Examen de junio de 2006

Ejercicio 1 (4,5 puntos)

Abordemos una vieja y agria polémica en el campo del diseño de sistemas operativos: A la hora de incluir una nueva funcionalidad que no requiere estrictamente ejecutar en modo núcleo y que puede tener una cierta complejidad, ¿es mejor añadirla como código del núcleo o como una biblioteca de usuario? (Sirva como ejemplo Windows NT, que, en su diseño original, realizaba el tratamiento gráfico en modo usuario, mientras que versiones posteriores lo incluyeron en el núcleo). A continuación, se analizan diversos aspectos de este debate:

a) **[Puntuación: 1,5/10]** Compare cómo se desarrolla el software en cada una de estas dos soluciones, estudiando los cinco aspectos siguientes: el grado de complejidad de (i) la programación del nuevo software, (ii) de su inclusión en el sistema y (iii) de su depuración, así como (iv) la necesidad de cambios en el software ante nuevas versiones del sistema operativo y (v) la fiabilidad del sistema resultante.

b) **[Puntuación: 1,5/10]** Algunos detractores de la solución basada en código de núcleo argumentan que puede empeorar el tiempo de respuesta de los procesos (es decir, el tiempo que transcurre desde que se desbloquea un proceso de máxima prioridad hasta que comienza a ejecutar) al incluir llamadas al sistema complejas y largas. Explique si es válido ese argumento analizándolo para núcleos expulsivos y no expulsivos.

c) **[Puntuación: 1,5/10]** Suponga que para incluir una cierta funcionalidad en el núcleo hay que añadir código en una determinada rutina de interrupción de manera que su duración se alarga considerablemente. Explique qué consecuencias tiene esta situación, analizándola para núcleos expulsivos y no expulsivos. ¿Qué técnica usaría para paliar este problema?

d) Para hacer una comparación cuantitativa, se plantea el ejemplo de añadir al sistema una función `CopiaFichero`, que reciba como argumentos los descriptores de fichero correspondientes al origen y al destino, y realice la copia. Suponga que el tamaño del fichero es de *T* bloques, y que la cache de bloques usa *delayed-write*, está inicialmente vacía y tiene suficiente capacidad para albergar las dos copias que quedarán en la cache al final de la operación. Se pretende calcular los siguientes valores:

- Número de activaciones del S.O. (***AC***), lo que incluye llamadas al sistema operativo, tratamientos de interrupción (sólo se considerarán las del disco) y fallos de página (sólo se tendrán en cuenta los vinculados con el acceso a los ficheros, presentes en la tercera implementación, apartado **d3**).
- Número de cambios de contexto voluntarios (***CC***).
- Cantidad de datos, medida en bloques, que se leen y escriben en memoria (***BD***). Nótese que una lectura de un bloque de disco implica una escritura en memoria de 1 bloque, una escritura en disco conlleva una lectura de 1 bloque, mientras que una copia de un bloque entre dos zonas de memoria implica una lectura de 1 bloque y una escritura de 1 bloque.

Estos valores se calcularán para las siguientes implementaciones de `CopiaFichero`:

d1) [Puntuación: 1,5/10] Se implementa como una llamada al sistema.

d2) [Puntuación: 1,5/10] Se implementa como una función de biblioteca de usuario que utiliza operaciones de lectura y escritura de 1 bloque.

d3) [Puntuación: 1,5/10] Se implementa como una función de biblioteca de usuario que proyecta en memoria el fichero origen y usa operaciones de escritura de 1 bloque para escribir en el destino. Suponga que el tamaño de página es igual al del bloque.

d4) [Puntuación: 1/10] Realice un análisis comparativo de los resultados obtenidos.

e) **[Sin puntuación]** ¿Tiene algo más que alegar a favor de una u otra solución que no se haya tratado en los apartados anteriores?

Solución

a) El desarrollo de software destinado a ser incluido en el núcleo del sistema operativo es, en todos los aspectos, mucho más complejo que el de usuario.

(i) En primer lugar, con respecto a la programación, hay, al menos, dos aspectos que dificultan la programación de software en modo núcleo: por un lado, hay ciertas partes que se deben de desarrollar en ensamblador; por otro lado, aunque, en su mayor parte, se trabaja con un lenguaje convencional (habitualmente, en C), se dispone de un subconjunto bastante reducido de las bibliotecas estándar del lenguaje, lo que dificulta apreciablemente la programación.

(ii) En cuanto a la inclusión del nuevo software, en el caso de la biblioteca de usuario, se trata de algo bastante directo: basta con compilar y montar los ficheros implicados y poner a disposición de los usuarios la biblioteca resultante. Sin embargo, con la solución basada en software de núcleo, hay que generar una nueva versión del sistema operativo que incluya ese nuevo software, lo que resulta bastante complejo y lento pues precisa su re-compilación. Algunos sistemas operativos ofrecen la posibilidad de incluir la nueva funcionalidad como un módulo de núcleo que se carga dinámicamente sin necesidad de detener el sistema, aunque, normalmente, esta opción sólo puede usarse de forma restringida (generalmente, sólo permite añadir manejadores de dispositivos).

(iii) Por lo que se refiere a la depuración, la del código del núcleo es más dificultosa, no sólo por su mayor complejidad intrínseca, sino por la falta de herramientas de depuración adecuadas. El entorno de depuración para el código de un determinado sistema operativo, en caso de existir puesto que algunos sistemas operativos no lo ofrecen, no proporciona la misma funcionalidad que los depuradores de código usuario. Esta limitación es lógica ya que un depurador convencional puede usar todos los servicios del sistema operativo subyacente, mientras que, evidentemente, esto no es posible en un depurador del código del sistema operativo.
(iv) Con respecto a la repercusión que tiene en el nuevo software la actualización de la versión del sistema operativo, tendrá mucho mayor impacto si se ha incluido en el núcleo. En caso de tratarse de una biblioteca de usuario, sólo se verá afectada si ha cambiado la interfaz del sistema operativo. Esta circunstancia no es nada habitual pues rompería la compatibilidad, teniendo como resultado que aplicaciones de usuario previamente desarrolladas dejen de funcionar, lo que no es admisible. La interfaz puede extenderse pero, normalmente, se mantendrá la compatibilidad. En el caso de software de núcleo, tanto las estructuras de datos del sistema operativo como la propia interfaz interna pueden cambiar entre versiones, quedando afectado todo el software que las utiliza, que tendrá que modificarse. Estos cambios pueden deberse a diversos factores tales como la corrección de errores detectados en una versión previa, o mejoras en la eficiencia y en la seguridad.
(v) Por último, en cuanto a la fiabilidad del sistema resultante, las repercusiones de cualquier error de programación son mucho peores si se trata de código de núcleo. En caso de software en modo usuario, el error sólo afectará a la aplicación que lo produce, mientras que en modo núcleo la repercusión puede ser desastrosa pudiendo causar pérdidas de datos. Haciendo un símil, el desarrollo en modo núcleo es como trabajar de trapecista sin usar una red de protección: cualquier error es fatal. Incluso un error que puede parecer relativamente inofensivo como el causar pérdidas (“goteras”) de memoria tiene mucha más repercusión en modo núcleo puesto que la memoria que se pierde es memoria residente.

b) Cuando se desbloquea un proceso que tiene mayor prioridad que el que está ejecutando, el sistema operativo activa una interrupción software para diferir el cambio de contexto involuntario hasta el momento propicio, que corresponderá al instante en que se ejecute la rutina de tratamiento de la interrupción software.
En el caso de un núcleo no expulsivo, las prioridades están establecidas de manera que la interrupción software no puede interrumpir la ejecución de una llamada al sistema. Por tanto, si en el momento del desbloqueo del proceso de máxima prioridad hay otro proceso ejecutando una llamada al sistema, la interrupción software generada en el desbloqueo tendrá que esperar a que termine la llamada en curso o se produzca un cambio de contexto voluntario dentro de la misma. En caso de que la llamada sea larga, como plantea el enunciado, el tiempo de respuesta será apreciable, lo que será especialmente negativo en aplicaciones interactivas o con un perfil de tiempo real.
En un núcleo expulsivo, la interrupción software generada en un desbloqueo de un proceso de mayor prioridad sólo tendrá que esperar a que terminen las rutinas de interrupción anidadas que pueda haber en ese momento. Si había un proceso ejecutando una llamada al sistema, ésta se verá interrumpida, produciéndose el cambio de contexto involuntario. Por tanto, en este caso no es aplicable el argumento expuesto en el enunciado en contra de la inclusión de nueva funcionalidad como código de núcleo.

c) Dado que mientras se ejecuta una rutina de interrupción de un determinado nivel están inhibidas las interrupciones del mismo nivel y de niveles inferiores, es importante minimizar la duración de la rutina. En caso de que, como plantea el enunciado, una nueva funcionalidad requiera añadir código a una determinada rutina de interrupción de manera que su duración se alargue considerablemente, habrá que analizar cuáles de las nuevas operaciones son realmente críticas y requieren ejecutarse en el entorno de la interrupción y cuáles no. Estas últimas pueden diferirse, eliminándose de la rutina de interrupción propiamente dicha, y ejecutándose en el contexto de una interrupción software.

d) Nótese que en todos los casos se producirán T operaciones de lectura del disco, con los consiguientes T cambios de contexto voluntarios y T interrupciones del disco. Sin embargo, no se produce ninguna operación de escritura ya que se usa una política *delayed-write* en la cache. Por otro lado, no se ha considerado el posible acceso a los bloques indirectos de los ficheros dado que afectaría de la misma forma en los tres casos planteados.

d1) Si se implementa como una única llamada, ésta comenzará intentando acceder en la cache al primer bloque del fichero origen. Al no estar presente, programará la operación de lectura del disco usando como destino cualquier bloque de la cache, por estar ésta vacía, y se bloqueará. La interrupción que indica el fin de la operación de disco desbloqueará al proceso, que, cuando vuelva a ejecutar, continuará realizando la llamada. Al proseguir, la llamada copiará los datos del bloque de la cache que se acaba de leer a otro bloque libre de la cache y pasará, a continuación, a leer el siguiente bloque, que provocará el acceso a disco y el bloqueo correspondiente, y, así sucesivamente para todos los bloques. Nótese que, al final de la operación, en la cache deben de quedar dos copias de cada bloque puesto que se trata de dos ficheros distintos.
A continuación, se detallan cuáles serían los valores pedidos en este caso:

- Activaciones del sistema operativo: 1 llamada al sistema y T interrupciones de disco.

- Cambios de contexto voluntarios: *T*.
- Accesos a memoria: *T* escrituras en memoria por la lectura del disco y, debido a la copia entre bloques de la cache para duplicar el contenido del fichero, *T* lecturas y *T* escrituras, totalizando 3*T*.

d2) Si se implementa como una función de biblioteca que accede de forma convencional al fichero, la operación comenzará con una llamada al sistema para la lectura del primer bloque del fichero origen. Esta llamada intentará acceder en la cache a ese bloque. Al no estar presente, programará la operación de lectura del disco usando como destino cualquier bloque de la cache, por estar ésta vacía, y se bloqueará. La interrupción que indica el fin de la operación de disco desbloqueará al proceso, que, cuando vuelva a ejecutar, copiará los datos al *buffer* de usuario especificado en la llamada de lectura y terminará la llamada. A continuación, la función de biblioteca realizará la llamada de escritura especificando dicho *buffer*. Esta llamada copiará los datos del *buffer* a un bloque libre de la cache y terminará. La función de biblioteca repetirá este proceso por cada bloque.
A continuación, se detallan cuáles serían los valores pedidos en este caso:

- Activaciones del sistema operativo: 2*T* llamadas al sistema y *T* interrupciones de disco.
- Cambios de contexto voluntarios: *T*.
- Accesos a memoria: *T* escrituras en memoria por la lectura del disco; *T* lecturas y *T* escrituras por las copias de los bloques desde la cache hasta el buffer de usuario en las sucesivas lecturas; *T* lecturas y *T* escrituras por las copias de los bloques desde el buffer de usuario hasta la cache en las sucesivas escrituras; en total 5*T*.

d3) Si se implementa como una función de biblioteca que accede al fichero origen usando una proyección, la operación comenzará con la llamada que provoca la proyección del archivo origen. A continuación, la función de biblioteca realizará directamente una llamada de escritura especificando como *buffer* la zona de memoria proyectada que corresponde con el primer bloque del fichero. Cuando dentro de la llamada de escritura, el sistema operativo intente acceder al *buffer* recibido, se producirá un fallo de página, que activa al sistema operativo de forma anidada. En el tratamiento del fallo de página, se programará la operación de lectura del disco y se bloqueará. La interrupción que indica el fin de la operación de disco desbloqueará al proceso, que, cuando vuelva a ejecutar, terminará el tratamiento del fallo de página y continuará con la llamada de escritura, que copiará los datos a un bloque libre de la cache y terminará. Este proceso se repetirá por cada bloque, terminando con una llamada al sistema para eliminar la proyección del fichero origen.
A continuación, se detallan cuáles serían los valores pedidos en este caso:

- Activaciones del sistema operativo: 2 llamadas para crear y eliminar la proyección, *T* llamadas al sistema para la escritura, *T* fallos de página y *T* interrupciones de disco.
- Cambios de contexto voluntarios: *T*.
- Accesos a memoria: *T* escrituras en memoria por la lectura del disco; *T* lecturas y *T* escrituras por las copias a la cache en las sucesivas escrituras; en total 3*T*.

d4) A partir de los resultados obtenidos, que se pueden extrapolar a otros tipos de funcionalidad, se observa que la estrategia de implementar la nueva funcionalidad como una llamada al sistema presenta un mejor rendimiento, básicamente, debido a dos factores:

- Un menor número de activaciones del sistema operativo. Aunque se producen el mismo número de interrupciones, la gran diferencia está en el número de llamadas al sistema, que en este caso sólo es una. Hay que resaltar que una activación del sistema operativo implica un cambio de modo usuario a sistema en su inicio y un retorno a modo usuario al terminar, lo que implica una sobrecarga apreciable.
- Una menor cantidad de datos transferidos. Al tratarse de una única llamada no es necesario que los datos se copien entre *buffers* del sistema y de usuario, y viceversa.

En cuanto a las estrategias basadas en una función de biblioteca, presentan un número de activaciones del sistema operativo similar. Sin embargo, con respecto a la cantidad de datos transferidos, es significativamente mejor la solución basada en proyección de archivos. Es interesante resaltar que con esta solución se evitan copias entre *buffers* del sistema y de usuario, puesto que el programa de usuario trabaja con la misma zona de memoria que el sistema operativo, paliándose el segundo factor identificado previamente.

e) Como recapitulación final, se puede afirmar que no hay una solución mejor que otra. Del análisis realizado, se puede resaltar que las soluciones basadas en modo núcleo tienden a ser más eficientes pero menos flexibles y fiables que las basadas en modo usuario. Para terminar este análisis de forma ilustrativa, pensemos en dos ejemplos extremos de estas dos estrategias. En un extremo estaría eliminar del núcleo del sistema operativo todo lo que no requiera estrictamente ejecutar en modo privilegiado. Esta es la solución adoptada por la arquitectura microkernel (y, más todavía, por el nanokernel), en la que el código en modo núcleo es mínimo y los servicios del sistema operativo se implementan como servidores en modo usuario. Aunque conceptualmente y en cuanto a la calidad del sistema resultante es una solución superior a la arquitectura tradicional

monolítica, los hechos demuestran que su impacto ha sido menor que el esperado cuando surgió como la que se suponía que iba a ser la arquitectura dominante en el campo de los sistemas operativos. El motivo de este relativo fracaso lo hemos observado en el análisis previo: problemas de eficiencia debidos, entre otros problemas, a la cantidad de información que hay que copiar entre los distintos subsistemas.

En el otro extremo estaría la inclusión de toda la funcionalidad como código en modo núcleo. Supongamos, por ejemplo, que el servidor web estuviera incluido completamente dentro del núcleo del sistema operativo. Posiblemente, se conseguiría una solución más eficiente, pero, como se analizó en el primer apartado, el desarrollo y mantenimiento del servidor web sería un infierno y afectaría a la fiabilidad de todo el sistema.