a) En un esquema con cola única de procesos listos, se tiene en cuenta en qué procesador ejecutó un proceso su última ráfaga, de manera que 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.

En un esquema con cola múltiple, cuando se crea un proceso, se le asigna un procesador y seguirá ejecutando en el mismo a no ser que sea necesaria una operación de migración por equilibrado de carga. Por tanto, a diferencia del esquema anterior donde la afinidad se incorpora de una manera tangencial, en este esquema se le da soporte automáticamente puesto que, si no hay necesidad de equilibrado, el proceso seguirá ejecutando en el mismo procesador.

(b) En cuanto al esquema con cola única de procesos listos, se extiende el mecanismo de *bonus* a toda la jerarquía de manera que no sólo se le da un *bonus* extra si un proceso ejecutó su última ráfaga en el procesador objeto de la planificación, sino que también se le asigna un *bonus* si el proceso ejecutó su

última ráfaga en otro procesador vinculado jerárquicamente con el que se está planificando. El valor de este *bonus* será mayor cuanto más cercanos en la jerarquía estén los procesadores involucrados (el objeto de la planificación y aquél donde ejecutó su última ráfaga el proceso).

Con respecto al esquema de cola múltiple, la afinidad por cercanía en la jerarquia influye a la hora de realizar una migración. En caso de desequilibrio de carga en el sistema, el procesador menos cargado intentará primero migrar procesos que estén ejecutando en un procesador lo más cercano posible en la jerarquía.

4) *Describa la técnica COW de gestión de memoria explicando* ***(a)*** *para qué tipo de regiones se usa,* ***(b)*** *en la implementación de qué llamada al sistema de gestión de procesos se utiliza,* ***(c)*** *sobre qué evento del procesador se articula y cómo se fuerza dicho evento y* ***(d)*** *el seudo-código simplificado de la rutina de tratamiento de ese evento.*

(a) Se usa para las regiones de carácter privado (por ejemplo, la región de datos con valor inicial de un ejecutable o de una bibioteca dinámica), 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.

(b) Se utiliza en la implementación de la llamada *fork* que crea un proceso que es un duplicado del proceso que invoca la llamada. Cada región de carácter privado en el mapa de proceso se marca como COW tanto en el mapa del proceso padre como en el hijo, optimizando considerablemente la implementación de esta operación ya que, 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.

(c) 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.

(d) Tratamiento simplificado del fallo de protección causado por una operación de escritura sobre la dirección D:

- Comprobar en la tabla de regiones si D pertenece a una región con permiso de escritura. En caso negativo, se trata de un error del programa y se le envía al proceso una señal de acceso a memoria inválido.
- Si el número de referencias asociado a la página correspondiente a D es mayor que 1 (más de un proceso comparte esa página), se busca un marco libre y se realiza un duplicado de la página. Se actualiza la entrada de la tabla de páginas asociada para que haga referencia a ese marco asignando permiso de escritura y decrementando el contador de referencias.
- En caso de que el número de referencias sea 1 (es el único proceso que hace referencia a esa página), basta con devolver el permiso de escritura a esa página.