Examen 2º parcial de Sistemas Operativos Avanzados (12/6/2015)

1) *Dado un sistema monoprocesador con núcleo expulsivo, indicar qué posibles conflictos de sincronización podría tener una rutina de interrupción SW de sistema con las siguientes rutinas concurrentes (llamada al sistema, excepción, interrupción SW de sistema, interrupción hardware de mínima prioridad e interrupción hardware de máxima prioridad) y cómo se resolverían.*

A continuación, se analiza cada caso planteado:

- Llamada al sistema: la ejecución de una llamada puede verse detenida por una interrupción que, a su vez, puede activar una interrupción software de sistema. Cuando se complete el tratamiento de la interrupción convencional, se activará el de la interrupción software de sistema, al ser más prioritaria que la llamada. Para resolver el posible conflicto, se inhibirán las interrupciones software de sistema en el fragmento de código de la llamada de manera que se impida la activación de la interrupción software de sistema durante el mismo.
- Excepción: al tratarse de un evento síncrono, plantearía la misma problemática que el caso previo.
- En el resto de los casos, no hay posibilidad de conflicto puesto que los eventos planteados tienen una prioridad mayor o igual que la interrupción software de sistema, no pudiéndose, por tanto, verse interrumpidos por la misma.

2) *Describa en qué consisten las técnicas de* ***predicción*** *de interbloqueos explicando el concepto de estado seguro, nombrando un esquema basado en las mismas e identificando todos sus inconvenientes.*

Antes de que se produzca un interbloqueo, existe un punto donde el sistema transita de un estado seguro a uno inseguro: si se evita que el sistema entre en un sistema inseguro, no podrá producirse nunca un interbloqueo (aunque, por otro lado, un sistema en estado inseguro podría no conducir a un interbloqueo). Para poder determinarse cuando el estado del sistema es seguro, se deben conocer cuáles serán las necesidades futuras máximas de cada proceso: El estado será seguro si, aunque en ese momento todos los procesos solicitaran sus necesidades máximas, no habría interbloqueo en el sistema.

Las técnicas de predicción de interbloqueos controlan la asignación de recursos asegurando que el sistema siempre está en un estado seguro: si la petición de un recurso hace que el sistema transite a un estado inseguro, no se satisface dicha petición bloqueando al proceso que la cursó, incluso aunque el recurso esté libre y, por tanto, no haya interbloqueo. El algoritmo del banquero es un ejemplo de este tipo de técnicas.

Estas técnicas son difícilmente aplicables a un sistema real. Por un lado, presentan la sobrecarga de ejecutar el algoritmo de comprobación cada vez que se solicita un recurso incluso aunque el recurso esté libre. Asimismo, causan infrautilización de recursos puesto que éstos pueden no asignarse aunque estén libres y sean requeridos por un proceso. A todo esto hay que añadir la necesidad de conocer las necesidades futuras de los recursos, lo cual no es factible en un sistema de propósito general. Además, estas necesidades futuras deben expresar el peor caso posible y, por tanto, pueden provocar falsos positivos.

3) *Para las dos alternativas vistas en el tema para la gestión de archivos de tamaño pequeño (la implementada en XFS y la proporcionada por Reiser), responda a las siguientes cuestiones:*

1. *Proponga una estructura básica de un i-nodo "normal" de un sistema UNIX.*
2. *Modifique el diseño e incluya la información necesaria (campos adicionales en las estructuras de metadatos) para implementar la gestión que hace XFS de ficheros de tamaño pequeño. Indique también qué condición debe cumplirse para que se esté usando esta estrategia de almacenamiento.*
3. *Haga lo mismo (diseño y condiciones) para el caso de ReiserFS.*

1. El típico *i-nodo* de UNIX consta de una parte con la metainformación del fichero (dueño, tipo de fichero, permisos de acceso, tamaño, fechas de acceso y modificación, número de enlaces, etc.) y otra con una colección de punteros directos e indirectos a los bloques del fichero.
2. XFS usa el espacio ocupado por los punteros para almacenar ficheros pequeños. Para poder hacer esto, es necesario, evidentemente, que el tamaño del fichero sea menor o igual que el espacio que ocupan los punteros. Nótese que si almacena el fichero dentro del propio *i-nodo* no se necesitan usar estos punteros y puede usarse el espacio ocupado por los mismos para guardar el contenido siempre que quepa en dicho espacio. En principio, no sería estrictamente necesario añadir nueva información en el *i-nodo* para saber si los datos están dentro del mismo o no puesto que esto se puede deducir del tamaño del fichero.
3. ReiserFS usa la técnica denominada *tail-packing* no sólo para optimizar el almacenamiento requerido por ficheros pequeños, sino, en general, para almacenar eficientemente los últimos bytes de los ficheros (la cola) cuando ocupan sólo una parte muy pequeña de un bloque, reduciendo, de esta forma, la fragmentación interna. Con esta técnica, se puede usar un bloque para almacenar fragmentos de distintos ficheros. Para implementar esta estrategia, es necesario guardar información en el *i-nodo* que, en caso de que la parte final de un fichero esté almacenada en un bloque compartido con otros ficheros, especifique en qué parte del mismo está almacenada.

4) *Responda a las siguientes cuestiones vinculadas con aspectos de seguridad:*

El código de la derecha:

1. *¿Qué efecto tiene?*
2. *¿Qué tipología de ataque es?*
3. *¿De qué forma se pueden evitar un ataque de este tipo?*

```
#include <stdio.h>
int main(int argc, char **argv){
    while(1) {
       malloc(1024);
       fork();
    }
}
```

1) Ese código, que ejecuta indefinidamente, crea un número exponencial de procesos (después de n iteraciones del bucle habrá 2^n procesos en el sistema), tal que cada uno reserva memoria en el *heap* en cada iteración. Si no se toma ninguna medida, ese código colapsará el sistema, tanto por el enorme número de procesos creados como por la memoria requerida por cada uno.
2) Se trata de un ataque de tipo conejo o bacteria que se caracteriza porque, aunque no realiza ningún tipo de acción directa destructiva, busca colapsar el sistema acaparando recursos, lo que causa un ataque de tipo denegación de servicio.
3) Los mecanismos que ofrece un sistema operativo para limitar el uso de recursos por parte de un usuario o un proceso pueden evitar este tipo de ataques. En el ejemplo planteado, en UNIX se podría usar el servicio setrlimit para limitar el número de procesos que puede crear un usuario (RLIMIT_NPROC), así como el gasto de memoria de cada proceso (RLIMIT_DATA).