Examen de septiembre de 2006

Ejercicio 1

Sobre la pereza y los sistemas operativos (si hay que ir se va, pero ir para nada es tontería). Se pretende estudiar diversas estrategias "perezosas" usadas en los sistemas operativos, analizando en qué situaciones pueden ser beneficiosas con respecto a las estrategias "no perezosas" y en cuáles no lo son o incluso pueden ser perjudiciales.

a) **[Puntuación: 1/10]** Una técnica perezosa en la gestión de memoria es la paginación por demanda. Suponiendo que hay suficiente memoria física, explique en qué situaciones es beneficiosa, en cuanto a una ejecución más eficiente del proceso afectado, comparándola con el uso de la solución no perezosa, y en cuáles perjudicial y por qué motivo, en caso de que pueda serlo.

b) **[Puntuación: 1,5/10]** ¿Qué otra técnica perezosa se usa en la gestión de memoria? Para esa técnica, responda a la misma pregunta que en el apartado anterior.

c) **[Puntuación: 1,5/10]** Durante el arranque, Linux almacena en un determinado marco una página de sólo lectura llena de ceros, que se mantendrá así todo el tiempo, y que se usa para implementar una técnica perezosa en la gestión de páginas de una región anónima (es decir, sin fichero asociado). Explique cómo sería esta técnica e identifique en qué situaciones puede ser beneficiosa y en cuáles perjudicial, si es que puede serlo.

d) Dado que muchos programas no usan números en coma flotante y la mayoría de los sistemas operativos tampoco, se puede plantear una técnica perezosa a la hora de guardar y restaurar el contenido de los registros de este tipo del procesador. A continuación se plantea una serie de preguntas sobre esta idea suponiendo que se usa un sistema operativo cuyo código no utiliza aritmética en coma flotante.

- **d1) [Puntuación: 0,5/10]** Explique razonadamente si sería necesario salvaguardar los registros de coma flotante al comenzar una llamada al sistema y restaurarlos al terminar ésta.
- **d2) [Puntuación: 1,5/10]** Explique razonadamente si puede ser necesario salvaguardar y restaurar los registros de coma flotante ante el cambio entre dos *threads* de distinto proceso. ¿Y si se trata de dos *threads* del mismo proceso? ¿Y en caso de que sean dos *threads*/procesos de núcleo?
- **d3) [Puntuación: 1/10]** ¿Qué valor inicial deben tener los registros de coma flotante para un nuevo proceso creado con *fork*? ¿Y cuál para un nuevo *thread*?
- **d4) [Puntuación: 3/10]** Para ayudar a gestionar esta política perezosa, el procesador Pentium incluye un mecanismo con las siguientes características: hay un campo denominado TS en un registro de control del procesador (*cr0*), accesible sólo en modo privilegiado, de manera que si TS vale 1 y se usa una instrucción que accede a los registros de coma flotante se produce una excepción denominada "Dispositivo no disponible". En caso de que TS valga 0, no se produce ninguna excepción al acceder a dichos registros.
  Plantee un mecanismo perezoso basado en este esquema que intente minimizar la necesidad de salvaguardar y restaurar los registros de coma flotante, especificando qué campos se añadirían al BCP del proceso, qué rutinas del sistema operativo habría que modificar y qué código habría que incluir en las mismas. Asimismo, analice cuándo el uso de este mecanismo puede ser beneficioso y cuándo perjudicial, en caso de que pueda serlo.

Solución

a) En un sistema sin memoria virtual, a la hora de activar la ejecución de un programa, hay que traer a memoria toda la imagen del proceso. Con la técnica de la paginación por demanda, esa operación de carga en memoria se puede diferir hasta que el proceso vaya accediendo a las distintas páginas de su mapa. Si nos fijamos solamente en el aspecto del rendimiento en la ejecución del programa, como plantea el enunciado, la técnica de memoria virtual es beneficiosa puesto que sólo se traen a memoria las páginas requeridas. Este beneficio es particularmente significativo cuando se accede a un número relativamente pequeño de páginas del mapa. El peor caso aparecerá si se acceden a todas las páginas del mapa. En ese caso, ambas soluciones requerirían cargar en memoria todas las páginas, pero en la paginación por demanda habría una sobrecarga adicional por las activaciones del sistema operativo causadas por los sucesivos fallos de página.

b) En el sistema de gestión de memoria se utilizan otras técnicas de carácter perezoso. En esta solución se ha elegido la técnica del *copy-on-write* (COW), que permite optimizar el duplicado de una región. Cuando un proceso requiere un duplicado de una región privada asociada normalmente a otro proceso (ya sea las regiones privadas del proceso padre en una llamada *fork* o las regiones privadas de un ejecutable, o de una biblioteca dinámica, cuando dos o más procesos lo comparten), en vez de realizar el duplicado, se comparte la región retardando la copia de cada página hasta que ésta se modifique por parte de alguno de los procesos. Esta técnica será tanto más ventajosa cuantas menos páginas de la región sean modificadas por los procesos involucrados. El peor caso correspondería con una situación en la que todos los procesos afectados modifican todas las páginas de la región. Nótese que podría parecer que en este caso el resultado en cuanto a eficiencia es igual que el que se obtiene con la solución no perezosa (es decir, la duplicación directa), ya que el número de

copias es el mismo. Sin embargo, con el COW se producen activaciones del sistema operativo cada vez que un proceso modifica por primera vez una página de la región (con una región de p páginas que usan P procesos puede haber hasta $P \times p$ activaciones del sistemas operativo por fallos de tipo COW) con la sobrecarga correspondiente.
En el sistema de gestión de memoria se utilizan otras técnicas de carácter perezoso como el uso de esquemas sin preasignación de *swap*, aunque no se analizan en esta solución. Por otro lado, hay que resaltar que, a diferencia de lo que ocurre en otros niveles de la jerarquía de memoria, como en la memoria cache, el uso de la escritura diferida en la memoria virtual no se puede considerar como una técnica perezosa, puesto que es la única solución factible. El uso de escritura inmediata no es admisible en este nivel de la jerarquía por razones evidentes (un fallo de página y una escritura en disco por cada escritura en memoria es absolutamente inviable).

c) Con la técnica planteada en el enunciado, cuando hay un primer fallo de página que corresponde con una región de tipo anónima, en vez de buscar un marco libre y rellenarlo con ceros por motivos de seguridad, se hace que el proceso haga referencia a esta página llena de ceros y de sólo lectura. Aplicando la idea del COW, se espera a que se produzca una primera modificación para reservar un marco independiente y rellenarlo con ceros.
Como es fácil apreciar, esta técnica es efectiva cuando hay páginas en una región anónima que nunca se modifican. Por otro lado, el peor caso se produce cuando un proceso provoca primero un fallo por un acceso de lectura a la página, que causa que se haga referencia al marco común lleno de ceros, y un luego un fallo de COW por un acceso de escritura, que genera la reserva del marco libre y su rellenado con ceros. Con una solución no perezosa, aunque se habrían hecho las mismas operaciones sobre la memoria, sólo habría existido una activación del sistema operativo, que correspondería con el primer acceso de lectura.

d1) Dado que el código del sistema operativo no modifica los registros, no es necesario salvarlos al producirse una activación del mismo. Dado que sólo se modifican estos registros desde código de usuario, la salvaguarda se puede diferir hasta que se produzca un cambio de contexto, como se verá en el próximo apartado, pero no al entrar y salir de una llamada.

d2) Cada *thread* debe de mantener su propia copia de los registros y, por tanto, en caso de que los use, habrá que salvarlos y restaurarlos cada vez que esté involucrado en un cambio de contexto. Esto es aplicable tanto a los *threads* del mismo proceso como de distintos.
En cuanto a los procesos/*threads* de núcleo, al ejecutar sólo código del sistema operativo, no usan los registros de coma flotante. En consecuencia, no se requiere salvar ni restaurar estos registros cuando se produce un cambio de contexto entre este tipo de procesos.

d3) Cuando se produce la llamada *fork*, el proceso creado es un duplicado del proceso padre y, por ello, el contenido de sus registros debe de ser una copia de los del padre. Nótese que si se puede determinar que el padre no ha usado los registros, no sería necesario hacer esta copia puesto que el valor de estos registros no es significativo.
En cuanto a la creación de un *thread*, en este caso también el valor inicial será una copia de los registros de coma flotante asociados al *thread* que realiza la operación de creación, siempre que su valor sea significativo.

d4) A continuación, se plantea una posible estrategia similar a la usada en Linux. La estrategia consiste en controlar dos aspectos de cada proceso en cuanto a su uso de los registros de coma flotante. Por un lado, es necesario saber si el proceso ha usado estos registros durante su último turno de ejecución. Si es así, habrá que salvarlos antes de que los pueda usar otro proceso. Por otro lado, se requiere conocer si el proceso ha usado alguna vez los registros de coma flotante durante su ejecución. En caso afirmativo, cada vez que vuelva a usarlos habrá que restaurar su valor. Acto seguido, pasamos a describir la propuesta planteada.

- Nuevos campos en el BCP
  - *Utiliza_Flot*: ¿el proceso ha usado alguna vez los registros de coma flotante desde que se inició? Si es así habrá que asegurar que siempre que vuelve a usarlos tienen los valores adecuados.
  - *Mod_Flot*: ¿el proceso ha modificado los registros de coma flotante durante su última racha de ejecución? En caso afirmativo, hay que salvarlos en el cambio de contexto.

- Rutinas que se deben modificar
  - Rutina de tratamiento de la excepción de “Dispositivo no disponible”
  - Rutina de cambio de contexto
  - Habría otras, como *fork* y *exec*, que sólo se comentarán de forma breve.

- Pseudo-código de la rutina de tratamiento de la excepción de “Dispositivo no disponible”

```
Si (pactual->Utiliza_Flot)
      restaurar registros de coma flotante de BCP
TS=0; // para evitar más excepciones
pactual->Mod_Flot=TRUE;
pactual->Utiliza_Flot=TRUE;
```

- Nuevo código en la rutina de cambio de contexto

```
Si (pprevio ->Mod_Flot) // pprevio es el proceso que va a dejar la CPU
      salvar registros de coma flotante en BCP
TS=1; // para que el primer acceso cause una excepción
pprevio->Mod_Flot=FALSE;
resto del código del c. de contexto (guardar registros generales, etc.)
```

- Otras rutinas:
  - En *fork* se copian los valores de *Mod_Flot* y *Utiliza_Flot* del padre; además, si está activo *Utiliza_Flot*, se copian los registros de coma flotante del padre.
  - En *exec* se inician a 0 los campos *Mod_Flot* y *Utiliza_Flot*, puesto que se inicia la ejecución de un nuevo programa desde cero y los valores de todos los registros dejan de ser significativos.

Este esquema será beneficioso para procesos que no usen los registros de coma flotante (nos ahorramos salvarlos y restaurarlos) o que los utilicen pocas veces (se minimizan salvaguardas y restauraciones). Sin embargo, puede ser perjudicial para un proceso que siempre que ejecuta usa los registros de coma flotante, ya que no nos ahorramos ninguna operación y tenemos la sobrecarga de una activación adicional del sistema operativo (correspondiente a la excepción) por cada turno de ejecución del proceso.

Por último, hay que resaltar que se podría plantear una solución más perezosa retardando la salvaguarda de los registros de un proceso hasta que realmente otro proceso los usara (es decir, la salvaguarda se realizaría en el tratamiento de la excepción en vez de en el cambio de contexto). Esta solución retarda la salvaguarda hasta el último momento, lo que minimiza el número de veces que se realiza esta operación. Sin embargo, complica un poco la gestión, puesto que hay que acordarse de qué proceso fue el último que los usó para salvarlos en su BCP (pero quizás ese proceso ya haya terminado por entonces…).