DISEÑO DE SISTEMAS OPERATIVOS 4º. Febrero de 2005

Ejercicio 1

Responda razonadamente a las siguientes preguntas sobre procesos y gestión de memoria.

a) La inversión de prioridades se produce cuando estando listo para ejecutar un proceso de una determinada prioridad, se está ejecutando otro proceso de menor prioridad. Se trata de un problema importante en los sistemas de tiempo real (que se lo pregunten a los diseñadores del *PathFinder*), que hay que intentar mantener acotado. Se plantea analizar este problema usando el siguiente ejemplo de tres procesos de prioridad creciente:

- *Pbaja*: está ejecutando una llamada al sistema que accede y actualiza durante un periodo de tiempo relativamente largo una estructura de datos del sistema operativo. Una vez terminada la llamada, el proceso ejecutará un cierto tiempo en modo usuario y terminará.
- *Pmedia*: está bloqueado. Mientras *Pbaja* está ejecutando la llamada, llegará una interrupción *X*, que desbloqueará a *Pmedia*. Cuando ejecute *Pmedia*, realizará una llamada al sistema que no tiene ninguna relación con la que realiza *Pbaja*, y, luego ejecutará un cierto tiempo en modo usuario y terminará.
- *Palta*: está bloqueado. Casi al final de la rutina de tratamiento de la interrupción *X*, se produce una interrupción *Y* de mayor prioridad que desbloqueará a *Palta*. Cuando ejecute *Palta*, realizará una llamada al sistema potencialmente conflictiva con la que realiza *Pbaja*, y, a continuación, ejecutará un cierto tiempo en modo usuario y terminará.

Suponiendo que se trata de un sistema con un núcleo no expulsivo que usa una planificación por prioridades de tipo expulsiva, responda a las siguientes preguntas:

a1) [Puntuación: 0,5/10] ¿Cómo se resuelve en este tipo de núcleos un posible conflicto entre llamadas como el planteado en el ejemplo?

a2) [Puntuación: 1,5/10] Muestre cuál es la traza de ejecución de los procesos planteados desde el instante que se ha descrito hasta que terminan, especificando **todas** las activaciones del sistema operativo que se producen y resaltando los cambios de contexto y el carácter de los mismos (voluntarios o involuntarios).

a3) [Puntuación: 1/10] Analice si se produce inversión de prioridades y, en caso afirmativo, especifique cuál es el alcance de la misma y plantee una posible solución para mitigarla, en caso que sea factible.

a4) a5) a6) [Puntuación: 3/10] Repita los tres apartados anteriores, para un sistema que usa el mismo algoritmo de planificación pero con un núcleo expulsivo.

b) La mayoría de los sistemas operativos mantienen una estadística de cuántos fallos de página se producen en el sistema que no implican una operación de lectura de disco (en Linux se denominan *minor faults* y en Windows *soft faults*). Suponiendo que se trata de un sistema de memoria virtual basado en paginación por demanda sin *buffering* de páginas, para cada una de las situaciones que se plantean a continuación, analice si se produciría un fallo de página (ya sea convencional o el asociado al *copy-on-write*) y, en caso afirmativo, si implicaría una lectura de disco o no. Si la implica, especifique si es de un fichero o de *swap*; si no la implica, explique si se requiere un marco libre para servir el fallo y qué información se almacena en el mismo.

b1) [Puntuación: 0,5/10] Después de reservar memoria dinámica, el proceso escribe en la misma.

b2) [Puntuación: 0,5/10] Después de realizar una llamada a procedimiento que hace que la región de pila se expanda, el proceso modifica una variable local.

b3) [Puntuación: 0,5/10] El proceso lee una variable global sin valor inicial que ha modificado previamente, pero cuya página no está actualmente presente.

b4) [Puntuación: 0,5/10] Un proceso escribe en una variable global con valor inicial a la que ha accedido previamente sin modificarla, pero cuya página no está actualmente presente.

b5) [Puntuación: 0,5/10] Inmediatamente después de que un proceso modifique una variable global con valor inicial y, a continuación, cree un hijo (FORK), el proceso hijo lee esa misma variable.

b6) [Puntuación: 0,5/10] Inmediatamente después de que un proceso lea una variable global con valor inicial y, a continuación, cree un hijo (FORK), el proceso hijo escribe en esa misma variable.

b7) [Puntuación: 0,5/10] Inmediatamente después de que un proceso modifique una variable global con valor inicial y, a continuación, cree un *thread*, el nuevo *thread* escribe en esa misma variable.

b8) [Puntuación: 0,5/10] Analice cómo afectaría a esta estadística el uso de la técnica del *buffering* de páginas. ¿Habría nuevas situaciones de fallos sin lectura de disco? ¿Desaparecerían algunas de las situaciones de fallos sin lectura de disco que se producen en un sistema sin *buffering*?

Solución

a1) En un núcleo de carácter no expulsivo no se permite que haya llamadas al sistema concurrentes. En el caso de que se requiera un cambio de contexto involuntario durante la ejecución de una llamada al sistema (debido a que se le haya terminado al proceso actual su turno de ejecución o a que se haya desbloqueado un proceso con mayor prioridad), el cambio de contexto se difiere hasta que termine la llamada en curso o hasta que se bloquee el proceso actual dentro de la misma (en ese caso, se trataría de un cambio de contexto voluntario). Este aplazamiento del cambio de contexto se implementa utilizando una interrupción software cuya prioridad no le permite interrumpir la llamada en curso. Por tanto, no pueden darse conflictos entre llamadas al sistema en este tipo de sistemas operativos.

a2) A continuación se muestra la traza de ejecución planteada en el enunciado:

- Durante la ejecución de una llamada al sistema por parte del proceso *Pbaja* se produce la interrupción *X*. Dado que mientras se ejecuta una llamada al sistema las interrupciones están habilitadas, se activa la rutina de tratamiento de dicha interrupción, que ejecutará en el contexto del proceso *Pbaja*.
- En la rutina de tratamiento de la interrupción *X* se desbloquea al proceso *Pmedia* y, al detectar que tiene mayor prioridad que el proceso que está ejecutando actualmente, se activa una interrupción software para diferir el cambio de contexto, anotando este hecho en alguna variable del sistema operativo (como, por ejemplo, *need_resched* en Linux). Cuando está terminando la ejecución de la rutina de interrupción, se produce una segunda interrupción *Y* de mayor prioridad, activándose su rutina de tratamiento, que seguirá ejecutando en el contexto del proceso *Pbaja*.
- En la rutina de tratamiento de la interrupción *Y* se desbloquea al proceso *Palta*. Nuevamente, el proceso desbloqueado tiene mayor prioridad que el proceso que está ejecutando actualmente. Se podría activar una interrupción software para diferir el cambio de contexto (como el resto de las interrupciones, este tipo de evento no se encola quedando sólo constancia de una aparición del mismo) o, al comprobar que ya existe una interrupción software activa, no realizar ninguna operación. La rutina de la interrupción *Y* terminará retornando a la de la interrupción *X*, que también concluirá reanudándose la llamada al sistema interrumpida. Nótese que, aunque la interrupción Y se hubiera producido antes del desbloqueo de *Pmedia* en el ámbito de la rutina de tratamiento de la interrupción *X*, el resultado global hubiera sido el mismo.
- Continúa la ejecución de la llamada al sistema terminando su cometido. En el momento que se va a retornar a modo usuario, se trata la interrupción software pendiente. La rutina de tratamiento de la interrupción software comprueba que hay un cambio de contexto involuntario pendiente de realizar por lo que invoca al planificador, que seleccionará al proceso *Palta*, y realiza el cambio de contexto al mismo.
- El proceso *Palta* realiza una llamada al sistema y cuando concluya la misma seguirá ejecutando en modo usuario sin verse perturbado por los otros dos procesos al tener estos menor prioridad. Finalmente, el proceso realizará una llamada al sistema para terminar su ejecución, que producirá un cambio de contexto voluntario al proceso seleccionado por el planificador, que será *Pmedia*.
- El proceso *Pmedia* realiza una llamada al sistema. Acto seguido, continúa su ejecución en modo usuario y, finalmente, el proceso realiza una llamada al sistema para terminar su ejecución, que producirá un cambio de contexto voluntario al proceso seleccionado por el planificador, que será *Pbaja*.
- El proceso *Pbaja* prosigue su ejecución en el punto donde se quedó, es decir, justo después del cambio de contexto dentro de la rutina de tratamiento de la interrupción software. Retorna a modo usuario y sigue ejecutando hasta que invoca una llamada al sistema que indica el final de su ejecución, realizándose un cambio de contexto voluntario al proceso nulo.

a3) En la traza desarrollada en el apartado anterior se detecta que tanto *Palta* como *Pmedia* sufren inversión de prioridades con respecto al proceso *Pbaja*. Desde que se desbloquea *Palta* hasta que empieza a ejecutar tiene que esperar no sólo la terminación de las rutinas de interrupción, lo cual es inevitable para asegurar el funcionamiento correcto del sistema, sino también la conclusión de la llamada al sistema de *Pbaja*. El alcance de esta inversión corresponde con el tiempo que dura la llamada al sistema. Por su parte, *Pmedia* también sufre esta inversión con respecto a *Pbaja*.

Hay que resaltar que esta inversión de prioridades es consustancial al modo de operación del núcleo no expulsivo y que, por tanto, la única forma de eliminarla es modificar el comportamiento del núcleo para que tenga un carácter expulsivo.

a4) En los núcleos expulsivos se pueden ejecutar de forma concurrente llamadas al sistema puesto que en este tipo de sistemas, cuando se detecta la necesidad de realizar un cambio de contexto involuntario, sólo es necesario aplazarlo hasta que terminen las rutinas de interrupción que puedan estar activas en ese momento (puede haber varias anidadas), pero no hay que esperar a que concluya la llamada al sistema en curso, en caso de que haya alguna. Este esquema se suele implementar basándose también en una interrupción software, pero tal que su prioridad sea capaz de interrumpir la ejecución de las llamadas al sistema.

Como resultado de este esquema, la ejecución de las llamadas al sistema de varios procesos se puede intercalar de forma impredecible. Con ello, se consigue un sistema con un mejor tiempo de respuesta, pero con más dificultades a la hora de lograr una correcta sincronización.

En este tipo de núcleos, por tanto, sí puede haber conflictos entre llamadas al sistema, existiendo, básicamente, dos técnicas para resoverlos:

- Elevar el nivel de interrupción durante la sección crítica para impedir el tratamiento de las interrupciones software. Nótese que sólo habría que impedir ese tipo de interrupciones de baja prioridad, estando habilitadas las restantes.
- Implementar un mecanismo de sincronización interno del tipo semáforo o similar (por ejemplo, *mutex*) que permita proteger adecuadamente la sección crítica afectada. Téngase en cuenta que para proteger a su vez las operaciones del semáforo habría que usar la técnica anterior (es decir, elevar el nivel de interrupción para impedir la activación de interrupciones software) pero sólo durante dichas operaciones, no a lo largo de toda la sección crítica.

Dadas las características del problema de sincronización planteado en el enunciado, donde el tamaño de la sección crítica es apreciablemente largo, la segunda solución sería la apropiada para evitar que quede afecatdo negativamente el tiempo de respuesta en el sistema debido al intervalo de tiempo relativamente largo en el cual estarían inhabilitadas las interrupciones software.

a5) A continuación se muestra la traza de ejecución planteada en el enunciado teniendo en cuenta la técnica de sincronización seleccionada en el apartado previo:

- Como parte de la ejecución de la llamada al sistema del proceso *Pbaja*, se cierra un semáforo que protege la estructura de datos conflictiva planteada en el enunciado. Durante la ejecución de esta llamada al sistema se produce la interrupción *X*, activando la rutina de tratamiento de dicha interrupción.
- En la rutina de tratamiento de la interrupción *X* se desbloquea al proceso *Pmedia* y, al detectar que tiene mayor prioridad que el proceso que está ejecutando actualmente, se activa una interrupción software para diferir el cambio de contexto. Cuando está terminando la ejecución de la rutina de interrupción, se produce una segunda interrupción *Y* de mayor prioridad, activándose su rutina de tratamiento, que seguirá ejecutando en el contexto del proceso *Pbaja*.
- En la rutina de tratamiento de la interrupción *Y* se desbloquea al proceso *Palta*. Nuevamente, el proceso desbloqueado tiene mayor prioridad que el proceso que está ejecutando actualmente, aunque no sería necesario realizar ninguna acción puesto que ya hay una interrupción software activa. La rutina de la interrupción *Y* terminará, reanudándose la rutina de la interrupción *X*, que a su vez también concluirá retornando a la llamada al sistema interrumpida. Justo en ese momento, el nivel de interrupción del procesador posibilita el tratamiento de la interrupción software pendiente.
- La rutina de tratamiento de la interrupción software comprueba que hay un cambio de contexto involuntario pendiente de realizar, invocando al planificador, que seleccionará al proceso *Palta*, y realiza el cambio de contexto al mismo.
- El proceso *Palta* realiza una llamada al sistema. Durante el curso de la misma, se intenta cerrar (o sea, bajar) el semáforo que controla el acceso a la estructura de datos, produciéndose el bloqueo del proceso al estar dicho semáforo en posesión de *Pbaja*. Por tanto, hay un cambio de contexto voluntario al proceso seleccionado por el planificador, que será *Pmedia*.
- El proceso *Pmedia* realiza una llamada al sistema. Acto seguido, continúa su ejecución en modo usuario y, finalmente, el proceso realiza una llamada al sistema para terminar su ejecución, que producirá un cambio de contexto al proceso seleccionado por el planificador, que será *Pbaja*.
- El proceso *Pbaja* prosigue su ejecución en el punto donde se quedó, es decir, justo después del cambio de contexto dentro de la rutina de tratamiento de la interrupción software, de donde retorna y reanuda la ejecución de la llamada al sistema interrumpida. Al proseguir la ejecución de la llamada, se alcanzará el punto donde se termine el acceso a la estructura de datos liberando el semáforo asociado a la misma, lo que causará el desbloqueo de *Palta*. Al detectarse que tiene mayor prioridad que el proceso que está ejecutando actualmente, se activa una interrupción software para realizar un cambio de contexto involuntario. Dado que el nivel de interrupción del procesador lo permite, se procesará inmediatamente la interrupción software, en cuya rutina de tratamiento se realizará el cambio de contexto involuntario a *Palta.*
- El proceso *Palta* completará la llamada al sistema ejecutando, a continuación, en modo usuario hasta que, finalmente, realizará una llamada al sistema para terminar su ejecución, que producirá un cambio de contexto voluntario al proceso seleccionado por el planificador, que será *Pbaja*.
- El proceso *Pbaja* prosigue su ejecución en el punto donde se quedó, es decir, justo después del cambio de contexto dentro de la rutina de tratamiento de la interrupción software, retornando de la misma, y reanudando y terminando la ejecución de la llamada al sistema interrumpida. A continuación, retornará a modo usuario y seguirá ejecutando hasta que termine, realizándose un cambio de contexto voluntario al proceso nulo.

a6) Con el uso de un núcleo expulsivo desaparecen las inversiones de prioridades vinculadas con el no poder cambiar de proceso durante la ejecución de una llamada al sistema. Sin embargo, sí se produce otro tipo de inversión de prioridades en el sentido de que un proceso de alta prioridad no puede continuar (*Palta* bloqueado en el semáforo) hasta que otro de menor prioridad se lo permita (*Pbaja* libere el semáforo). En este caso, el alcance de esta inversión es, en principio, ilimitado, puesto que la existencia de procesos de una prioridad intermedia (*Pmedia*) puede impedir indefinidamente que prosiga la ejecución del proceso de baja prioridad y, por tanto, indirectamente, la del proceso de alta prioridad. La solución habitual a este problema en los entornos de tiempo real (aunque también está implementada en sistemas operativos de propósito general como Solaris) es *la herencia de prioridades*: aumentar la prioridad del proceso que está en posesión del semáforo de manera que “herede” la de los procesos que están bloqueados esperando la liberación del mismo.

Nótese que este tipo de situaciones también se da en un núcleo de tipo no expulsivo, aunque no ocurran en el ejercicio planteado. Sin embargo, en los núcleos expulsivos sólo se producen en secciones críticas vinculadas con situaciones de sincronización entre aplicaciones (por ejemplo, en el uso de semáforos de usuario) y no dentro del código de las llamadas del sistema operativo, que se ejecutan atómicamente.

b1) En un sistema con memoria virtual, la operación de reserva de memoria dinámica (región de *heap*) no conlleva la asignación de memoria física hasta el momento en el que se va accediendo a las distintas páginas de la zona reservada. Por tanto, cuando se produce la escritura en la zona reservada, se producirá un fallo de página que no conllevará una lectura del disco, puesto que esa zona no tiene ningún contenido previo prefijado (se trata de una región sin soporte). Se requerirá un marco libre en el que, dependiendo de la política de seguridad del sistema operativo, se borraría (se escribirían ceros) su contenido previo o se dejaría tal cual está, lo que implicaría una solución más eficiente pero con un menor grado de seguridad.
b2) La expansión de la pila es similar a una reserva de memoria dinámica. No se asigna espacio físico hasta que se accede a la misma provocándose un fallo de página. Las características de este fallo de página son, por tanto, las mismas que el considerado en el apartado anterior: sin lectura de disco, requiere un marco libre, que podría rellenarse con ceros por motivos de seguridad.
b3) Al no estar residente la página que se accede, se producirá, evidentemente, un fallo de página. Dado que se trata de una página correspondiente a la región de datos sin valor inicial, que no tiene soporte en el ejecutable, pero que ya ha sido modificada previamente, actualmente estará residente en *swap* y, por tanto, se tratará de un fallo de página que requiere una lectura de disco, concretamente, del dispositivo de *swap*.
b4) Dado que este caso corresponde con un acceso a una página no residente perteneciente a la región de datos con valor inicial, también requerirá lectura de disco, pero ya que no se ha modificado la página, se servirá directamente del fichero ejecutable.
b5) Como resultado del FORK, el proceso padre y el hijo comparten, aunque sólo en modo de lectura mediante el uso de la técnica del *copy-on-write*, la página que contiene la variable afectada. Dado que la página estará residente, ya que acaba de accederse, y el proceso hijo no modifica la variable, no se produce un fallo de página en esta situación.
b6) A diferencia del punto anterior, en este caso el proceso hijo escribe en la variable, lo que causa que se produzca un fallo debido al *copy-on-write*, que requerirá usar un marco libre y duplicar en el mismo el contenido de la página creando una copia específica para el hijo.
b7) En este caso no se produce un fallo de página ya que los *threads* de un proceso comparten el mismo mapa de memoria. Por tanto, si un *thread* de un proceso ha accedido a una variable causando que la página que la contiene pase a estar residente, cualquier otro *thread* de ese proceso no causará un fallo al accederla.
b8) El uso de la técnica del *buffering* de páginas provocaría que algunos fallos de página que, en principio, requerirían un acceso a disco, no lo necesiten al encontrarse la página solicitada residente en memoria, aunque no esté asociada a ningún proceso.
Por otra parte, no desaparecería ninguna de las situaciones de fallos de página que se producen en un sistema que no usa esta técnica: si un fallo de página no requería leer del disco utilizando un esquema sin *buffering* de páginas no se verá afectado por el uso de esta técnica.