Sistemas Informáticos
Curso 2005-2006

ESTUDIO SOBRE LA RELEVANCIA DE LOS CONTADORES DE RENDIMIENTO EN SITUACIONES DE CONFLICTO

Darío Tórtola Navarro
Carlos Vives Lafuente

Dirigido por:
Francisco Javier Crespo Yáñez
Natalia López Barquilla
Dpto. Sistemas Informáticos y Programación (SIP)

Facultad de Informática
Universidad Complutense de Madrid

Se autoriza a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado en este trabajo

Carlos Vives La Fuente

Darío Tórtola Navarro

Contenidos

Resumen

Actualmente los análisis del rendimiento de sistemas informáticos se basan en métodos comparativos. Estos métodos, aunque han permitido grandes avances, tienen ciertas limitaciones. Este trabajo propone medios para obtener análisis cuantitativos del rendimiento.

Estos medios permitirán facilitar el acceso a los datos de rendimiento y su posterior análisis, consiguiendo que este acceso sea realizable por usuarios no necesariamente expertos. Al mismo tiempo abren la posibilidad de estudiar la capacidad de prever situaciones de conflicto, y otorgar esta capacidad a los propios ordenadores. Esto llevaría en un futuro a la posibilidad de diseñar e implementar sistemas informáticos autoadministrados, con el consiguiente avance sobre la tecnología actual.

En el trabajo, además, se muestran problemas generales del campo del análisis del rendimiento, así como específicos de los sistemas operativos estudiados, Microsoft Windows y Linux. Este estudio aumenta los motivos para revisar la organizacióm de los datos de rendimiento en los sistemas operativos actuales. Por supuesto, en el trabajo no sólo se critican las soluciones actuales, sino que se proponen soluciones alternativas para resolver estos problemas.

Palabras clave: Rendimiento, Benchmark, Sistemas operativos, Datos de rendimiento, Sistemas autoadministrados.

Abstract

Nowadays, the analyses on the field of computer systems performance are based on compartive methods. These methods have made possible important achievements, but they also suffer several limitations. This project proposes ways to obtain quantitative analyses of performance.

These means will simplify the access to performance data and its subsequent analysis, in order to make this access approachable by non expert users. The quantitative analysis methods also give the chance to study the ability of prediction of conflict situations on computers, and even to hand this ability over the very computers. This could lead in a future to the possibility of design and implement self-management computer systems, and could result in a technology improvement.

This project also shows general problems on the field of performance analysis, as well as specific problems of the studied operating systems, Microsoft Windows and Linux. This information supports the motivations to revise the current organization of performance data in operating systems. This project not only questions the current solutions, but proposes alternative means to solve these problems.

Keywords: Performance, Benchmark, Operating system, Performance data, Self-management systems.

0. Introducción

1. Objetivos

El objetivo fundamental de este trabajo es el estudio de la medición del rendimiento de un computador, entendiendo como tal el nivel de actividad de los distintos elementos funcionales que forman parte de él. Dentro del alcance de este trabajo, un elemento funcional es cualquier elemento, software, hardware o combinación de ambos, que realiza una función dentro del equipo. Los citados niveles de actividad serán descritos mediante el valor de diversos "datos de rendimiento". Los datos de rendimiento son diversas medidas realizadas sobre un ordenador, que proporcionan información acerca de sus elementos funcionales. En este estudio se analizará la fiabilidad de los datos de rendimiento, así como sus modos de uso. En cuanto a éstos, se analizan los datos de rendimiento presentes en los sistemas operativos Microsoft Windows y en Linux.

En los sistemas operativos Windows los valores de los datos de rendimiento son accedidos a través de los denominados contadores de rendimiento. En Linux, esos valores están contenidos en determinados ficheros del directorio /proc. En ambos sistemas operativos la cantidad de datos de rendimiento es lo bastante grande como para que sea necesaria una organización de éstos. En la investigación que se realizará para este trabajo se analizará la optimalidad de la organización actual de los datos de rendimiento, optimalidad en cuanto a facilidad de localización y uso de los datos de rendimiento por parte de los usuarios. En caso de encontrar dificultades para el desarrollo de tales actividades por parte de los usuarios, el desarrollo del trabajo no se limitará a una crítica de los métodos actuales sino que deberá estudiar posibles soluciones que eliminen o disminuyan tales dificultades.

Para que sea valiosa la información que se obtenga a partir de los datos de rendimiento deberá garantizarse la fiabilidad de éstos. Tal garantía exige un estudio en profundidad que determine la precisión de las mediciones que se toman en el computador. Para realizar este estudio se requerirán metodologías de medición del rendimiento distintas a las actuales: éstas utilizan benchmarks diseñados para utilizarse de modo comparativo, basados en las diferencias con ciertos parámetros de referencia. A pesar de que los benchmarks han hecho posibles importantes avances en la tecnología de computadores, no son la herramienta ideal para el objetivo que se propone en este trabajo.

En el diseño de los benchmarks actuales se parte de parámetros de referencia con los que se comparan los objetos analizados, pudiendo ser éstos desde elementos funcionales a computadores completos. Estos parámetros de referencia deben especificarse adecuadamente para que los resultados del benchmark posean un verdadero valor como elemento de estudio del rendimiento. La dificultad en el diseño de benchmarks es que con el aumento de la complejidad del objeto de referencia aumenta la cantidad de variables a definir hasta una cantidad no manejable por seres humanos.

Por otro lado, el uso de puntos de referencia abre la puerta a manipulaciones con fines publicitarios. En realidad, existen benchmarks diseñados para optimizar sus resultados con determinados sistemas. Es el caso del iComp, diseñado por Intel para evaluar el rendimiento de procesadores, y cuyo punto de referencia es cambiado

periódicamente para permitir obtener mejores resultados gracias a las nuevas tecnologías. Esta práctica no podría realizarse con un sistema de análisis cuantitativo ya que el punto débil, los parámetros de referencia, no se encuentran presente en el método de análisis.

El análisis cuantitativo del rendimiento de un computador permitirá realizar el estudio de la precisión y fiabilidad de los datos de rendimiento. Para esto, sin embargo, se requerirán ciertas capacidades adicionales: el conocimiento de los fundamentos teóricos del funcionamiento de los elementos funcionales del sistema, y la capacidad de diseño de bancos de pruebas específicos para hacer uso de las funciones de esos elementos. A partir del conocimiento teórico del funcionamiento de un elemento funcional se diseña el banco de pruebas. De este modo, puede preveerse en teoría los valores que tendrán los datos de rendimiento al ejecutar ese código. Tras la ejecución, registrando los valores de los datos de rendimiento durante la misma, los resultados pueden analizarse y compararse con los esperados, permitiendo así evaluar la precisión de los datos de rendimiento.

2. Metodología

Para satisfacer los objetivos detallados en el primer punto debe seguirse una serie determinada de pasos para que las conclusiones del trabajo puedan tener una validez científica.

2.1 Estudio del estado del arte

En el trabajo se analizará de modo detallado el modo en el que los datos de rendimiento están organizados en computadores que trabajan bajo sistemas operativos Linux y Windows.

En el sistema operativo Linux los datos de rendimiento se encuentran contenidos en archivos del sistema de archivos virtual /proc. El sistema de archivos /proc se comporta como un directorio corriente de archivos desde el punto de vista del usuario, pero según la documentación consultada, en realidad su contenido se calcula bajo petición, ya sea ésta del usuario del computador o de una aplicación. La estructura del sistema de archivos /proc está definida en el kernel de Linux, pero los contenidos son realmente virtuales en el sentido de que no están almacenados.

En el sistema de archivos /proc no sólo están presentes los datos de rendimiento, sino también otros datos del sistema como opciones de configuración, información sobre componentes hardware instalados, etc.

En el sistema operativo Windows los datos de rendimiento se encuentran localizados a través de los denominados contadores de rendimiento. El uso de los datos de rendimiento en Windows da la impresión de estar reservado a usuarios con un nivel adecuado de conocimientos debido a la complejidad de la documentación al respecto y el tiempo y entrenamiento necesario para acceder a su análisis.

Una vez que se haya estudiado la organización y metodología de localización de los datos de rendimiento, se detallarán los medios de acceso disponibles actualmente para obtenerlos. Cualquier servicio del sistema operativo Linux para leer archivos normales puede utilizarse para leer los archivos virtuales del directorio /proc, incluidas

las funciones habituales de entrada y salida de archivos de las librerías C. Además, existen numerosas aplicaciones para obtener los valores de los datos de rendimiento. Estas aplicaciones, sin embargo, presentan ciertos problemas para permitir un estudio tan profundo como el que se busca permitir en este trabajo:

· Las aplicaciones para Linux son creadas por más personas todavía que las que modifican el núcleo o los servicios del sistema operativo. Eso significa que el control y la capacidad de mantenerse actualizado son prácticamente imposibles de alcanzar.

· Estas aplicaciones no son desarrolladas para examinar todos los datos de rendimiento presentes, sino sólo aquellos seleccionados por los desarrolladores de las aplicaciones.

· La precisión de tales aplicaciones es generalmente fácil de poner en duda ya que no se presentan estudios de carácter científico para justificar las capacidades de los programas desarrollados.

Por lo que respecta a Windows, es necesario utilizar aplicaciones, librerías o métodos propietarios para acceder a los valores de los datos de rendimiento. Esto quita al desarrollador control sobre los modos de obtención de los datos de rendimiento. Al mismo tiempo, si la accesibilidad de los usuarios a las aplicaciones de monitorización no era lo bastante alta, menos aún lo es el acceso a las librerías y los demás métodos actuales, al igual que la información sobre su uso. Este hecho dificulta la actividad de los desarrolladores, al menos de los que no pertenecen a la empresa Microsoft.

Las características de los medios de acceso en ambos sistemas operativos podrá justificar la búsqueda de nuevos medios de acceso a los datos de rendimiento, búsqueda que se estudiará en este trabajo.

2.2 Comprobación del funcionamiento del acceso a los datos de rendimiento

El acceso a los datos de rendimiento no es, por sí solo, suficiente para permitir un estudio adecuado del funcionamiento de éstos. Esto es debido a ciertos problemas que deben analizarse y tenerse en cuenta durante la planificación, realización, y análisis de resultados de pruebas:

a) Espacio: El análisis de los resultados de las pruebas puede necesitar realizarse a través de ciertas metodologías que exigen registros de los valores de los datos, y no de sólo del valor actual. Esto es especialmente cierto cuando son análisis de cierta profundidad y complejidad, fuera de la capacidad humana. Si hay que registrar estos datos, es necesario un espacio de almacenamiento para ello, lo que implica el límite que imponga el espacio disponible.

b) Tiempo: Aunque el problema anterior reviste importancia en casos como el que se busca permitir con este trabajo, existe otra dificultad que actúa también sobre los análisis basados en valores en tiempo real, y es precisamente el tiempo. Obtener los valores de los datos de rendimiento, y procesarlos como se estime necesario en cada caso, gasta tiempo. Este tiempo puede imponer un límite sobre la frecuencia de muestreo de los datos, y teniendo en cuenta que estos valores pueden cambiar cada pocos milisegundos, este problema puede generar una pérdida de información.

c) Fiabilidad: Dando por supuesto que los valores de los datos de rendimiento estén correctamente medidos y calculados en los sistemas operativos, los dos problemas anteriores crean una nueva dificultad: La fiabilidad. Las exigencias de tiempo y espacio

de aplicaciones de monitorización pueden unirse a las necesidades de ocupación de recursos de estas aplicaciones, y este conjunto limita la fiabilidad de los resultados, en cuanto a que cambia los valores de los datos de rendimiento del sistema estudiado. El principio de incertidumbre de Heissenberg, ideado inicialmente para la posición de los electrones en un átomo, tiene su particular versión en el estudio de los datos de rendimiento. En cuanto a física atómica, el principio de Heissenberg se basa en que los instrumentos que pueden localizar la posición de un electrón utilizan energías que varían esa posición, y por tanto el resultado que obtienen es una posición que no es la que debía ser. Debido a ello no puede trabajarse con la posición exacta de los electrones sino más bien con una suposición sobre la posición. En el caso de los datos de rendimiento el problema no es absoluto en cuanto a que la incertidumbre no es completa. Sin embargo, sí es cierto que la fiabilidad de los datos depende del diseño de las herramientas para obtenerlos. Debido a que las propias herramientas modifican los valores de los datos de rendimiento, los valores que se obtengan pueden no ser exactamente los que serían si no estuviera actuando la aplicación de monitorización. Este problema de fiabilidad debe ser estudiado adecuadamente para poder realizar un estudio con la suficiente validez científica.

2.3 Testeo con bancos de pruebas

Para desarrollar herramientas adecuadas de prueba para los sistemas informáticos es necesario un conocimiento muy detallado del funcionamiento de cada componente del sistema. Este estudio no está al alcance de este trabajo, ya que por sí solo exigiría bastante más tiempo que el estimado para la duración de la asignatura para la que se realiza este trabajo. Sin embargo, con los conocimientos obtenidos en el estudio de Ingeniería Informática, y analizando las herramientas actuales de prueba de elementos funcionales de computadores, bastará para iniciar este estudio.

El estudio que se permite en este trabajo puede determinar la fiabilidad de los datos de rendimiento disponibles en los sistemas operativos estudiados. Esto se realiza a través de tres fases:

a) Análisis teórico de los efectos que el código de la prueba debería tener sobre los datos de rendimiento

b) Ejecución de las pruebas un número suficiente de veces como para que se pueda obtener un resultado representativo

c) Análisis de los resultados obtenidos

Sin embargo, las posibilidades que abre este trabajo no se limitan a eso. Si se une al estudio de la fiabilidad el estudio de la importancia relativa (al objetivo del estudio) de cada dato de rendimiento y de las capacidades predictivas que pueden tener los datos de rendimiento con respecto a situaciones de conflicto por los recursos del sistema, se puede abrir la puerta a sistemas informáticos autoadministrados, con la innovación que ello supondría. Considérense sólo los sistemas de computación distribuida: La posibilidad de que los propios ordenadores puedan decidir dónde se debe realizar qué función puede permitir no sólo una computación más rápida, sino una administración de tareas más veloz, compleja y, probablemente, correcta, que la que pueda realizar un ser humano. Esto sería así por que el computador podría considerar más datos que el administrador humano, establecer más rápidamente el significado de esos datos, calcular antes las órdenes necesarias para administrar el sistema de computación, y transmitir esas órdenes.

3. Evaluación

La consecución de los objetivos definidos en este trabajo será un elemento de evaluación del mismo. Expuestos brevemente, serán:

· Estudiar la medición del rendimiento de un computador

· Analizar la optimalidad de los medios de acceso actuales

· En caso de detectar dificultades para el acceso al análisis del rendimiento por parte de los usuarios, se estudiarán las causas de tales dificultades así como medios que las solucionen.

· Se estudiará la precisión de los datos de rendimiento como herramienta de análisis del rendimiento.

· Se analizará la posibilidad de métodos cuantitativos de análisis del rendimiento para complementar los métodos comparativos utilizados actualmente.

Debido al estado actual del campo que ocupa este trabajo, los resultados deberían conducir a evidencia suficiente para justificar investigaciones futuras que partan de los resultados alcanzados.

La motivación de este trabajo es doble:

· Facilitar el acceso al análisis de rendimiento.

· Analizar la posibilidad y utilidad de métodos cuantitativos de análisis del rendimiento de computadores.

Si se dispusiera de la posibilidad de realizar análisis cuantitativos del rendimiento de los sistemas informáticos podría aumentarse la información obtenida. La importancia de este aspecto puede empezar a intuirse con una sencilla idea:

Actualmente la información sobre el rendimiento de los computadores que está disponible consiste sólo en afirmaciones como “El procesador A debería ejecutar este código en un 75% del tiempo que tarda el procesador B” Supóngase que al ejecutar el código el procesador A tarda el 80% del tiempo que utiliza el B. Según los métodos de análisis actuales, basados en comparación, estos resultados indicarían que el procesador A está trabajando por debajo de su nivel óptimo, pero nada más. Ahora supóngase que se realiza de nuevo la prueba y se incluyen exámenes cuantitativos de la actividad de los procesadores, determinándose que el procesador A sólo dedica al código del orden del 90% de su capacidad de proceso mientras que el B le dedica el 100%. Con la inclusión de estos resultados pueden analizarse las causas del aparente bajo funcionamiento.

Este tipo de análisis cuantitativo es la posibilidad que se pretende abrir en este trabajo.

El análisis cuantitativo, como no requiere de la definición de unos parámetros de comparación, abre la posibilidad de realizar análisis del comportamiento conjunto de varios elementos funcionales de un ordenador; este tipo de análisis actualmente no se realiza debido a la limitación que tienen las metodologías comparativas, y que se analizarán en detalle en este trabajo.

1. Proceso de obtención de datos para la realización del proyecto en equipos con sistema operativo Windows 2000, XP

0. Introducción

Para la realización del proyecto es necesario obtener datos sobre el funcionamiento y uso de recursos de un ordenador. Esos datos serán llamados datos de rendimiento (performance). Los datos de rendimiento son diversas mediciones que pueden realizarse en un computador y proporcionar información acerca del funcionamiento y aprovechamiento de sus elementos funcionales. Para analizar este aprovechamiento se hará uso de benchmarks como se verá en el capítulo 4. En la literatura consultada no se ha encontrado una definición precisa de "elemento funcional", pero para los efectos de este trabajo, "elemento funcional" es un elemento que realiza una función, independientemente de si es un componente hardware, software, o combinación de ambos.

La cantidad de datos de rendimiento que se puede obtener de un computador y su disparidad es lo bastante grande como para requerir una taxonomía que facilite su localización y uso. En este trabajo desarrollaremos un modelado del sistema actual de organización y una taxonomía que lo recubra. Esta taxonomía debe ser realizada de modo que sea fácilmente extensible, para dar cabida a datos de rendimiento que surgieran, por ejemplo, por la aparición de un nuevo elemento funcional, para asegurar esa escalabilidad, la taxonomía debe estar fundamentada en reglas específicas.

1.- Los datos de rendimiento bajo Sistemas Operativos Microsoft

En los ordenadores que funcionan bajo sistemas operativos Microsoft, en concreto Windows 2000 y Windows XP, se registran los datos de rendimiento en los denominados contadores de rendimiento. [MSDN1]

Según la documentación de Microsoft, los contadores de rendimiento son métodos de registro de datos de rendimiento. Algunos contadores de rendimiento registran los resultados de operaciones aritméticas con otros datos de rendimiento. Los datos de rendimiento que se obtienen de mediciones simples en el sistema son intuitivamente más útiles y necesarios. Disponer de contadores de rendimiento para datos de rendimiento secundarios no es tan necesario ni útil en cuanto a que dichos datos pueden ser fruto de cálculos.

Se llamará dato de rendimiento primario a un dato tal como ha sido definido en el primer párrafo. Se llamará dato de rendimiento secundario a los resultados de operaciones con datos de rendimiento primarios.

Fruto de la experimentación realizada para este trabajo se ha encontrado que no es exacto decir que los contadores de rendimiento son métodos de registro, ya que los valores de algunos de ellos en realidad se calculan cuando se realiza la consulta, y no están realmente almacenados.

En cuanto a los datos de los contadores de rendimiento hay que hacer notar que Microsoft no hace distinciones entre datos de rendimiento primarios y secundarios. Los motivos para la inclusión de contadores para datos de rendimiento cuya necesidad es menos intuitiva, no han sido publicados. Debido precisamente a la inclusión de datos de

rendimiento secundarios aumenta la cantidad de información que debe conocer un usuario interesado en la captura de datos de rendimiento, lo que dificulta su objetivo. De hecho, Microsoft permite que incluso los usuarios creen sus propios contadores de rendimiento y no sólo los fabricantes de hardware y software.

1.1 Uso de los datos de rendimiento

Los datos de rendimiento se utilizan habitualmente para obtener información sobre el funcionamiento de una aplicación, servicio, controlador, etc. Así mismo, según la documentación de Microsoft, también permiten ayudar a localizar "cuellos de botella" (bottlenecks) en el sistema y ajustar el rendimiento de éste y de las aplicaciones que se ejecuten en él. [MSDN1]

En la documentación de Microsoft se explica, igualmente, que un desarrollador de software puede preparar sus aplicaciones para que obtengan y utilicen los datos de los contadores de rendimiento con el fin de determinar los recursos del sistema mínimos que deben consumir para mejorar el rendimiento global del sistema. Una posible aplicación de esta idea sería un gestor de descargas de archivos para internet que detectase cuánto del ancho de banda está siendo pedido por otras aplicaciones de forma que la competencia por los recursos sea mínima y el funcionamiento de la aplicación sea lo más transparente y efectivo para el usuario, entendiendo como "más efectivo" una mejora del rendimiento. [MSDN1]

Los datos de los contadores de rendimiento no se utilizan solamente en tiempo real, sino que también pueden guardarse en ficheros de registro para después ser procesados por otras aplicaciones como se desarrollará en este trabajo. [MSDN1]

Para acceder a los datos de rendimiento, los sistemas operativos de la serie Microsoft Windows disponen de métodos propietarios en distintas librerías, como pueden ser Windows Management Instrumentation, Performance Data Helper o el Interfaz del Registro de Windows, entre otras. El uso de estos métodos requiere de usuarios capacitados.

2. Organización de los contadores de rendimiento

2.0 Necesidad de una organización

En un ordenador que trabaje bajo el sistema operativo Windows XP puede haber del orden de varios miles de contadores de rendimiento distintos, en cuanto a sus significados y los datos que registran. Al mismo tiempo, existen contadores de rendimiento en puntos distintos de la organización que, en realidad, contienen el mismo dato de rendimiento. Como se explicó en la introducción, los fabricantes de hardware y software pueden crear sus propios contadores de rendimiento, e incluso los usuarios (con los conocimientos adecuados), pueden crear sus propios contadores de rendimiento. Esta cantidad de contadores, y la posibilidad que tiene de aumentar, exige la creación de una taxonomía de los contadores de rendimiento, ya que se necesita un mayor control sobre el dominio de los elementos.

La cantidad de contadores de rendimiento va a requerir de una taxonomía para facilitar el acceso a usuarios no expertos. El trabajo de localizar un dato de rendimiento adecuado para un objetivo particular es más rápido si se dispone de una taxonomía

puesto que las reglas que fijan ésta están bien definidas. El estudio de una solución a la necesidad de organización de los contadores de rendimiento se desarrollará en este trabajo.

2.1 Modelo de organización de los contadores de rendimiento

A partir del modo de localización de los datos de los contadores de rendimiento y basándose en la documentación consultada y la experimentación realizada, los autores presentan una nueva conceptualización que simplifica la organización que ha utilizado Microsoft. Recuérdese que sólo es una presentación del modelo. Posteriormente el lector podrá encontrar un desarrollo de las mismas con explicaciones más completas y complejas.

La organización de los contadores de rendimiento presentada considera varios niveles jerárquicos, basados en los siguientes conceptos.

a) Nivel de identificación de red
b) Nivel de identificación de computador
c) Nivel de identificación de objeto de rendimiento
d) Nivel de identificación de instancia de un objeto de rendimiento
e) Nivel de identificación de contador de rendimiento

a) Nivel de identificación de red

Para realizar la captura de datos de rendimiento el universo está formado por el ordenador a través del que se realiza la captura y todos los ordenadores conectados a él. (Fuera de ese universo, no hay datos de rendimiento accesibles)

El dominio de los contadores de rendimiento que es necesario clasificar es, por tanto, el conjunto de todos los contadores de rendimiento presentes en algún ordenador del citado universo.

b) Nivel de identificación de computador

La primera división del dominio de los contadores de rendimiento se realiza según el ordenador particular en el que están localizados los contadores a consultar. Esta división permite establecer subconjuntos del dominio asociados a cada ordenador accesible.

En un computador existen distintos tipos de elementos funcionales. Como se explicó en la introducción, en la documentación de Microsoft no se ha encontrado una definición precisa de elemento funcional. Éste podría, por tanto, ser un concepto utilizado *ad hoc* para agrupar los contadores de rendimiento. Para el propósito de la organización realizada en este trabajo se seguirá entendiendo “elemento funcional” como se especificó en el primer párrafo de la introducción: un elemento que realiza una función, independientemente de si es un componente hardware, software, o combinación de ambos.

El conjunto de contadores de rendimiento asociados con un computador es disjunto con el asociado a cualquier otro ordenador, ya que los datos que contienen los contadores de rendimiento de un ordenador están asociados con ese ordenador, que es en el que se realizaron las mediciones. La propia estructura resultante de organizar ambos conjuntos ni siquiera tiene porqué tener la misma forma. Esto es porque esa

estructura se basa en los elemento funcionales de cada ordenador, y éstos no tienen porqué tener los mismos componentes.

En este nivel se divide el conjunto de contadores de rendimiento asociados al ordenador en conjuntos asociados a los tipos de elementos funcionales presentes en el ordenador. Esto debe hacerse de modo que los datos de rendimiento contenidos en los contadores asociados a un objeto de rendimiento sean mediciones tomadas sobre elementos funcionales del tipo representado por el objeto. Esta relación no está explícita en la documentación de Microsoft.

c) Nivel de identificación de objeto de rendimiento

Un objeto de rendimiento es una abstracción que representa un tipo de elemento funcional de un ordenador, como pueden ser los procesadores, los procesos, o las conexiones activas en el computador.

Los objetos de rendimiento se dividen en dos tipos:

· Objetos de rendimiento simples

Hay tipos de elementos funcionales de los que como máximo puede aparecer un representante en un computador determinado. Estos tipos de elementos funcionales se representan mediante objetos de rendimiento simples. Un ejemplo de uno de estos tipos es la memoria de un computador.

Un objeto de rendimiento simple tiene asociado el elemento funcional, presente en el ordenador, perteneciente al tipo representado por el objeto.

En la documentación de Microsoft se denomina a los objetos de rendimiento simples como “objetos de rendimiento que no soportan múltiples instancias”. En la misma documentación, además, se identifica un objeto de rendimiento simple con un elemento funcional y no con un tipo de elemento funcional, lo que resta homogeneidad a la organización.

· Objetos de rendimiento múltiples

Por otro lado, hay tipos de elementos funcionales que pueden contar con varios representantes en un computador. Estos tipos se representan a través de objetos de rendimiento múltiples. Un ejemplo de este tipo sería el procesador.

Un objeto de rendimiento múltiple tiene asociado el conjunto de elementos funcionales representado por el objeto.

En la documentación de Microsoft se denomina a los objetos de rendimiento múltiples como “objetos de rendimiento que soportan múltiples instancias”.

En la organización de los contadores de rendimiento que sigue Microsoft, el conjunto de contadores de rendimiento asociado a un objeto de rendimiento no es necesariamente disjunto del asociado a un objeto distinto. Por ejemplo, los objetos de rendimiento Objetos y Sistema tienen acceso al mismo dato de rendimiento a través de un contador llamado Procesos.

En este nivel, se divide el conjunto de contadores de rendimiento asociado a un objeto de rendimiento entre los elementos funcionales del ordenador que pertenecen al tipo representado por el objeto. Esto se hace de modo que los datos de rendimiento contenidos en los contadores asociados a un elemento funcional sean mediciones

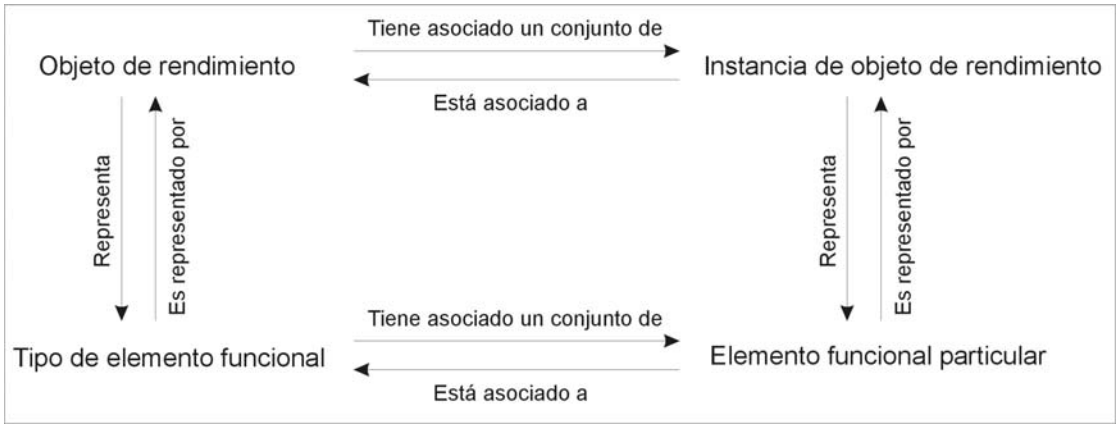

Figura 1: Relaciones entre objeto de rendimiento, instancia de objeto, elemento funcional y tipo de elemento funcional

realizadas sobre ese elemento. De nuevo, esta idea no está explícitamente en la documentación de Microsoft consultada.

El hecho de que todos los elementos funcionales del mismo tipo comparten sus características esenciales. Esto implica que el conjunto de mediciones que puede obtenerse sobre el funcionamiento de elementos del mismo tipo es común a esos elementos. Por consiguiente, la estructura de la organización de los contadores de rendimiento asociados a un elemento funcional debe ser igual a la de los asociados a otro elemento del mismo tipo. Sin embargo, en la organización de Microsoft esto no se puede asegurar debido a la libertad que se ha dado para la creación de nuevos contadores de rendimiento.

d) Nivel de identificación de instancia de un objeto de rendimiento

Una instancia de un objeto de rendimiento representa un elemento funcional del tipo representado por el objeto. Si se atiende a la división de objetos de rendimiento entre simples y múltiples, la instancia de un objeto simple representa el único elemento funcional del tipo representado por el objeto de rendimiento que está presente en el ordenador. Por otro lado, una instancia de un objeto de rendimiento múltiple representa uno de entre los elementos funcionales presentes en el sistema, que pertenecen al tipo representado por el objeto de rendimiento.

Como se indicaba en el punto anterior, en la documentación de Microsoft se usa una definición distinta para objetos de rendimiento simples y múltiples. Según ésta se considera que un objeto de rendimiento simple no tiene instancias sino que él mismo representa el elemento funcional correspondiente presente en el ordenador. Esto reduce la homogeneidad de la organización, dificultando su procesamiento.

e) Nivel de identificación de contador de rendimiento

Un contador de rendimiento representa un dato de rendimiento. En la documentación de Microsoft se define contador de rendimiento como un método de registro de un dato de rendimiento, a pesar de que como se explicó en la introducción, los valores que algunos de ellos “registran” se calculan en realidad cuando se realiza la consulta, y no están realmente calculados y almacenados.

3. Localización de un contador de rendimiento

Microsoft permite el acceso a los contadores de rendimiento mediante funciones a las que se debe proveer de un parámetro que identifica al contador de rendimiento deseado. Este parámetro se denomina “ruta de acceso al contador de rendimiento” o “ruta de acceso al contador”. [MSDN1, PDHREF1, WIN32-1]

Para establecer la ruta de un contador de rendimiento se utiliza un subconjunto de los siguientes localizadores, según sean necesarios:

a) Localizador del computador
b) Localizador del objeto de rendimiento
c) Localizador de la instancia de objeto
d) Localizador del contador de rendimiento

a) Localizador del computador

Como se explicó en la sección 1, el universo está compuesto por el ordenador a través del que se realiza la obtención de datos y los ordenadores conectados con él. En el universo es necesario identificar el ordenador al que corresponde el contador de rendimiento deseado.

El modo de localizar un ordenador es mediante su nombre o su dirección IP. En Windows se considera que el ordenador por defecto es el que realiza la captura de datos, haciendo así opcional el localizador de computador cuando se desea acceder a un contador de rendimiento del propio ordenador accesor. [MSDN1]

De este modo se pasaría del nivel de identificación de red al nivel de identificación de computador.

b) Localizador del objeto de rendimiento

En el nivel de identificación de computador se debe identificar el objeto de rendimiento al que está asociado el contador de rendimiento deseado. Cada objeto de rendimiento posee un nombre que lo identifica y lo diferencia de los demás objetos de rendimiento en el mismo ordenador. Ese nombre sirve como localizador. [PDHREF1 y MSDN1]

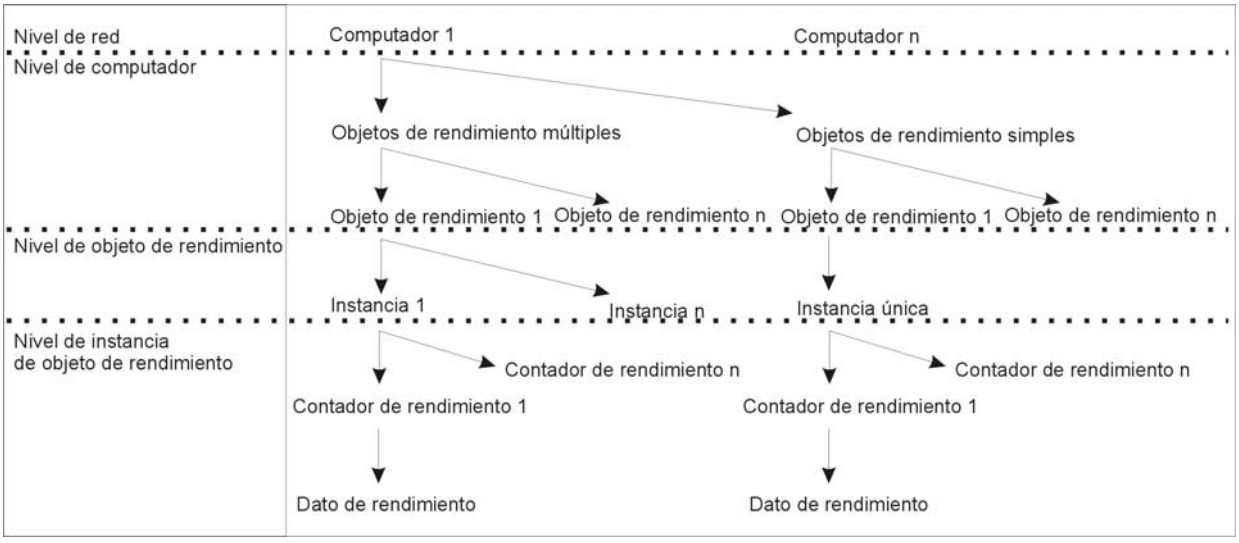

Figura 2: Esquema de la clasificación de mayor usabilidad de los contadores de rendimiento

A diferencia del identificador de computador, que puede ser opcional, el identificador del objeto de rendimiento siempre es de necesaria aparición en la ruta de datos.

A través del localizador del objeto de rendimiento se pasa del nivel de identificación de computador al nivel de identificación de objeto.

En el sistema operativo Windows, si el objeto de rendimiento es simple se pasa al nivel de identificación de contador de rendimiento; si el objeto es múltiple, se pasa al nivel de identificación de instancia de objeto.

c) Localizador de la instancia de objeto

En el nivel de identificación de instancia de objeto de rendimiento caben dos posibilidades:

· El objeto de rendimiento es simple

Sólo puede existir una instancia del objeto en el computador, y por tanto se puede considerar dicha instancia como instancia por defecto. Ya se explicó en el punto 1.b que Microsoft considera que el objeto de rendimiento es equivalente a su única instancia. Por consiguiente, el localizador de la instancia de objeto no aparece en la ruta de datos. [PDHREF1 y MSDN1]

· El objeto de rendimiento es múltiple

Si esta es la situación, es necesario localizar la instancia a la que está asociado el contador de rendimiento buscado, incluso aunque sólo exista una instancia del objeto en el ordenador particular. Para localizar una instancia son necesarios entre uno y tres de los siguientes sub-localizadores:

c.1) Nombre de la instancia del objeto

c.2) Índice de la instancia del objeto

c.3) Nombre del padre de la instancia del objeto

[PDHREF1 y MSDN1]

c.1) El nombre de instancia de objeto es el localizador principal, lo que le hace necesario para localizar correctamente cualquier instancia de un objeto de rendimiento múltiple.

c.2) Existen casos en los que un objeto de rendimiento puede dar lugar a dos instancias con el mismo nombre. En estas ocasiones debe especificarse el **índice de la instancia del objeto.** Este índice es un numero que localiza a una repetición en particular entre instancias del mismo objeto que tengan el mismo nombre de instancia de objeto.

En las pruebas realizadas y en la literatura consultada no ha sido encontrado ningún ejemplo en el que sea necesario el índice de instancia de un objeto salvo en los casos de los objetos de rendimiento Procesos y Subprocesos. Las instancias del objeto de rendimiento Procesos tienen un nombre dependiente de la aplicación o programa lanzado en el ordenador. Si se lanza a ejecución una aplicación y, antes de cerrarla, se lanza de nuevo, hay dos instancias de dicha aplicación coexistiendo y por tanto dos instancias del objeto Procesos con el mismo nombre. Para distinguir los contadores de rendimiento asociados a una u otra se hace necesario el índice de instancia de objeto. El caso del objeto Subprocesos se verá en el punto siguiente. [PDHREF1 y MSDN1]

c.3) Hay casos en los que la instancia de un objeto de rendimiento guarda una cierta relación con una instancia de otro objeto de rendimiento distinto. El **nombre del**

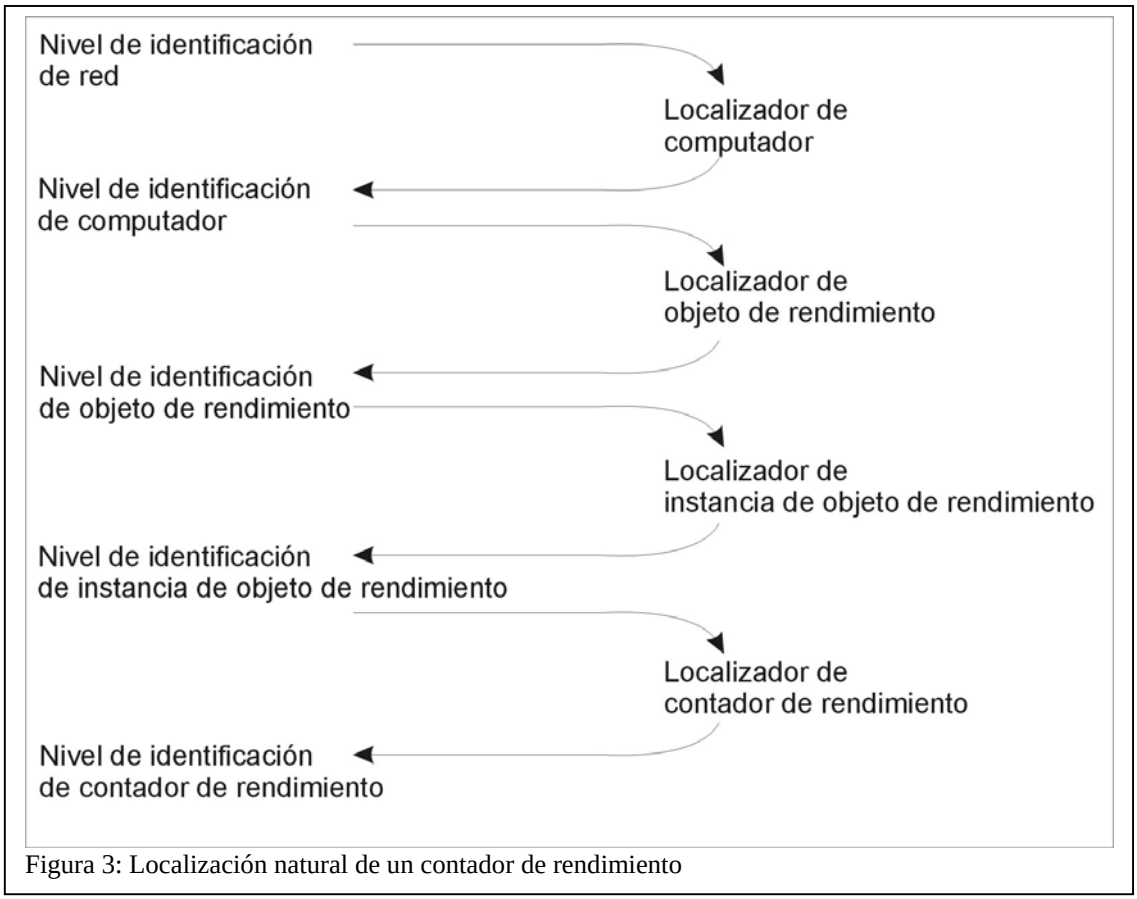

Figura 3: Localización natural de un contador de rendimiento

padre de la instancia del objeto es de necesaria aparición para especificar perfectamente la instancia del objeto particular al que se asocia el contador de rendimiento que se debe localizar.

Al igual que en sucedía con el punto c.2 no se ha encontrado ni en los experimentos realizados ni en la literatura consultada ningún ejemplo en el que sea necesario el nombre del padre de una instancia salvo el del objeto de rendimiento "Subprocesos". Una instancia del objeto de rendimiento Procesos representa un proceso en particular. Cuando ese proceso crea subprocesos, aparecen éstos como instancias del objeto de rendimiento Subprocesos. Las instancias del objeto Subprocesos que representan subprocesos de procesos distintos se distinguen porque el nombre del padre de la instancia del objeto Subprocesos es distinto para cada una. Por otro lado, si un proceso crea varios subprocesos, o varios procesos con el mismo nombre crean subprocesos, debe aparecer en las instancias de Subprocesos el índice de instancia de objeto. [PDHREF1 y MSDN1]

En la documentación de Microsoft se dice que, a pesar de esto, existe la opción de concatenar el identificador de proceso al nombre de una instancia del objeto Procesos y el identificador de subproceso al nombre de una instancia del objeto Subprocesos. Esta opción haría innecesarios los índices de instancia de objeto y el nombre de instancia del padre, pero es modificando el Registro de Windows como se activa la opción, y por ello requiere de usuarios con ciertos conocimientos.

Mediante la identificación de instancia de objeto se pasa del nivel de identificación de objeto de rendimiento al nivel de identificación de contador de rendimiento.

d) Localizador del contador de rendimiento

En el nivel de instancia de objeto de rendimiento debe localizarse el contador deseado entre todos los contadores asociados con la instancia particular. Esta localización se realiza mediante el nombre del contador de rendimiento. Estos nombres son únicos dentro de los contadores asociados a una instancia de un objeto de rendimiento.

Los nombres de contadores de rendimiento no son únicos en los conjuntos asociados a objetos de rendimiento múltiples, e incluso hay casos de contadores de rendimiento con el mismo nombre, asociados a distintos objetos de rendimiento y distintas instancias de objeto. Para ilustrar esto, baste observar que existen contadores de rendimiento llamados “% de tiempo de procesador” asociados a las instancias de los objetos de rendimiento Procesador, Procesos y Subprocesos, y que cada uno de los datos de rendimiento a los que se accede son distintos.

Aunque un objeto de rendimiento sólo tuviera un contador de rendimiento asociado, es necesario indicar el nombre del contador de en la ruta del contador de rendimiento.

A través del localizador de contador de rendimiento se pasa del nivel de identificación de instancia de objeto de rendimiento al nivel de identificación de contador.

El carecer en la documentación consultada de una definición exacta de elemento funcional, unido al hecho de que los elementos del nombre de instancia “índice de instancia” y “nombre de instancia del padre” sólo se usen, hasta donde ha sido posible comprobar, en los objetos “Procesos” y “Subprocesos” refuerza la impresión de los autores de que tanto la organización de los contadores de rendimiento en conjuntos asociados a objetos de rendimiento como la especificación de las rutas de acceso a los contadores han sido realizadas *ad hoc*.

4. Métodos de acceso a los datos de rendimiento

Para acceder a los datos de rendimiento se requiere de ciertas interfaces. En la literatura consultada se han encontrado varias de estas interfaces, requiriendo distintos niveles de conocimientos:

a) Aplicaciones ofrecidas por Microsoft
b) El Interfaz del Registro de Windows
c) La librería Performance Data Helper (PDH)
d) El conjunto de librerías de Windows Managemente Instrumentation

a) Aplicaciones ofrecidas por Microsoft

Existen varias aplicaciones de Microsoft que realizan capturas de los datos de rendimiento, aunque todas tienen un problema común, dado el objetivo de este trabajo: la incapacidad de realizar capturas con la suficiente frecuencia. Se presentan aquí las aplicaciones más conocidas:

1) Performance Monitor

Incluida con Microsoft Windows a partir de la versión NT, la aplicación Performance Monitor puede mostrar y almacenar estadísticas sobre recursos del sistema como procesadores, memoria o procesos.

Permite establecer un conjunto de contadores de rendimiento que monitorizar para presentar sus datos en tiempo real o guardarlos en un archivo de log, que después puede leer y mostrar. Las mediciones son periódicas, con precisión de intervalos del orden de un segundo.

Según la documentación de Microsoft, Performance Monitor hace uso de las funciones de la librería Performance Data Helper. [MSDN1]

2) System Monitor

System Monitor es el API que se usa para configurar el control ActiveX Microsoft System Monitor. Éste control permite ver datos de contadores de rendimiento tanto en tiempo real como desde ficheros de log. El hecho de que sea un componente ActiveX implica que puede usarse, por ejemplo, desde un fichero HTML. [MSDN1]

b) El Interfaz del Registro de Windows

El Interfaz del Registro de Windows contiene ciertas funciones a través de las cuales pueden conseguirse las rutas de acceso a los contadores de rendimiento, y de ese modo a los datos almacenados en ellos. Sin embargo, en la documentación de Microsoft se recomienda no hacer uso de las funciones del registro para obtener y procesar datos de contadores. En cambio, la opción que se sugiere es utilizar para ese fin las funciones de la librería Performance Data Helper. [MSDN1]

A pesar de ello, los autores consideraron que tenía interés experimentar con esta opción para intentar ahorrar posibles ineficiencias de otros métodos y trabajar directamente sobre la fuente de los datos. Sin embargo, se vio que las funciones del Interfaz del Registro de Windows sólo proporcionan las rutas de acceso a los contadores, y que sigue siendo necesario utilizar otros métodos propietarios para acceder a los contadores de rendimiento. Debido a esto, la alternativa fue abandonada. En el anexo 1.2 se ofrece una explicación más detallada del proceso de captura de las rutas de los contadores de rendimiento, así como el código utilizado.

c) La librería Performance Data Helper (PDH)

La librería Performance Data Helper contiene funciones de acceso a las rutas de los contadores de rendimiento, así como a los datos de éstos. Según la documentación consultada, las funciones de PDH utilizan funciones del Interfaz del Registro de Windows o de WMI.

El modo que tienen las funciones de PDH de acceder a los datos de rendimiento es a través de “consultas”. Una consulta consiste básicamente en una serie de datos de acceso a contadores de rendimiento. Cuando se decide realizar la captura de valores, se obtiene el contenido de los contadores de rendimiento cuya información de acceso se encuentra en la consulta. [MSDN1, PDHREF1]

d) El conjunto de librerías de Windows Managemente Instrumentation

Windows Management Instrumentation es según Microsoft un componente del sistema operativo Windows a través del cual puede obtenerse información de gestión y

control en un entorno empresarial. Es la implementación por parte de Microsoft de WBEM, y trabaja sobre el Common Information Model (CIM). Hay disponible más información acerca de WBEM y de CIM en el anexo 1.1.

Los administradores de sistemas pueden usar WMI para consultar y establecer información en ordenadores, aplicaciones, redes y otros componentes empresariales. Los desarrolladores pueden usarlo para crear aplicaciones de monitorización que avisen a los usuarios cuando ocurran ciertos eventos.

El propósito de WMI es proveer de unos medios estándar para gestionar un sistema de computadores, ya sea un único ordenador o todos los de una empresa. Cuando se habla de gestión aquí, debe entenderse capacidad de acceder a información sobre el estado de un objeto controlado dentro del sistema y también a la alteración de dicho estado cambiando los datos almacenados sobre él. Este objeto controlado puede ser un elemento hardware como un disco duro, o un elemento software, como un archivo de paginación.

WMI permite monitorizar la cantidad de espacio libre que hay en el disco, y también, de modo remoto, alterar el estado de ese disco borrando archivos, cambiando los niveles de seguridad o formateando el disco. Usando WMI puede también crearse una aplicación que monitorice un sistema informático, que proporcione alertas cuando ocurran ciertos eventos, y que permita a un administrador controlar diferentes aspectos de ese sistema. [MSDN1]

En cuanto a la utilización de WMI, hay varias interfaces de uso entre las que pueden contarse C++, ODBC (Open DataBase Conectivity), Microsoft Visual Basic, HTML e incluso cierto tipo de scripts. [MSDN1]

WMI puede utilizarse en cualquier aplicación basada en Windows a partir de la versión 2000, aunque es más útil en aplicaciones empresariales. En cuanto al desarrollo, está orientado al uso de C/C++, Microsoft Visual Basic o un lenguaje de script que tenga un motor en Windows y pueda gestionar objetos del tipo Microsoft ActiveX. Para aquellos que utilicen C++ será necesaria cierta familiaridad con programación orientada a COM. [SWT2]

Otra característica de WMI es que almacena los datos que obtiene usando un estándar público, el CIM (Common Information Model), del que se habla en el Anexo número 1. [SWT2]

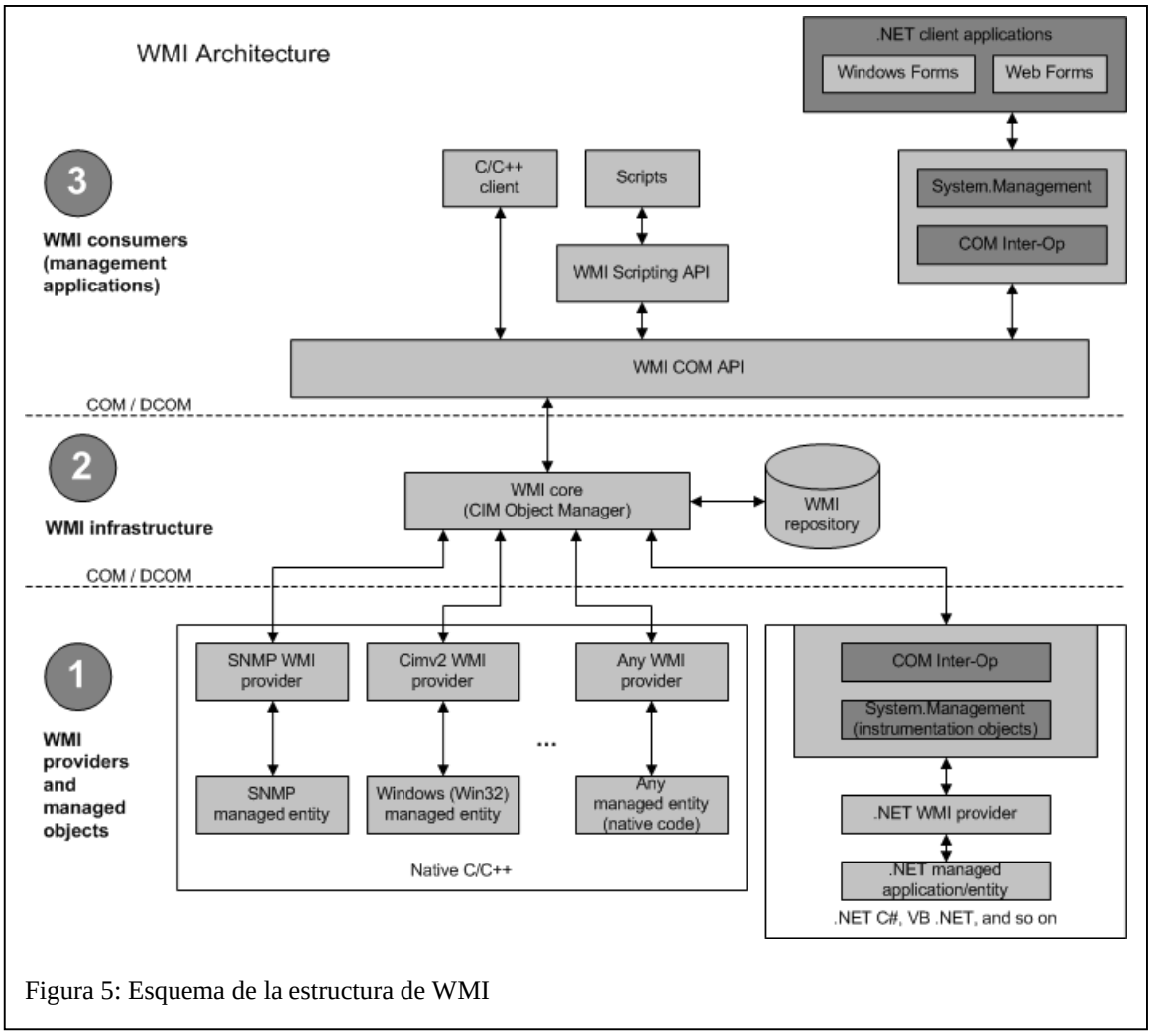

Figura 5: Esquema de la estructura de WMI

A continuación se explican ciertos elementos relevantes de WMI cuyo conocimiento se considera importante para la comprensión del funcionamiento de Windows Management Instrumentation. Estos datos son útiles para entender cómo funciona Windows Management Instrumentation:

- Proveedores y Objetos gestionados:

Un **Proveedor** es un objeto COM que monitoriza uno o más objetos gestionados para WMI. De forma similar a un controlador, el Proveedor suministra a WMI datos sobre el objeto gestionado y también maneja los mensajes de WMI a dicho objeto.

Un **Objeto gestionado** es un componente lógico o físico de un sistema informático, como un disco duro, un adaptador de red, un sistema de base de datos, un sistema operativo, un proceso o un servicio. Los objetos gestionados se comunican con WMI a través de un Proveedor WMI, que llama a métodos en el API COM para WMI.

[MSDN1]

- Infraestructura de WMI:

Es un componente del sistema operativo Windows formado por dos componentes: El servicio de Windows Management, incluyendo el núcleo de WMI, y el WMI Repository. El Windows Management service actúa como intermediario entre los proveedores, aplicaciones de gestión y el WMI Repository. En éste sólo se almacenan datos estáticos sobre los objetos, como las clases definidas por los proveedores. WMI

obtiene la mayor parte de los datos directamente del proveedor y de forma dinámica, cuando un cliente lo pide.

[MSDN1]

- Aplicaciones y scripts WMI

Una aplicación o script de gestión es una aplicación que interactúa con la infraestructura WMI. Una aplicación puede consultar o enumerar datos de gestión llamando o bien al API COM para WMI o el Scripting API para WMI. Las aplicaciones pueden usar métodos de ambas API para enviar instrucciones o reconfigurar un objeto gestionado. Los únicos datos o acciones disponibles para un objeto gestionado, como un disco duro, son los que proporcione el proveedor.

[MSDN1]

WMI accede a los datos a través de los proveedores que son responsables de manejar la comunicación entre WMI y los datos generados por un componente hardware o software (el objeto gestionado) En el caso de los datos de rendimiento, estos datos son generados por librerías de rendimiento de acuerdo con la configuración establecida en el registro. Estos valores del registro se leen al iniciarse un servicio WMI.

WMI usa esta información para poblar su base de datos interna, el Repository, con la lista de los objetos y contadores registrados en el sistema. Las clases relacionadas con el rendimiento pueden encontrarse enumerando los hijos de la clase Win32_PerfRawData.

[SWT2]

5 Problemática de la obtención de datos de rendimiento

Antes de realizar capturas de datos de rendimiento deben analizarse ciertos problemas que surgen en el proceso de planificación de la captura:

a) Problemas de fiabilidad:

El principio de incertidumbre de Heissenberg, aunque suele aplicarse a física atómica, también tiene validez en el tema de este trabajo. La esencia de dicho principio es que hay un límite a la precisión con la que se puede medir algo porque la propia observación cambia aquello que se observa. En efecto, no es posible monitorizar los datos de rendimiento de un sistema informático sin alterarlo. Intuitivamente, el tiempo que un ordenador esté invirtiendo en la obtención de datos de rendimiento es tiempo que no está dedicando a otros procesos, y por lo tanto afecta al porcentaje de tiempo que el procesador dedica a cada proceso; igualmente puede ocurrir con otros recursos como los sistemas de memoria, el uso de los buses, de las conexiones del computador, etc.

El impacto sobre el sistema es mayor cuanto mayores sean:

· La frecuencia de captura de los datos de rendimiento.

· El conjunto de datos de rendimiento monitorizados.

· El tiempo que tarde en capturarse cada dato, más el tiempo de operación si se monitorizan datos de rendimiento secundarios.

b) Problemas de tiempo:

El tiempo genera limitaciones en cuanto que la toma de los valores de los contadores de rendimiento supone cierto retardo. Si además es necesario procesar los

datos obtenidos, implicará un retardo aún mayor. Esto limita la frecuencia de muestreo por el tiempo necesario para obtener y procesar los datos de rendimiento que estén siendo monitorizados.

El problema es más grave cuanto mayores sean:

· El conjunto de datos de rendimiento que están siendo monitorizados.

· El tiempo que tarde en capturarse cada dato, más el tiempo de operación si se monitorizan datos de rendimiento secundarios.

b) Problemas de espacio:

El espacio disponible es importante si se quieren almacenar los resultados de los muestreos. El valor de cada dato de rendimiento ocupará un cierto espacio. De nuevo, como en el caso del tiempo, el problema del espacio se agrava con el aumento del número de contadores de rendimiento monitorizados.

En la documentación consultada se califica como razonable muestrear cada quince segundos para tiempos inferiores a cuatro horas. En la misma documentación se dice que si el tiempo de monitorización sube a ocho horas el intervalo no debe ser menor de cinco minutos. Esto significa menos de mil muestras en todo el intervalo. Teniendo en cuenta los cambios que puede sufrir el estado de un ordenador en quince segundos, más aún en cinco minutos, es fácil ver el problema que representan estas limitaciones para el análisis del rendimiento.

La gravedad de este problema será mayor cuanto mayores sean:

· El número de datos de rendimiento consultados.

· La precisión deseada en los valores de los datos de rendimiento.

· El período de tiempo durante el que se toman muestras.

· La frecuencia de muestreo.

[MMCH1]

c) Problemas de disponibilidad:

La disponibilidad de las alternativas para el acceso a los contadores de rendimiento varía, aunque en este caso el problema no trabaja contra la monitorización sino contra el desarrollador o el encargado de decidir qué opción de monitorización se prefiere.

6. Decisión sobre el método de acceso a utilizar

Entre las opciones disponibles para obtener los datos de rendimiento se tienen, como se explicaba en el punto 3, las siguientes:

a) Interfaz del Registro de Windows

b) Windows Management Instrumentation

c) Performance Data Helper

d) Performance Monitor y System Monitor

a) Interfaz del Registro de Windows

El Registro de Windows tiene funcionalidades mayores que las de proporcionar rutas de contadores, y además sus funciones son de un nivel más bajo que el necesario. Sería difícil demostrar que es rentable dedicar tiempo a crear funciones de alto nivel que usen las del Registro de Windows cuando están disponibles las funciones de la librería Performance Data Helper. A pesar de la mayor complejidad de uso, la utilización de

funciones de bajo nivel ha permitido en el pasado un ajuste más fino y la obtención de menores retardos de las funcionalidades que se pretenden, lo cual da valor a esta posibilidad. Un valor añadido es la oportunidad de trabajar directamente con la fuente de la información, lo que aumentará el fondo de conocimiento de quienes lo desarrollen.

b) Windows Management Instrumentation

De nuevo se encuentra una opción con miras demasiado amplias para los objetivos marcados ya que no se pretende modificar el sistema sino únicamente monitorizarlo. Además, para el uso de WMI sería necesaria la carga del servicio WMI, un análisis posterior a esa carga, y después el uso de los proveedores adecuados para que proporcionen los datos de rendimiento.

La necesidad del uso de WQL aparece porque la orientación de WMI es menos específica que la de PDH. Esto dificulta el acceso y alarga el tiempo de desarrollo de aplicación así como el de obtención de los datos, lo cual es más importante en cuanto que el tiempo de proceso, como se vio en el punto 3, es ya un obstáculo.

Windows Management Instrumentation no trabaja contra los contadores de rendimiento, pero requiere de una estructura de información específica que no está presente, al menos por completo, en las distribuciones del sistema operativo Windows. Esto representa un problema en cuanto a la disponibilidad del método de acceso.

c) Performance Data Helper

Las funciones de la librería PDH son de mayor nivel que las disponibles en el Interfaz del Registro de Windows, lo que acelera el desarrollo de software. Además, hay que valorar el hecho de que la librería está enfocada al objetivo que se pretende.

Por otro lado, el trabajar con funciones de alto nivel disminuye el control sobre los retardos y bloquea el conocimiento sobre el funcionamiento interno de las funciones utilizadas, ya que no puede estudiarse su implementación.

d) Performance Monitor y System Monitor

Estas dos opciones son las más sencillas de utilizar, pero también tienen desventajas que pueden ser importantes:

- La primera está los intervalos de muestreo, ya que no son menores a un segundo.
- La segunda es su orientación es a mostrar los datos obtenidos a un usuario humano, lo que le quita generalidad ya que podría desearse que fuera una aplicación quien analizara los resultados de la monitorización.

Ambos inconvenientes dan al administrador una capacidad de control y monitorización que puede resultar insuficiente para la realización de sus tareas.

7. Implementación

Hubo varios intentos para implementar una aplicación que pudiera realizar la monitorización de determinados conjuntos de valores de contadores de rendimiento:

Utilizando el interfaz del registro de Windows

En la sección 3 ya se explicó que está posibilidad está desaconsejada por Microsoft, y también los motivos por los cuales los autores juzgaron que no carecía de interés el estudio de sus posibilidades. Lamentablemente, el interfaz del registro de Windows sólo permite obtener las rutas de acceso a los contadores de rendimiento, no el acceso a los datos. El acceso a estos datos debía hacerse, por tanto, por otros medios.

Por estos motivos se abandonó la alternativa en favor de otras posibilidades. El código obtenido durante el proceso se ofrece en el anexo 1.2.

Utilizando la librería PDH y código obtenido de Microsoft

En el curso del desarrollo se descubrió que el código de Microsoft poseía ciertas características que se consideraron no deseables:

· En primer lugar se encontró el uso de funciones no definidas en la librería PDH sino en librerías propias de la aplicación Microsoft Visual Studio .NET 2003.

· En segundo lugar, las funciones de la librería System que se utilizan en el código aparecen en la instalación de Microsoft Visual Studio .NET 2003, pero no se encuentran en la librería System estándar que se instala con el sistema operativo Windows.

Se considera que estas características imponen una notable falta de generalidad en el código desarrollado, ya que mientras la librería Performance Data Counter es de fácil acceso, no ocurre así si limitamos el desarrollo al uso de un entorno de desarrillo concreto. Estos motivos llevaron al abandono de la opción.

Utilizando la librería PDH

Para la utilización de la librería Performance Data Helper en el proceso de obtener datos de rendimiento se siguen los siguientes pasos:

1. Obtener las rutas de los contadores de rendimiento.

Existen funciones en PDH que sirven para obtener los elementos necesarios para construir la ruta de un contador de rendimiento:

- PdhEnumMachines () devuelve la lista de nombres o direcciones de las máquinas conectadas a la máquina local. Como para identificar un contador de la máquina local no es necesario especificar el nombre de ésta, PdhEnumMachines () no lo incluye en la lista de nombres que devuelve.
- PdhEnumObjects () devuelve la lista de los nombres de los objetos de rendimiento presentes en el ordenador que se le especifique.
- PdhEnumObjectItems () devuelve la lista de los nombres de contadores de rendimiento, e instancias en su caso, que pertenecen a un objeto cuyo nombre se proporcione. En los nombres de las instancias que se devuelven se incluye en su caso el nombre de instancia del padre, aunque no el índice de instancia.
- PdhExpandCounterPath () recibe como parámetro una ruta de contador de rendimiento en la que se permite la presencia de caracteres “comodín” en lugar de algunos de los elementos de la ruta. La función devuelve una lista con todas las rutas válidas que se obtienen de sustituir esos caracteres comodín por cada elemento adecuado. Por ejemplo, si se llama a PdhExpandCounterPath () y se le entrega como ruta una en la que se especifique el nombre de objeto y de instancia, pero en el lugar del nombre de contador aparezca un asterisco, la lista que devolverá la función (suponiendo que el nombre de

instancia corresponda a ese objeto y que éste este presente en el sistema) contendrá todas las rutas con el nombre de objeto e instancia entregados, y en las que en lugar del asterisco aparece el nombre de un contador válido.
- PdhValidatePath () comprueba la validez de una ruta que se le pase como parámetro. [PDHREF1]

2. Crear una consulta:

La creación de una consulta se realiza a través de la función PdhOpenQuery (). Esta función devuelve un manejador de la consulta que se usará más adelante. [PDHREF1]

3. Añadir contadores a la consulta:

Por cada contador de rendimiento que desee añadirse a la consulta debe usarse la función PdhAddCounter(), que requiere como parámetro la ruta del contador de rendimiento que se desee monitorizar, debiendo estar aquella completamente cualificada. [PDHREF1]

4. Obtener los datos de rendimiento:

Llamando a la función PdhCollectQueryData() y pasándole como parámetro el manejador de una consulta, se obtienen los datos de todos los contadores presentes en dicha consulta. Como ya se dijo en la sección sobre el interfaz del registro, los valores de los contadores no siempre bastan por sí mismos. Por esta razón, al llamar a PdhCollectQueryData(), no se desechan los valores obtenidos en la última vez que se realizó la llamada (si la hubo) con la misma consulta, sino que se guardan para poder usarlos en los contadores necesarios. [PDHREF1]

4. Procesar los datos de rendimiento

Para dar un formato a los datos obtenidos en el paso anterior se usa PdhGetFormattedCounterValue(). Esta función convierte los datos del contador seleccionado a formato integer, float o double según se le indique en uno de sus parámetros. La función almacena el resultado de las operaciones en una estructura del tipo PDH_FMT_COUNTERVALUE. [PDHREF1]

La clase principal de la aplicación obtenida es la siguiente:

monitor
ofstream salida HQUERY hQuery HCOUNTER *hCounter TStringList* contadores
void creaQuery () void cuentaP () void formateaDatos () void iniciaCabecera () void contadorAnadido () void acabaCabecera () void ponHora () void ponMediciones () static AnsiString codigoError ()

void inicia () Cardinal estimaTiempo () void cuenta () void para () static void devuelveObjetos () static void devuelveInstanciasContadores () static void extiendeInstancias () static AnsiString hazPathContador ()

· El atributo salida de cada monitor es el archivo donde éste vuelca las mediciones realizadas.
· El atributo hQuery es un manejador para la información sobre la consulta que el monitor está preparado para hacer. La estructura HQUERY está definida en PDH
· El atributo hCounter es una colección de manejadores de contadores, uno por cada contador introducido en la query. HCOUNTER es una estructura definida en PDH.
· El atributo contadores es una colección de AnsiString que guarda las rutas de los contadores incluidos en la query. TStringList es una estructura para colecciones de AnsiString incluida en las librerías estándar de Borland C++ Builder

· Antes de poder empezar a tomar medidas hay que realizar una llamada al método inicia(), que recibe una lista de cadenas y una cadena. La cadena contiene la ruta del archivo donde desean guardarse los datos obtenidos. La lista de cadenas contiene las rutas de los contadores que debe usar el monitor.
Este método inicia(), por su parte, usa otros métodos privados para crear la consulta, añadir a ésta los contadores, y escribir en el archivo de salida la cabecera del informe. Estos métodos son creaQuery(), iniciaCabecera(), contadorAnadido() y acabaCabecera() creaQuery es el único de esos métodos que usa funciones de la librería PDH, en particular PdhAddCounter().

· Para realizar una estimación del tiempo que se tardará en obtener las mediciones y guardarlas en el archivo se puede usar el método estimaTiempo(). Este método devuelve un tipo Cardinal (el mismo tipo que usa la clase TTimer de Borland C++ Builder para el intervalo de un reloj)
Este método realiza una simulación con los mismos métodos que usaría en el momento real, pero usando un archivo de prueba para realizar las pruebas de grabar datos.

· Cada vez que deseen medirse los contadores seleccionados en monitor debe llamarse al método cuenta(). Éste método transmite la llamada al método privado cuentaP(), que usa PdhCollectQueryData(), de la librería PDH, para recoger los valores de los contadores, y efectúa después llamadas a otros métodos:

- ponHora() para guardar en el archivo la hora y fecha en la que se han pedido los datos.
- formateaDatos(). Este método convierte los valores medidos por el sistema en valores de formato double a través de la función de la librería PDH PdhGetFormattedCounterValue(). La precisión de double es la máxima que puede ofrecer PDH, y se decide usarla para obtener la mayor precisión posible en los datos.
- ponMediciones() para guardar en el archivo los valores de los contadores. Este método no es llamada directamente por cuentaP(), sino que es formateaDatos() quien lo hace.

· Una vez que ha terminado la obtención de los datos debe liberarse la memoria ocupada y cerrarse el archivo. Para hacer esto debe llamarse al método para()

· codigoError () es un método estático que devuelve, dado un valor de error devuelto por una función de la librería PDH, una cadena con un mensaje explicatorio.

· devuelveObjetos () es otro método estático que recibe la dirección de una TStringList, llenándola con los nombres de los objetos de rendimiento que hay en el sistema. El método hace uso de la función de PDH PdhEnumObjects().

· devuelveInstanciasContadores() realiza una función parecida a la que hacía devuelveObjetos para los objetos, salvo que devuelveInstanciasContadores recibe el nombre de un objeto de rendimiento y rellena dos listas de cadenas con los nombres de las instancias y contadores de rendimiento del objeto indicado. La función de PDH de la que hace uso este método es PdhEnumObjectItems(). Este método también hace uso de extiendeInstancias().

· extiendeInstancias() Hace uso de PdhValidatePath () para obtener los índices de instancia, necesarios en las instancias repetidas. Podríamos haber usado la opción comentada en el punto 1-c.3 de este capítulo, pero sólo resolvería el problema de instancias duplicadas para el caso de Procesos y Subprocesos, cuando no podemos asegurar que no vaya a darse en otros objetos de rendimiento.

· hazPathContador() recibe cadenas con los nombres de un objeto, una instancia y un contador. Si el objeto en particular no tiene instancias, instancia debe ser igual a “”. El método crea y comprueba la ruta formada por ese objeto, instancia (si es aplicable) y contador. La comprobación de la ruta se hace a través de una función de PDH, PdhValidatePath().

· Para cambiar el formato del archivo resultado de la monitorización a través de la clase monitor deberían modificarse (o reescribirse en una clase descendiente de monitor) las siguientes funciones:

- *iniciaCabecera():* Se llama una sola vez, al empezar la inicialización de una instancia de monitor, con la creación y configuración de su hQuery.
- *contadorAnadido():* Se llama una vez cada vez que se añade un contador a la consulta de la instancia de monitor.
- *acabaCabecera():* Se llama una sola vez tras haber añadido todos los contadores de la Query y deja el archivo preparado para introducir las mediciones.
- *ponHora():* Escribe la fecha y hora en el archivo de salida.
- *ponMediciones():* Escribe las mediciones en el archivo salida.

· Debido al orden de llamadas, el informe creado tiene la siguiente estructura:
  - Datos iniciales de cabecera
  - Datos en la cabecera para cada contador en el orden en el que han sido introducidos
  - Marca de inicio de mediciones
  - Líneas formadas por Fecha y Hora - Mediciones.

Si el formato deseado tiene una estructura distinta a la mencionada, entonces deberían modificarse también las funciones cuentaP(), creaQuery() y formateaDatos(), en cuanto a que son quienes llaman a las funciones que modifican el archivo de salida.

· Una de las funcionalidades de la aplicación desarrollada es la de monitorizar una serie de contadores a intervalos fijos de tiempo durante un periodo determinado.
En PDH existe la función PdhCollectQueryDataEx(), que toma el valor de los contadores seleccionados en la consulta cada cierto intervalo que se le indica. El motivo por el que ha sido usado PdhCollectQueryData(), que sólo toma los datos en el momento de su llamada, en lugar de PdhCollectQueryDataEx(), está en que el intervalo que admite ésta última función tiene precisión de segundos, mientras que en el otro caso podemos tomar los datos con tanta velocidad como permita el sistema.

2. Proceso de obtención de datos para la realización del proyecto en equipos con sistema operativo Linux

0. Introducción

Para la realización del proyecto se van a tratar de obtener los datos de rendimiento de un computador. Como se dijo en el capítulo anterior, los datos de rendimiento son mediciones realizadas sobre un ordenador que proporcionan información acerca de sus elementos funcionales. Recuérdese que por "elemento funcional" se entiende, en este trabajo, cualquier elemento, hardware, software o combinación de ambos, que realiza una función del computador.

En computadores que operan bajo el sistema operativo Linux, los datos de rendimiento se encuentran almacenados en diferentes ficheros del sistema de archivos virtuales */proc*, que se estudiará en posteriores apartados. Uno de los principales problemas que se presentan a la hora de trabajar con los datos de rendimiento en estos computadores, es que dentro del directorio */proc* se encuentran, además de los citados datos de rendimiento, una ingente cantidad de archivos diferentes en los que se almacena información ajena al rendimiento del computador. Así, bajo el sistema de archivos virtuales */proc*, podemos encontrar datos referentes a características técnicas del equipo, elementos hardware instalados en el mismo, o incluso información acerca de la configuración del propio sistema operativo.

Esta falta de homogeneidad en el contenido del directorio */proc* hace que la información, en opinión de los autores, no esté demasiado claramente organizada. Además, al ser un sistema de archivos virtuales, su contenido se monta cada vez que se solicita información. Por este motivo podría considerarse que los datos de rendimiento no disponen de un dispositivo físico estable en el que estén almacenados.

La manera en que es presentada la información en el sistema de archivos virtuales */proc* resulta, en opinión de los autores, algo caótica. Esto hace pensar en ocasiones que haya sido improvisada ad hoc en función de necesidades puntuales de cada momento, habiendo sido ampliada sin un diseño previo en diversas etapas. Esta aparente falta de planificación previa y de generalidad en la presentación de la información relativa a los datos de rendimiento no hace sino reforzar la idea de los autores respecto a la necesidad de una nueva taxonomía global sobre los datos de rendimiento.[1]

A priori puede parecer que la propia estructura de directorios y subdirectorios ofrece la posibilidad de organizar los datos de rendimiento sin añadir nuevos conceptos. Sin embargo, los archivos del directorio */proc*, de manera muy similar a como ocurría en el caso de los sistemas operativos de Microsoft, son clasificados de acuerdo a la idea de agrupar a los archivos que contengan datos sobre una cuestión similar bajo el mismo subdirectorio; pero no se ha encontrado una especificación de qué es exactamente una

[1] Esta característica, fundamento de la taxonomía propuesta en nuestro proyecto, ya fue expuesta en el capítulo primero.

“cuestión similar”. Se puede decir que la clasificación está más dirigida a una mente humana que a un sistema de reglas, y que en cualquier caso no hay una homogeneidad que haga lo suficientemente clara la asociación de los archivos en subdirectorios dentro de */proc*.

Los datos de rendimiento del equipo se utilizan con frecuencia para analizar el funcionamiento de aplicaciones que estén ejecutándose en él, así como para detectar posibles “cuellos de botella” que se produzcan en el computador, con el fin de mejorar el rendimiento del mismo.

El posible uso de los contadores de rendimiento en computadores que operan bajo sistema operativo Linux requiere de un profundo conocimiento de la compleja estructura del directorio */proc*, puesto que la heterogeneidad del mismo no facilita la rápida obtención de la información requerida. Por este motivo, el estudio que se realice de los contadores de rendimiento dentro de este sistema operativo será más lento y deberá estar más dirigido por el desarrollador.

Además del uso que se pueda dar a los datos de rendimiento en tiempo real, estos también pueden ser almacenados para su ulterior procesamiento y análisis por parte de otras aplicaciones. Esta es una de las principales utilidades que se va a dar a los datos de rendimiento en este trabajo.

1. Localización y organización de los datos de rendimiento

En equipos que trabajan bajo el sistema operativo Linux, los datos de rendimiento se encuentran almacenados, junto con otra gran cantidad de información diferente, en el sistema de archivos virtuales */proc*. Este directorio se encuentra incluído dentro del núcleo de Linux, por lo que antes de extenderse en la descripción de su organización, y con el fin de facilitar la posterior comprensión de la misma, conviene explicar someramente algunas de las características principales del núcleo.

El núcleo de un sistema operativo es el programa que gobierna el hardware del ordenador y ofrece una interfaz de servicios al resto de programas (procesos) que se ejecutan en dicho ordenador. Normalmente, el núcleo es el primer código que se carga en memoria cuando se enciende el ordenador y permanece en ella mientras el sistema está en funcionamiento, entrando en ejecución cada vez que un proceso requiere uno de sus servicios o un dispositivo físico requiere su atención. Una de las diferencias fundamentales entre el núcleo del sistema y los procesos que se ejecutan en niveles superiores al mismo es que uno y otros se ejecutan en modos de procesador distintos (siempre que el hardware del ordenador soporte esta característica). El núcleo suele ejecutarse en alguno de los modos privilegiados del procesador, en el que tiene completo acceso a todo el hardware (incluyendo la programación de los dispositivos), mientras que los procesos se ejecutan en "modo usuario", en el que no pueden ejecutar ciertas instrucciones máquina del procesador. De esta forma, el núcleo se convierte en una especie de intermediario obligado en algunas de las acciones que los procesos ejecutan, lo que le permite implementar de forma adecuada la multitarea, repartiendo de forma equitativa y segura los recursos del sistema entre los diferentes procesos que se ejecutan en cada momento.
[CLX1]

En la mayoría de situaciones, el núcleo pasa desapercibido al usuario común (incluso al administrador), puesto que los usuarios interactúan con los programas y aplicaciones (o más correctamente con los procesos), que internamente interactúan con el núcleo. Sin embargo, existen situaciones en las que se necesita modificar el soporte ofrecido por el núcleo, para conseguir mejores prestaciones, un sistema más seguro o mejor adaptado a un uso concreto, o simplemente para poder acceder a un nuevo tipo de dispositivo físico. En otras ocasiones puede ser necesario simplemente un acceso controlado a diferentes partes del núcleo, para consultas acerca del rendimiento y otros muchos aspectos cuya información esté presente en él, sin que por ello se produzcan modificaciones en el mismo.
[CLX1]

El código ejecutable del núcleo de Linux, que se carga en memoria al arrancar el ordenador, reside en un fichero denominado vmlinux (o en su homólogo comprimido, vmlinuz). Normalmente, este fichero se encuentra en el directorio /boot/ y suele tener como sufijo la versión del núcleo que generó el ejecutable (por ejemplo, /boot/vmlinuz-2.4.22-1). Cuando se instala una distribución de Linux, el fichero binario del núcleo que incorpora la distribución es instalado en ese directorio y el cargador del sistema operativo (habitualmente, LILO o GRUB) lo carga en memoria y le ofrece el control al encender el ordenador. Este fichero binario es por tanto imprescindible para el funcionamiento del sistema.
[CLX1]

En realidad, el código binario del núcleo puede dividirse entre diferentes ficheros: el fichero binario principal vmlinuz mencionado arriba (que es absolutamente necesario) y, opcionalmente, un conjunto de ficheros denominados módulos de núcleo (kernel modules). Los módulos de núcleo son ficheros objeto (compilados de forma especial) que incorporan funcionalidades concretas que pueden ser instaladas y desinstaladas del núcleo de Linux sin tener que generar un nuevo ejecutable vmlinuz e incluso sin tener que reiniciar el ordenador. Normalmente, la misma funcionalidad puede ser incorporada al núcleo (en fase de compilación) como parte del binario principal o como un módulo de núcleo.

Tal y como se dijo unos párrafos más arriba, el núcleo de Linux incluye un sistema de archivos especial en el directorio */proc*. Este sistema de archivos contiene cierta jerarquía de archivos especiales que representan el estado actual del núcleo, permitiendo a las aplicaciones y a los usuarios observar el sistema.

Dentro del directorio */proc* pueden encontrarse datos que detallan el hardware del que dispone el sistema y cualquier proceso que esté ejecutándose. Además, algunos de los archivos en este directorio pueden ser manipulados por aplicaciones y usuarios para comunicar al núcleo cambios de configuración.

[RHM1]

El sistema de archivos */proc* no es más que un mecanismo para que el núcleo y los módulos del mismo envíen información a los procesos que haya en ejecución en el sistema. Su nombre se debe sencillamente a que en origen fue diseñado para permitir un cómodo acceso a la información sobre los procesos, aunque actualmente lo utiliza además cualquier otro elemento del núcleo que tiene algún tipo de información que mostrar, como por ejemplo los módulos existentes, si accediéramos a /proc/modules, o las estadísticas del uso de la memoria, si lo hiciéramos en /proc/meminfo.

Cuando se ha dicho que el directorio */proc* es un sistema de archivo especial y los archivos en él son especiales, es porque éstos pertenecen a una clase particular de archivo que se llama “archivo virtual” y aquel es denominado “sistema de archivos virtuales”. Es un sistema de archivos virtuales porque su contenido no está almacenado en un dispositivo físico de almacenamiento (disco duro, CD, disquete, etc) sino que se construye y presenta dinámicamente cada vez que se le pide al núcleo, y lo mismo ocurre con sus subdirectorios y archivos, de aquí el nombre “archivos virtuales” y “sistema de archivos virtuales”.

Por ser construidos bajo petición, si se lista el contenido del directorio en dos momentos diferentes, o en dos ordenadores distintos, es posible que los dos listados no coincidan exactamente; del mismo modo, es posible que el contenido de los archivos y directorios también difiera. Ello es debido a que el contenido del directorio refleja el estado actual del núcleo de Linux, y evidentemente este estado varía con el tiempo y de un sistema a otro (por ejemplo, por disponer de hardware distinto).

Hay otras características especiales que poseen los archivos en el directorio */proc*. La mayoría de ellos aparecen como teniendo un tamaño de cero bytes incluso cuando, si los leemos, podemos encontrar una gran cantidad de datos. De hecho, tan sólo los archivos kcore, mtrr y self tienen un tamaño mayor que cero. Además, la mayoría de ellos, en los campos fecha y hora reflejan la fecha y hora actuales, indicando que están siendo constantemente actualizados.

[RHM1 y CLX1]

De una manera más formal, puede definirse */proc* como una interfaz entre el núcleo de Linux y el nivel de usuario con la forma de un sistema de archivos virtual. Así, el núcleo de Linux presenta una manera única de presentar su información interna y se convierte en la alternativa a utilizar múltiples herramientas particulares. La ventaja principal es que, aunque Linux incorpora varias de estas herramientas (en algunos casos, por compatibilidad con otras versiones de UNIX), no es imprescindible disponer de una herramienta por cada información que el núcleo pone a disposición del usuario, puesto que toda esa información está en */proc*. Esto es especialmente útil en el caso de Linux, puesto que el desarrollo del núcleo es independiente del desarrollo de las herramientas y programas que se ejecutan por encima (lo que se denomina "distribución").

[CLX1]

Como consecuencia de todas las características del sistema de archivos virtuales */proc* anteriormente citadas, se puede considerar al mismo como un centro de información y control sobre el núcleo. De hecho, la mayoría de las utilidades del sistema no son más que llamadas a archivos de este directorio. Así, ‘lsmod’ es totalmente equivalente a la instrucción ‘cat /proc/modules’, mientras que ‘lspci’ es el resultado de la llamada ‘cat /proc/pci’. Nótese además, que mediante la modificación de los archivos alojados en este directorio se pueden llegar a leer y cambiar los parámetros del núcleo mientras el sistema está en ejecución.

Es importante destacar la heterogeneidad del sistema de archivos virtuales */proc*. Los archivos residentes en el mismo son muy distintos entre sí, tanto en formato como en contenido. Esta característica provoca que la forma de uso y consulta de los mismos difiera mucho de un archivo a otro. Además, la descripción de cuál es la información alojada en cada archivo, así como la forma en que dicha información está organizada dentro del mismo, tan sólo se encuentra presente en los manuales especializados sobre

el núcleo de Linux. Esta circunstancia hace que el análisis de los datos de rendimiento presentes en */proc* sea únicamente realizable por usuarios cualificados y conocedores de la organización interna del mismo. En general es necesario conocer de antemano el significado de los diferentes campos de cada archivo en el directorio */proc* para interpretar correctamente los datos encontrados sobre el estado del núcleo.

Otro hecho que hace que la organización de los archivos dentro del directorio */proc* sea muy poco homogénea es que el propio directorio es modificable por el usuario. Esto significa que cualquier usuario con los conocimientos suficientes puede alterar el contenido de los archivos presentes en */proc*, añadir nuevos subdirectorios y ficheros al sistema de archivos virtuales, así como modificar la propia estructura del mismo. La heterogeneidad que se deriva de todos estos posibles cambios no facilita en absoluto el análisis de los datos de rendimiento presentes en */proc* de manera sistematizada.

Por otra parte, cabe destacar que la información almacenada en los archivos de */proc*, está presentada en ocasiones de modo que sea procesable por una aplicación informática, otras veces de modo que sea directamente observable por un usuario, y en último caso una mezcla de ambas. Esto no hace sino complicar aún más la tarea de análisis de los datos de rendimiento presentes en el sistema de archivos virtuales */proc* de forma homogénea.

Otra característica reseñable es que el acceso a algunos de los archivos en el directorio */proc* está limitado a los administradores, y por ello su acceso sólo está permitido al usuario root. Esto es debido a que dentro de este directorio se encuentran archivos que contienen información importante acerca de la configuración del propio núcleo. Como algunos de estos archivos son modificables por el usuario, no sería seguro para el sistema un acceso totalmente abierto al directorio */proc*, puesto que algunas modificaciones realizadas sobre ciertos archivos podrían dañar gravemente el equipo.
[RHM1 y CLX1]

Dentro del sistema de archivos virtuales */proc* hay una gran cantidad de archivos y subdirectorios. En concreto, es importante conocer que hay un subdirectorio para cada proceso que esté en ejecución en el equipo en el momento en que se accede a */proc* (recuérdese que por ser un sistema de archivos virtuales, su contenido se monta cada vez que se accede al mismo, pudiendo variar significativamente de un instante de tiempo a otro). Cada uno de estos subdirectorios tiene por nombre el identificador del proceso al que hace referencia, es decir, su PID. Cuando el proceso en cuestión finaliza su ejecución, el subdirectorio correspondiente desaparece de */proc*. Sin embargo, mientras que se está ejecutando el proceso, una gran cantidad de información específica a ese proceso está contenida en varios ficheros del directorio de procesos. Esta información hace referencia a aspectos como el uso que hace cada proceso de las CPUs del sistema, una lista de sus variables de entorno, el estado del proceso, o mapas de memoria para los diversos ejecutables y ficheros de librería asociados a dicho proceso.
Otros subdirectorios importantes en */proc* son los directorios de dispositivo. Cada elemento hardware del sistema, como un disco duro o un CD-ROM, tendrá su propio subdirectorio en el que están incluídas una recopilación de información relativa al mismo, así como estadísitcas de uso y rendimiento.
Un subdirectorio especial y diferente al resto de los presentes en */proc* es /proc/sys. Esto es debido a que, además de proporcionar multitud de información sobre

el sistema, permite realizar cambios en la configuración de un kernel en ejecución. Es precisamente esta circunstancia la que hace que sea un subdirectorio en cierto modo delicado, puesto que cambios poco prudentes en la configuración del núcleo pueden dar lugar a cierta inestabilidad en el sistema.

2. Métodos de estudio del rendimiento de un computador

El estudio y análisis de los datos de rendimiento de los equipos que operan bajo el sistema operativo Linux ha sido, tradicionalmente, un campo no lo suficientemente explorado, según se aprecia en la literatura existente. De hecho, se considera que las aplicaciones comerciales tienen en esta materia un filón por explotar, puesto que las aplicaciones de libre distribución disponibles al respecto no son todavía lo suficientemente potentes para un análisis pormenorizado.

En cualquier caso, vamos a detallar a continuación algunas de las principales herramientas con que habitualmente se realiza el estudio del rendimiento en los equipos con sistema operativo Linux:

a) Medidas precisas del tiempo.
b) gprof.
c) Oprofile.
d) PAPI.
e) Abyss.
f) Xlogmaster.

a) Medidas precisas del tiempo.

Uno de los primeros problemas con que nos encontramos a la hora de analizar el rendimiento de un equipo son las medidas del tiempo. La manera en que Unix ha realizado esta medida tradicionalmente ha sido mediante la instrucción gettimeofday( ), cuya ventaja es que es totalmente portable para cualquier versión de Linux. Sin embargo, la precisión de la medida es de tan sólo un *tick* de reloj, lo que equivale aproximadamente a 1 ms. en equipos con tecnología Alpha. Se puede apreciar claramente que esta medida no es lo suficientemente ajustada si observamos que equivale a 500000 ciclos de CPU en un equipo de 500 MHz. Otra clara desventaja de esta instrucción es que implica una llamada al sistema, lo que interfiere inevitablemente en las medidas que se estén realizando, tal como se ha indicado en situaciones parecidas en capítulos anteriores, puesto que las llamadas al sistema son demasiado lentas en relación a la mayoría de las mediciones sobre el rendimiento que se puedan hacer.

Por todo esto, sería necesario usar un mecanismo alternativo que nos proporcionara medidas del tiempo de mayor precisión. La mayoría de los equipos actuales disponen de un registro cuyo valor se incrementa en cada ciclo de CPU. La instrucción rpcc (*read processor cycle count*), que nos proporciona la arquitectura Alpha, accede a este registro, permitiéndonos medir los ciclos virtuales de la ejecución de un proceso. Además, mediante el uso de esta instrucción se minimiza muy

significativamente el problema del tiempo que introducían las llamadas al sistema que se vieron en el párrafo anterior.

b) gprof.

Tanto gprof como Oprofile, la herramienta que se verá en el siguiente apartado, están principalmente orientadas a analizar el rendimiento de los programas en ejecución, más que del equipo en sí. Ambos son mecanismos usados para obtener el perfil de ejecución de los programas (*profile*), y detectar a través de él en qué puntos de los mismos se están produciendo hipotéticos cuellos de botella, para así poder actuar sobre ellos mejorando el rendimiento. Sin embargo, una reorientación de estas herramientas para conseguir un análisis del rendimiento, tanto *hardware* como *software*, del equipo, es posible siempre que dispongamos de los programas apropiados sobre los que realizar el correspondiente análisis de perfil.

En cuanto a gprof, la manera en que trabaja es haciendo que sea el propio programa el que genere la información de su perfil, mediante una combinación de la instrucción gcc con gprof. Al compilar el programa mediante la instrucción gcc, se debe añadir al final de la misma la opción –pg, que se encargará de generar automáticamente la información de perfil del programa. Posteriormente, al llamar a la instrucción gprof, esta información será leída, y quedará almacenada en el fichero indicado en la propia instrucción, en tres formatos diferentes. El primero de ellos se conoce como *flat profile*, y contiene la información de cuánto tiempo invierte el programa en cada función del mismo, así como el número de veces que dicha función es invocada. Así, este formato del perfil del programa tan sólo sirve para indicarnos cuáles de las funciones del programa son las que están invirtiendo una mayor cantidad de tiempo en ejecutarse. El segundo formato, *call graph*, muestra, para cada función del programa, qué otra función fue la que la llamó, así como a qué otras funciones llama ella. Además, se muestra una estimación del tiempo que se ha invertido en cada una de estas subrutinas, lo que puede darnos una buena indicación para saber cuáles de ellas sería interesante eliminar si buscáramos mejorar el rendimiento del programa. El último formato en que se almacena la información del perfil, *annotated source*, tan sólo muestra cada línea del programa, añadiendo a su lado información sobre el número de veces que la misma ha sido ejecutada.

c) Oprofile.

La principal diferencia de Oprofile respecto a gprof, la herramienta estudiada en el apartado anterior, es que, mientras gprof ejecutaba el programa en su totalidad para crear un perfil global del mismo al final, Oprofile va interrumpiendo la ejecución mediante mecanismos externos (*hardware* y *kernel* habitualmente), para ir tomando muestras del estado del programa y su perfil en cada paso. A pesar de que gprof es un mecanismo más cómodo, cuando necesitamos un análisis detallado del perfil de un programa es conveniente usar Oprofile, pues es una herramienta más potente. Como ejemplo de sus posibilidades, basta ver que esta herramienta es capaz de dar el perfil de librerías dinámicas, e incluso puede usarse para analizar la ejecución del propio *kernel* de Linux. Además, no sólo es capaz de obtener información acerca del consumo de CPU, sino también de mostrar muchas otras estadísticas, como el número de fallos de caché, de TLB, o el número de predicciones de salto acertadas.

La mayoría de los procesadores actuales incluyen algunos registros hardware con información acerca del rendimiento, que se van incrementando a medida que ocurren determinados eventos en el equipo, específicos de cada contador. Son precisamente estos contadores los que va a usar Oprofile para indicar a la CPU qué eventos son los que quiere monitorizar, y en qué intervalos de tiempo. De esta manera, cuando un contador determinado alcance un valor que Oprofile habrá indicado como parámetro, la CPU interrumpirá la ejecución del programa para que la herramienta pueda almacenar la información sobre el perfil del programa que haya disponible en ese instante de la ejecución.

Precisamente por esta dependencia de Oprofile respecto al *hardware* del equipo sobre el que se esté ejecutando, las posibilidades de monitorización del rendimiento varían mucho de una computadora a otra, haciendo muy complicada su portabilidad y generalidad.

d) PAPI.

La aplicación Performance Application Programming Interface (PAPI), es sin duda una de las herramientas más potentes que existen para medir y consultar el rendimiento de un equipo que opere bajo el sistema operativo Linux, y según la literatura consultada, también una de las más usadas.

PAPI, al igual que el mecanismo Oprofile estudiado en el apartado anterior, basa su funcionamiento en la consulta de los almacenamientos hardware de información de rendimiento existentes en la gran mayoría de procesadores actuales. Estos contadores consisten en un conjunto de registros que se encargan de almacenar información sobre eventos, entendiendo los mismos como cada ocurrencia de una señal específica relacionada con alguna función determinada del procesador. El objetivo de PAPI es permitir a los diseñadores de aplicaciones observar, prácticamente en tiempo real, la relación existente entre el rendimiento y estructura del software analizado y los eventos del procesador almacenados en sus contadores hardware de rendimiento, lo que permite determinar la dependencia del rendimiento del software respecto a la arquitectura que lo soporta.

Esta relación que PAPI se encarga de mostrar tiene muy diversas posibilidades de aprovechamiento, como la optimización de los compiladores, el desarrollo de *benchmarks*, la creación de nuevas tecnologías y arquitecturas que atenúen el problema de los cuellos de botella, etc.

PAPI dispone de dos interfaces diferentes para acceder a los contadores hardware: una de alto nivel y otra de bajo nivel. La de alto nivel, más simple, tan sólo proporciona medidas y consultas básicas de los contadores mencionados, mientras que la de bajo nivel permite una programación avanzada y está orientada a usuarios que requieran una profundidad mayor en su análisis.

El interfaz de bajo nivel agrupa los eventos del procesador que se almacenan en los registros hardware en los llamados conjuntos de eventos (*event sets*), en base a similitudes funcionales. Esto permite una visión global de ciertos comportamientos del procesador, haciendo más fructuosa la tarea del análisis de los contadores. Además, los

conjuntos de eventos son totalmente programables por el usuario, de manera que las hipotéticas nuevas relaciones que se encuentren puedan ser plasmadas en la interfaz.

A pesar de que PAPI es relativamente portable entre equipos con diferentes características cuando usamos la interfaz de alto nivel, a la hora de un análisis en profundidad con la interfaz de bajo nivel, y con conjuntos de eventos programados por el propio usuario, esa compatibilidad entre distintas plataformas se ve claramente mermada.

e) Abyss.

El elemento de disco del sistema operativo Linux abyss, situado en */dev/abyss*, proporciona acceso a los registros de los contadores hardware de rendimiento presentes en los procesadores Pentium 4, siendo el programa en lenguaje C Abyss la aplicación diseñada para el uso del mismo.

Para usar esta aplicación basta indicar el programa que se desea monitorizar y los eventos que se quieren contar. Ella se encarga de la creación, mediante la instrucción fork( ), de procesos adicionales para la ejecución del programa, y tras la finalización de la misma realiza la escritura, en la salida estándar, de los datos recogidos en las lecturas de los contadores hardware de rendimiento que se hayan producido según lo indicado en los datos pasados como parámetros.

f) Xlogmaster.

Xlogmaster es un programa de GNU, y por lo tanto de libre difusión, escrito por Georg C. F. Greve, cuya función es la de monitorizar los datos de rendimiento y otros aspectos del sistema en que se esté ejecutando la aplicación. Está basado en la herramienta GTK+ (*GIMP Toolkit*), que es una plataforma con diversas utilidades para crear interfaces gráficas de usuario.

El objetivo de Xlogmaster es el de monitorizar la actividad del sistema y el estado del hardware del mismo, de manera que se puedan localizar y solucionar hipotéticos problemas de rendimiento. En este capítulo, en el que lo que se pretende es obtener los datos de rendimiento presentes en el sistema virtual de archivos */proc*, Xlogmaster puede ser de especial utilidad si tenemos en cuenta que sirve para monitorizar todos los archivos y dispositivos que permitan que su estado actual sea leído por ‘cat’, y los archivos alojados en */proc* lo son.

Xlogmaster puede actuar de acuerdo a tres modos de operar diferentes. El primero es el llamado *tail mode*, el segundo es el *cat mode* y el tercero es el *run mode*. Para todos los modos de operación, es posible definir un intervalo, que habrá que expresar en décimas de segundo, y que no será más que el tiempo transcurrido entre cada dos llamadas a las funciones de interrupción.

El primero de los modos de operación, *tail mode*, actúa leyendo al inicio del programa un fichero o *logfile* que se le haya pasado como parámetro, y posteriormente chequeando, con la frecuencia que le indique el intervalo introducido por el usuario, si han ocurrido o no cambios en dichos ficheros o *logfiles*.

En *cat mode*, el segundo modo de operación, lo que hace la aplicación es realizar, con la frecuencia que le indique el usuario mediante el intervalo pasado como parámetro, una llamada *cat* sobre el directorio o archivo que se le pase al comenzar el programa. Este será el modo de operación que más convenga al proyecto en realización, pues tal como se dijo unas líneas más ariba, lo que se pretende es leer, con la periodicidad deseada, el sistema virtual de archivos */proc*.

Xlogmaster dispone aún de un tercer modo de operación, *run mode*, que servirá para ejecutar un programa que se le pase como parámetro a la aplicación, y posteriormente al inicio del mismo, ir recogiendo, a cada intervalo pasado por el usuario como parámetro, la salida generada y almacenándola en un fichero.

Además de los modos de operación ya citados, Xlogmaster dispone de dos clases de filtros que serán útiles a la hora de presentar la información leída por el programa. La primera de estas dos clases de filtros, denominada en el programa *Class0*, actúa a la hora de monitorizar los resultados de las lecturas efectuadas, aplicando diferentes opciones sobre las líneas de resultado obtenidas. Esta clase dispone de tres filtros diferentes, de manera que para cada línea sólamente uno de ellos puede estar activo al mismo tiempo. En caso de haber varios seleccionados, sólo permanecerá el de mayor prioridad. Los tres filtros, ordenados de mayor a menor prioridad, son: *hide, raise* y *lower*. Cuando el primero de ellos está activado, la línea que lo haya seleccionado no se mostrará por pantalla en la monitorización. En el caso de *raise*, la línea que lo haya activado aparecerá resaltada en la monitorización, mientras que con *lower* aparecerá atenuada.

La segunda clase de filtros de Xlogmaster, denominada como *Class1*, actúa a más bajo nivel, cuando los datos están siendo leídos. A diferencia de los filtros de la clase anterior, estos sí que pueden estar activos simultáneamente para una misma línea de resultados, y son por lo tanto no excluyentes. Además, estos filtros llevan asociada una señal sonora configurable para facilitar el control de los avisos que genere el programa para los eventos especificados por el usuario. Los tipos de filtro de *Class1* son los siguientes:

* ***alert:*** la funcionalidad de este filtro es marcar, con un color que lo resalte, el botón correspondiente al evento que el usuario desee controlar y que será pasado como parámetro. Puesto que está pensado para que notifique eventos que son relevantes tan sólo durante un período de tiempo inmediatamente posterior a que ocurran, este color de alerta que resaltará el botón se va atenuando poco a poco, indicando el tiempo que haya pasado desde que ocurrió el evento. Este grado en que el color de alerta se irá difuminando es configurable por el usuario, que podrá cambiarlo en función de por cuánto tiempo considere relevantes los eventos ocurridos en el fichero correspondiente.

* ***execute:*** este filtro permite al usuario hacer que se ejecute un determinado programa o *script* en el momento en que aparezca como salida una determinada línea, que el usuario introducirá como parámetro. Este programa que se ejecutará recibe su información por la línea de comandos o por las variables de contexto. Xlogmaster proporciona una serie de estas variables para poder pasar información al programa, y son las siguientes:

`%F'
`XLM_FILENAME'
Ruta absoluta donde se encuentra el fichero o componente de entrada.

`%H'
`XLM_HELP'
Texto de ayuda para la entrada.

`%L'
`XLM_LINE'
Línea de salida que provoca que se dispare el filtro *execute*.

`%M'
`XLM_MODE'
Modo de entrada al programa.

`%N'
`XLM_NAME'
Nombre del archivo o *logfile* que provoca que se dispare el filtro *execute*.

Es importante destacar que el filtro *execute* sólo permite la ejecución y procesamiento de un comando, de manera que si se quieren crear configuraciones más complejas, como por ejemplo *piping*, será necesario que el propio usuario escriba un pequeño programa que sirva a este fin.

**** notice:*** este filtro está pensado para aquellos eventos que, a diferencia de los que resaltaba el filtro *alert*, son de gran importancia independientemente de cuándo hayan sido producidos. Por eso, las alertas que genera este filtro son permanentes, de manera que el usuario pueda consultarlas cuando lo necesite, y no desaparecen con el tiempo como las de *alert*. Cuando uno de estos eventos ocurren, una ventana de aviso aparece en la posición en que se encuentre el ratón en ese momento. Esta ventana contiene una lista con información acerca de la fecha y la hora en que se produjo el evento desencadenante, así como el nombre de la entrada que lo provocó. Si el usuario pulsa sobre una línea de esta ventana, se verá la línea desencadenante en el texto de salida que haya tras la ventana emergente. Para que la ventana sea borrada y desaparezca, basta con pulsar sobre el botón "*dismiss*". Sólo en este momento la información sobre el evento desaparecerá, de manera que permanecerá almacenada a menos que el usuario indique lo contrario.

**** uniconify:*** este filtro, muy útil cuando no se dispone de suficiente espacio en el equipo, hace que Xlogmaster permanezca minimizado en un icono, de manera que sólo se mostrará completamente cuando ocurra un determinado evento, que el usuario pasará como parámetro.

3 Problemática de la obtención de datos

Antes de comenzar a estudiar la manera en que vamos a obtener los datos de rendimiento del equipo, conviene observar ciertas dificultades que se pueden presentar durante el proceso. Los problemas que se explicaban en el capítulo anterior acerca del tiempo y el espacio necesarios para capturar datos de rendimiento de un sistema

también son aplicables en el entorno Linux. Además de aquellos, existen otros problemas específicos:

a) Falta de generalidad:

Como se dijo en apartados anteriores, el núcleo del sistema operativo Linux puede ser modificado y recompilado por cualquier administrador que conozca el procedimiento para hacerlo y desee seguirlo. En concreto, el sistema de archivos virtuales */proc*, que es el objeto de nuestro estudio, puede ser modificado por el usuario. Esto significa que no puede garantizarse la generalidad del contenido del sistema de archivos */proc*, ni su estructura de subdirectorios, ni qué archivos están presentes en él, ni la propia clasificación de estos archivos. Aunque la recompilación del núcleo no es algo excesivamente frecuente, debido a la dificultad y riesgo del proceso, hecho que disminuye el impacto de este problema, si se plantea la posibilidad de desarrollar una aplicación que obtenga datos de los archivos en el directorio */proc*, es necesario contar con este problema y pensar que la aplicación debe usar código específico según la versión del núcleo en la que se desee trabajar.

b) Heterogeneidad de los tipos de archivos en */proc*:

Una vez considerada la falta de generalidad del sistema de archivos virtuales */proc*, ya que puede variar de un computador a otro por haber sido modificado por un usuario, hay que atender al contenido del mismo. Dentro del directorio */proc* no sólo se encuentran almacenados archivos con información acerca del rendimiento del computador, sino otra gran cantidad de ficheros diferentes con datos totalmente ajenos al rendimiento. De este modo, podemos encontrar información relativa a los componentes hardware instalados en el equipo, así como a la propia configuración del núcleo del sistema operativo. Esta dificultad se ve agravada por la existencia de archivos en los que se encuentran mezclados datos referentes al rendimiento del ordenador junto con otros que no lo son.

De esta manera, es necesario un conocimiento en profundidad del directorio */proc* si se quiere poder acceder a la información deseada en cada caso. Tan sólo en manuales específicos sobre el sistema operativo y su núcleo se puede encontrar información detallada sobre los principales archivos de */proc*. Esto supone que la tarea de encontrar y analizar la información de los datos de rendimiento almacenada en el equipo sea lenta y compleja.

Estas dificultades se suman a las expuestas en el punto anterior, puesto que los manuales en los que encontrar información acerca de cómo está organizado */proc* presuponen una cierta generalidad en el núcleo del sistema operativo. Si algún usuario ha modificado el sistema de archivos virtuales */proc*, sólo él será quien conozca su nueva estructura organizativa.

c) Heterogeneidad de los archivos con datos de rendimiento:

Una vez centrados en los ficheros del sistema de archivos virtuales */proc* que contienen información referente al rendimiento del equipo, es decir, datos de rendimiento, se observa que las dificulatades para acceder a los mismos no acaban aquí. Como se dijo en el punto anterior, en */proc* se alojan archivos que contienen al mismo tiempo información de rendimiento y otra ajena al mismo. En este subgrupo resulta obvio que si se quiere acceder a los datos de rendimiento propiamente dichos será

necesario indicar a la aplicación el punto exacto en que se encuentran los mismos dentro del archivo. Pero incluso en aquellos archivos que tan sólo contienen datos de rendimiento, el acceso no es trivial. Cada uno de estos archivos posee su propio formato y semántica, lo que implica la necesidad de código específico diferente para cada uno de ellos a la hora de recoger los datos de rendimiento.

Este problema se ve aumentado una vez más por su combinación con los anteriores, ya que ese estudio del directorio */proc*, su estructura, y sus archivos, deberá realizarse individualmente para cada versión del núcleo de Linux en el que se desee que funcione la aplicación.

4 Decisión sobre el método de acceso a utilizar

A lo largo del capítulo se ha visto que los datos de rendimiento que se pretenden obtener y analizar en este estudio están almacenados en el sistema de archivos virtuales */proc*. En base a dicha información, así como a las diferentes caracterísicas de los principales métodos de estudio del rendimiento estudiados en el apartado segundo de este capítulo, se va a decidir ahora sobre la más conveniente forma de realizar este análisis.

Lo que se pretende analizar en este estudio son los datos de rendimiento de un computador. Este motivo explica por sí solo que se haya optado por descartar como métodos de estudio del rendimiento a las herramientas que no accedían a los datos de rendimiento propiamente dichos. Como se dijo en el apartado segundo, cuando se analizaron los métodos de estudio del rendimiento, algunos de ellos accedían a los registros hardware de los procesadores. Estos registros contienen, en efecto, información relativa al rendimiento, pero no son los datos de rendimiento que se pretende analizar y que se encuentran en */proc*. Se vio asimismo que otros métodos estaban orientados fundamentalmente a estudiar el rendimiento de un programa o aplicación concreta. Este hecho hacía que el uso de esas herramientas para el análisis del rendimiento del computador de manera global fuera complicado. Para ello había que combinar el uso de las mismas con el de otros programas específicos que prepararan al computador para su análisis, y el estudio total resultaba así más lento que por otros caminos.

Debido a que la funcionalidad que se busca en la aplicación para el estudio de los datos de rendimiento es bastante concreta, y puesto que ninguna de las herramientas estudiadas en el capítulo la satisface plenamente, se opta finalmente por el desarrollo de una aplicación propia. Dicha aplicación se encargará de acceder a los datos de rendimiento presentes en el sistema de archivos virtuales */proc*. Posteriormente, la información de estos archivos será almacenada en otros ficheros con el fin de facilitar el análisis del rendimiento del equipo.

5 Implementación

La aplicación que se desea implementar debe tener la posibilidad de realizar las siguientes funciones:

· Listar el contenido de un directorio, es decir, todos los subdirectorios y archivos que haya dentro del mismo. En nuestro caso, el directorio a listar será */proc*.

· Permitir al usuario seleccionar, entre los archivos de un directorio, a varios de ellos. En nuestro caso, estos archivos serán aquellos que estén en */proc* y que contengan los datos de rendimiento.

· Permitir al usuario cancelar la selección de archivos realizada, poniendo la lista de archivos con datos de rendimiento otra vez a cero.

· Permitir al usuario guardar en un fichero la lista de los archivos con datos de rendimiento que esté seleccionada en ese momento. De esta manera, en un futuro se podrá recuperar la lista de los contadores con los que se estaba trabajando en un momento dado.

· Permitir al usuario recuperar la lista de los archivos con datos de rendimiento previamente guardada.

· Permitir al usuario elegir un fichero de destino en el que se guardarán las medidas realizadas sobre los archivos con datos de rendimiento.

· Permitir al usuario seleccionar el intervalo de tiempo deseado en base al cual se realizarán las mediciones.

· Iniciar la toma de mediciones de los archivos con datos de rendimiento seleccionados.

· Detener la toma de mediciones de los archivos con datos de rendimiento seleccionados.

La funcionalidad de la aplicación será la de realizar medidas del contenido de los archivos de */proc* que el usuario haya indicado, en los intervalos de tiempo que el usuario haya seleccionado, guardando dichas medidas en formato ARFF en el fichero de destino que se haya indicado.

En una primera aproximación a la implementación de esta aplicación, se encontró el siguiente problema. El sistema de archivos virtuales */proc* contiene un subdirectorio especial denominado self. Este subdirectorio contiene información acerca del proceso que está llamando al propio */proc*. De esta manera se puede tratar al proceso que invoca al directorio */proc* del mismo modo que al resto de procesos, pudiendo consultar los datos almacenados bajo su subdirectorio. El problema al desarrollar nuestro programa surgió cuando se recorre el directorio */proc* y se listan todos los archivos presentes en él. Como el proceso invocante de la función que realiza ese recorrido es la propia aplicación, el subdirectorio self de */proc* contiene la información del proceso correspondiente a la aplicación. Y como la aplicación está en ejecución mientras se lista el contenido del resto del directorio */proc*, el resultado es que el contenido a listar es infinito. De esta manera, se entra en un bucle sin fin en el que las rutas de los subdirectorios de */proc* que se están recorriendo con el fin de listar los archivos que contienen, son indefinidamente largas. En estas rutas de subdirectorios se observa cómo la entrada self se repite periódicamente en ellas, y puesto que self representa un enlace simbólico al proceso invocante del montaje de */proc*, el resultado es que se vuelve, a efectos de la estructura del directorio, al comienzo del mismo. Por este motivo, la aplicación queda bloqueada al realizar la llamada a la función que lista el contenido del directorio */proc*, inutilizando así el resto de funcionalidades de la misma.

Además de este problema, surgido por el subdirectorio self del sistema de archivos virtuales */proc*, se observó al realizar el recorrido del resto del directorio que había muchos otros archivos y subdirectorios que obstaculizaban el desarrollo de la aplicación. El tamaño de los mismos era excesivamente grande para las prestaciones que se deseaban obtener de la aplicación, mientras que su relevancia respecto a los datos

de rendimiento del computador era nula. Recuérdese que, como se dijo en apartados anteriores, no todos los subdirectorios y archivos alojados en */proc* contienen datos de rendimiento, puesto que es un directorio muy heterogéneo con información muy diversa acerca del estado general del equipo.

Ante esta perspectiva, en la que tan sólo algunas de las entradas del directorio */proc* contenían datos de rendimiento, mientras que el resto no hacían sino ralentizar, y en ocasiones, como en el caso del subdirectorio self, obstaculizar gravemente la implementación de la aplicación, se optó por la siguiente solución. Se permite al usuario la posibilidad de elegir cuáles de las entradas de primer nivel del directorio */proc* son las que desea que sean recorridas para listar todos sus archivos.

De este modo, la filosofía del análisis de los datos de rendimiento cambia diametralmente de enfoque. Al comienzo de nuestro estudio, la idea fue comenzar explorando el contenido del sistema de archivos virtuales */proc* al completo. Ahora, y ante la evidencia de que tan sólo una pequeña parte del directorio es relevante para nuestro proyecto, se tratará de estudiar, por medio de manuales del sistema operativo, cuáles de los subdirectorios son los que contienen datos de rendimiento. De esta manera se irá ampliando la lista de archivos de */proc* a analizar, consiguiendo paso a paso el objetivo final del estudio completo de los datos de rendimiento del equipo.

En el desarrollo posterior de la aplicación se encontró otro problema de importancia capital a la hora de almacenar los datos de rendimiento. Como se explicó en apartados anteriores, los archivos del directorio */proc* no son homogéneos en cuanto a la forma en que presentan su información. Algunos de ellos contienen datos de rendimiento junto con otra información totalmente ajena a ellos. E incluso aquellos archivos que sólo contienen datos de rendimiento, poseen una estructura para la información que es particular de cada archivo. Por todo esto, se considera conveniente organizar de alguna manera los tipos de archivos existentes en */proc*, en base a la forma que tienen de presentar los datos de rendimiento presentes en ellos. Para ello, se crea un nuevo tipo de datos de tipo “struct”, que representará al dato de rendimiento, con atributos sobre el archivo que lo contiene y la forma en que el dato se sitúa dentro de dicho archivo. Su estructura es la siguiente:

struct dato;
char* ruta; char* nombre; int tipo; int linea; int inicio; int fin; char* separador; int cual;

//constructoras
dato nuevoDato1(char* ruta, char* nombre, int tipo, int linea, int inicio, int fin);
dato nuevoDato2(char* ruta, char* nombre, int tipo, int linea, char* separador,int cual);
//accesoras
char* getRuta(dato d);
char* getNombre(dato d);
int getTipo(dato d);
int getLinea(dato d);

```
int getInicio(dato d);
int getFin(dato d);
char* getSeparador(dato d);
int getCual(dato d);
```

· En el atributo “ruta” se almacena la ruta completa correspondiente al archivo en el que está almacenado el dato de rendimiento. Esta ruta, en nuestro caso, comenzará siempre por la raíz “/proc/”.

· El atributo “nombre” contiene el nombre del dato de rendimiento, que deberá ser indicativo del elemento funcional al que hace referencia.

· El atributo “tipo” indicará la clase de dato de rendimiento ante el que estamos. En la implementación proporcionada, este tipo podrá ser uno o dos.

· El atributo “linea” indica la línea, dentro del archivo en que se almacena el dato de rendimiento, en la que este se encuentra.

· Los atributos “inicio” y “fin”, propios tan solo del tipo uno, indican entre qué dos posiciones de la línea indicada por el atributo anterior se sitúa exactamente el dato de rendimiento.

· Los atributos “separador” y “cual” son propios del tipo dos de dato de rendimiento, y sirven para localizar exactamente al dato dentro de la línea marcada por el atributo “linea”. “separador” indica cuál es el carácter que separa un token de otro, y “cual”, cuántos tokens hay que saltar hasta encontrar el dato de rendimiento.

· Las constructoras de ambos tipos se encargan de rellenar los atributos que corresponda a cada tipo con la información pasada como parámetros.

· Las accesoras se encargan simplemente de devolver los valores de la estructura de datos que se le pidan en cada caso.

· También se añade un atributo “almacenDatos”, que es un array de datos de rendimiento en el que se almacenarán todos los datos conocidos hasta el momento. Además, se implementa un método “anadeDato” para introducir un nuevo dato de rendimiento dentro de dicha estructura.

· Se incluye un método “inicializar”, que se encarga de definir todos los datos de rendimiento conocidos hasta el momento, e introducirlos en “almacenDatos”. Este método será invocado cada vez que se ejecute la aplicación.

Con la introducción de esta estructura de datos, el estudio de los datos de rendimiento dentro del sistema operativo Linux queda guiado de la siguiente manera. Cada vez que se encuentre un nuevo dato de rendimiento, si este se puede localizar dentro de su archivo de alguna manera que ya esté descrita en alguno de los tipos de datos implementados, bastará con incluír una definición del nuevo dato (con la constructora que se ajuste a su tipo) dentro del método “inicializar”, tras la cual habrá que llamar a “anadeDato”. Mientras que si el nuevo dato de rendimiento encontrado implica además una nueva forma de localizar el dato dentro de su archivo, aún no reflejada por los tipos de datos existentes, habrá que añadir una nueva constructora y su escritora correspondiente, similares a las proporcionadas, y sólo en caso de que fuera necesario, nuevos atributos para la estrucura, con sus correspondientes accesores.

Los pasos que sigue el funcionamiento de la aplicación definitiva para registrar mediciones sobre los datos de rendimiento del computador, son los siguientes:

· El usuario introduce en un campo de texto el nombre de la entrada del directorio */proc* cuyo contenido desea que sea listado. El nombre de la entrada introducida puede corresponder tanto a un archivo como a un subdirectorio. En el primer caso, el propio archivo será el único que aparezca en el listado correspondiente, mientras que en el segundo caso aparecerá la lista de todos los archivos, con sus rutas completas, presentes en dicho subdirectorio.

· Una vez introducido el nombre de la entrada, esta queda registrada en la aplicación al pulsar sobre el botón "añadir entrada". El proceso descrito en estos dos primeros pasos puede repetirse para el número de entradas que se quieran.

· Ahora que tenemos todas las entradas del directorio */proc* que se quieren listar, se puede pulsar en el botón "inicio". Esto provocará que el comboBox correspondiente se rellene con todos los archivos, con sus rutas completas, que haya en las entradas de */proc* que se indicaron en los pasos anteriores. El comboBox también contendrá el nombre de todos los datos de rendimiento que se hayan registrado hasta el momento, y que estarán almacenados en "almacenDatos".

· Con el comboBox se tiene la posibilidad de seleccionar cualquiera de los archivos y contadores presentes en él. Una vez seleccionado el que se quiere medir, pulsando el botón "añadir", dicho archivo o contador se añade a una estructura interna que contiene a todos los ficheros de los que posteriormente se harán las mediciones. Además, el nombre del archivo o contador se escribe sobre un campo de texto en el que se pueden ver todos los archivos seleccionados hasta el momento.

· Si se pulsa el botón "eliminar", la estructura interna que contiene a todos los archivos y contadores que se quieren medir se vacía. También el campo de texto con el nombre de los archivos queda liberado al pulsar dicho botón.

· Opcionalmente, el usuario tiene la posiblidad de introducir, en un campo de texto, el nombre de un archivo en el que se guardarán, o del que se recuperarán, la lista de los archivos y contadores de los que se pretende realizar la medición.

· Si el botón que se pulsa tras introducir el texto es el de "guardar", la lista de archivos y contadores a medir quedará almacenada para un uso futuro.

· Si el botón que se pulsa tras introducir el texto es el de "cargar", se cargarán en la lista de archivos y contadores a medir los que hubiera guardados en el fichero indicado.

· Para seleccionar el intervalo de tiempo en base al que se realizarán las mediciones de los archivosy contadores, el usuario dispone de dos elementos gráficos. En uno debe seleccionar el número de unidades del intervalo, y en el otro la unidad de medida del mismo. Esta última elección se realizará entre segundos y minutos.

· Una vez decidida la lista de contadores y archivos del directorio */proc* sobre los que se realizarán las mediciones, y el intervalo de tiempo que separará una medición de la siguiente, tan sólo queda que el usuario introduzca en un campo de texto el archivo destino sobre el que se almacenarán las mediciones. Al pulsar el botón "iniciar medición", la aplicación comienza a escribir sobre el fichero de destino, y al pulsar el botón "parar medición" dicho archivo se cierra y la aplicación finaliza su ejecución.

La clase de la aplicación en la que están las funcionalidades de la interfaz gráfica es callbacks.c. Los atributos y métodos de la misma son los siguientes:

callbacks.c
char* contadores[BUFSIZ]; char* contadores2[BUFSIZ]; char* datos[BUFSIZ]; char* datos2[BUFSIZ]; char* datos3[BUFSIZ]; int fd_destino; int mide; int dormir;

void on_anadirEntrada_clicked (GtkButton* button, gpointer user_data);
void on_inicio_clicked (GtkButton* button, gpointer user_data);
void recorrer_arbol (char* dir, GtkWidget* cp, int a);
void on_anadir_clicked (GtkButton* button, gpointer user_data);
void on_eliminar_clicked (GtkButton* button, gpointer user_data);
void on_guardar_clicked (GtkButton* button, gpointer user_data);
void on_cargar_clicked (GtkButton* button, gpointer user_data);
void on_iniciar_clicked (GtkButton* button, gpointer user_data);
void on_parar_clicked (GtkButton* button, gpointer user_data);
void iniciarMedidas (int fd_destino, gchar* fichero);
void iniciaARFF (int fd_destino, gchar* fichero);
void contadorAnadido (int fd_destino, char contador[]);
void ponHora (int fd_destino);
void introduceDatos (int fd_destino);
void escribeTodo(char* contador,int fd_destino);
void escribe1(char* contador,int fd_destino,int linea,int inicio,int fin);
void escribe2(char* contador,int fd_destino,int linea,char* separador,int cual);
void acabaARFF (int fd_destino);
void cuenta ();

· En el atributo “contadores” se van a almacenar todas las entradas del directorio */proc* que se hayan añadido a la lista de subdirectorios a listar. Esta función la implementará el método “on_anadirEntrada_clicked”. El atributo “contadores2” no es más que un atributo auxiliar para realizar copias de seguridad antes de manipular los atributos originales.

· El atributo “datos” se encarga de almacenar los archivos de */proc* y los contadores que se hayan seleccionado para realizar mediciones de su contenido. El atributo se irá rellenando cada vez que se llame al método “on_anadir_clicked”, y se vaciará al llamar al método “on_eliminar_clicked”. Los atributos “datos2” y “datos3” son auxiliares, para facilitar las copias de seguridad y no perder en ningún momento la información original.

· El atributo “fd_destino” es el identificador del fichero destino en el que se escribirán las mediciones realizadas, y que será especificado por el usuario en un campo de texto de la interfaz gráfica.

· El atributo “mide” se encarga de indicar durante qué espacio de tiempo se deben realizar las mediciones. Se igualará a uno cuando se pulse en el botón “iniciar”, y volverá a cero al pulsar sobre “parar”.

· El atributo “dormir” almacena el intervalo de tiempo que debe pasar entre cada dos tomas de mediciones. Será calculado a partir de la elección que el usuario haga del número de unidades y la unidad de tiempo.

· El método “on_anadirEntrada_clicked” añade a la lista de entradas del directorio */proc* cuyo contenido se debe listar un nuevo elemento. La forma en que realiza esto es rellenando con una nueva entrada el atributo “contadores”.

· El método “on_inicio_clicked” llama al método “recorrer_arbol”, que se encargará de rellenar el comBox de los archivos del directorio. También llama a “inicializar”, tras lo que introduce en el comboBox el nombre de todos los datos de rendimiento conocidos hasta el momento, que se enontrarán almacenados en el atributo “almacenDatos”. Además, rellena el comBox de las unidades de tiempo con segundos y minutos.

· El método “recorrer_arbol” se encarga de rellenar el comboBox con los archivos del directorio */proc* cuyos subdirectorios hayan sido seleccionados. Para ello, va recorriendo todo el directorio */proc*, comparando las entradas que va leyendo con las que están almacenadas en “contadores”. En el caso de que coincidan , recorrerá el resto del subdirectorio, y cuando alcance las hojas –archivos– del mismo, las introducirá con su ruta completa en el comBox correspondiente.

· El método “on_anadir_clicked” añade el datos de rendimiento o archivo que haya seleccionado en el comBox anteriormente descrito a la lista de archivos a medir por la aplicación. Rellena el campo de texto de la interfaz en el que se muestra dicha lista, y también el atributo “datos” donde se almacena la misma.

· El método “on_eliminar_clicked” elimina, tanto del campo de texto de la interfaz como de “datos”, todos los archivos y datos de rendimiento que hubiera seleccionados para su medición.

· El método “on_guardar_clicked” se encarga de escribir, sobre un fichero que el usuario ha introducido por medio de la interfaz, la lista de los archivos y datos de rendimiento actualmente seleccionados para realizar la medición.

· El método “on_cargar_clicked” recupera la lista de datos de rendimiento y archivos a medir del fichero que el usuario pase por medio de la interfaz. Dicha lista se muestra en el campo de texto habilitado para ello, y también se carga en el atributo “datos”, definido para tal efecto.

· El método “on_iniciar_clicked” se encarga de crear el fichero de destino de las medidas que el usuario haya escrito a través de la interfaz. También calcula el valor del atributo “dormir” a través de los valores de unidades de tiempo que se hayan seleccionado. Tras ello, llama al método “iniciarMedidas”.

· El método “iniciarMedidas” llama inicialmente a “iniciaARFF”. Posteriormente, para cada archivo contenido en “datos”, invoca a “contadorAnadido”. Después llama al método “acabaARFF”, y por último llama, mientras no se pulse el botón “parar”, al método “cuenta”.

· Los métdos “iniciaARFF”, “contadorAnadido” y “acabaARFF” se encargan de rellenar la cabecera del archivo destino en el que se escribirán las mediciones. Esta cabecera, y el resto del archivo, está escrita en base al Attribute-Relation File Format (ARFF), que es un formato de archivo muy usado en análisis de datos como data mining.

· El método “cuenta” es invocado por la aplicación (en el método “iniciarMedidas”, como se comentó más arriba), en los intervalos de tiempo indicados por el usuario, comenzando cuando se pulsa el botón “iniciar” y finalizando al pulsar “parar”. Este método llama, cada vez que es invocado, a “ponHora” y “introduceDatos”.

· El método “ponHora” calcula la hora exacta actual a través de la función de C time(). Posteriormente introduce dicha hora en el archivo destino de las mediciones respetando el formato ARFF.

· El método “introduceDatos” se encarga de analizar cada elemento presente en la lista “datos”. Para cada uno de ellos, recorre “almacenDatos”, y si el elemento coincide con el nombre de alguno de los datos de rendimiento almacenados en “almacenDatos”, se llama a la escritora correspondiente al tipo del dato de rendimiento (en la implementación proporcionada, “escribe1” ó “escribe2”). En el caso de que el elemento de “datos” que se está analizando no coincida con el nombre de ninguna entrada de “almacenDatos”, estaremos ante una ruta de un archivo */proc*, y no ante un dato de rendimiento conocido; en este caso, se llama a “escribrTodo”.

· El método “escribeTodo” introduce en el archivo indicado por “fd_destino”, siempre en base al formato ARFF, el contenido completo del fichero cuya ruta se pasa en el parámetro “contador”.

· El método “escribe1” corresponde al tipo de dato de rendimiento uno. Se encarga de acceder a la línea inidicada por “linea” del archivo indicado por “contador”, y una vez situado en ella, escribe sobre el archivo de destino que le indica “fd_destino”, y según el formato ARFF, los caracteres que haya entre “inicio” y “fin” en dicha línea.

· El método “escribe2” corresponde al tipo de dato de rendimiento dos. Accede a la línea inicada por “linea” del archivo “contador”, y una vez en ella, escribe sobre el fichero “fd_destino”, respetando el formato ARFF, el token situado en la posición “cual” según el carácter separador “separador”.

3. Aproximación a una implementación alternativa de la organización de datos de rendimiento en los sistemas operativos Windows y Linux

0. Introducción

En los capítulos anteriores se ha explicado el punto de vista de los autores en cuanto a cómo debiera ser una taxonomía de los datos de rendimiento. Las ideas esenciales son:
· Taxonomía fácilmente extensible.
· Reglas precisas para la asignación de los datos de rendimiento en una u otra clase.
· Estructura clara y lo más homogénea, para facilitar la comprensión del modelo y el trabajo con los datos de rendimiento.

Igualmente, se ha explicado la organización de los datos de rendimiento en los sistemas operativos Windows y Linux, y se ha puesto de manifiesto que no satisfacen las características señaladas, además de poseer ciertas características, en opinión de los autores, poco deseables:

Características poco adecuadas de la clasificación en el sistema operativo Windows:
· La clasificación da la impresión de no haber sido planificada *a priori*
· No se encuentran reglas claras en la documentación, sino descripciones poco precisas de los motivos para asignar un dato de rendimiento a un grupo u otro.
· Se admiten dos definiciones distintas para objeto de rendimiento, lo que resta homogeneidad a la estructura.
· Hay faltas de paralelismo entre localización y clasificación de datos de rendimiento. En primer lugar, tras localizar un objeto de rendimiento, a veces se debe localizar una instancia del objeto y a veces no. En segundo lugar, la instancia de objeto se localiza de modo distinto según el objeto de rendimiento particular. A pesar de que Microsoft ofrece una solución para esta segunda falta de paralelismo, no aplica esta solución por defecto, sino que requiere que un usuario capacitado la establezca.
· Las definiciones de los niveles de organización podrían ser más completas y sencillas. Aunque se reconoce un intento por seguir la naturalidad de la organización, no se aprecia suficiente esfuerzo en la modelización.

Características poco adecuadas de la clasificación en el sistema operativo Linux:
· La clasificación no es particular de los datos de rendimiento, sino que se comparte con datos de configuración del ordenador e incluso con información como el fabricante y modelo de componentes específicos.
· No se ha encontrado documentación explícita sobre la modelización de la organización de los datos de rendimiento. Esto hace pensar que ésta fue, como mínimo, apenas planificada.
· No existe una división clara de la organización en niveles, lo que dificulta la comprensión del modelo seguido para la clasificación de los datos de rendimiento.
· La organización, aún no considerando los datos que no sean de rendimiento, carece de homogeneidad.

· Posee cierta dificultad acceder a un dato de rendimiento particular.
· El acceso a los datos de rendimiento no está diseñado para ser usable tanto por usuarios humanos como por aplicaciones automatizadas, y ni siquiera por parte de uno de los dos.
· La información sobre el significado de cada dato en la organización se encuentra en manuales de usuario diseñados para su uso por parte de operadores humanos, y no para la ayuda al proceso automático.
· Para contribuir a la falta de información sobre los datos en la organización, y a la falta de homogeneidad, el usuario puede modificar tanto qué datos aparecen en el dominio de la clasificación como la propia estructura de ésta.

Desde el punto de vista de los autores, no resulta ocioso realizar una reflexión sobre cómo podría facilitarse el acceso a los datos de rendimiento en ambos sistemas operativos. Además, se tienen presentes las siguientes ideas:
· Es importante mantener la compatibilidad con las clasificaciones actuales, para que sean necesarios los menos cambios posibles en los sistemas operativos, y de este modo, que los fabricantes y desarrolladores estén más dispuestos a realizarlos.
· Es importante mejorar la organización de los datos de rendimiento para facilitar su utilización a los usuarios y desarrolladores de software.

Estas ideas se comprobará que están presentes en la propuesta de desarrollada en este capítulo.

1. Características destacables de la organización en Windows

A continuación se desarrollan más las características poco adecuadas de la organización de los datos de rendimiento en el sistema operativo Windows. Estas características fueron introducidas en el capítulo 1 y recordadas en la introducción del capítulo 3.

a) Falta de planificación a priori

En la documentación de Microsoft no aparecen una definición precisa de “elemento funcional”, un concepto esencial para la planificación de la taxonomía. En realidad, dicho concepto parece haber sido utilizado como nombre colectivo para los tipos de elementos representados por los objetos de rendimiento.

Las faltas en cuanto a definición de conceptos continúan con los propios objetos de rendimiento, otro concepto importante. En particular los objetos de rendimiento a veces deben entenderse como elementos funcionales y a veces como tipos de elementos funcionales, según si es un objeto de rendimiento simple o múltiple. La falta de una única definición clara de objeto de rendimiento también influye sobre el concepto instancia de objeto de rendimiento, que sólo tiene sentido en objetos de rendimiento múltiples.

Esta doble definición dificulta establecer niveles en la organización, lo que dificulta la comprensión del modelo. Aunque se puede apreciar un intento de buscar naturalidad en la organización de los contadores de rendimiento, lo cierto es que también es aparente una falta de tiempo y esfuerzo invertido en la modelización. Esa falta podría haber mejorado la estructura utilizada, simplificándola y ayudando a cualquier usuario, profesional o no, interesado en los datos de rendimiento.

La propia falta de definición de elemento funcional permite la adición de nuevos objetos de rendimiento sin una adecuada justificación, debido precisamente a la lasitud demostrada en la definición de conceptos esenciales para la taxonomía de los datos de rendimiento.

b) Reglas oscuras

En la documentación ofrecida por Microsoft tampoco aparecen satisfactoriamente regladas las relaciones entre un contador de rendimiento y un objeto de rendimiento. En lugar de eso se da una descripción difusa de los motivos que llevan a asociar un contador de rendimiento determinado con un objeto de rendimiento, asegurando que esa asociación se realiza en base a ciertas relaciones, que tampoco son definidas.

En resumen, resulta oscura y poco clara la organización de los contadores de rendimiento utilizada por Microsoft.

c) Falta de homogeneidad en la estructura

Una estructura regular es más fácil de comprender que una irregular, ya que contiene menos casos particulares. A diferencia de la taxonomía propuesta en la sección 1 del capítulo 1, la clasificación utilizada por Microsoft es irregular. Las causas para la falta de homogeneidad son varias:

Un concepto, dos significados

Objeto de rendimiento posee dos significados según el objeto del que se trate. Si realmente era necesario dar dos significados, podría haberse creado otro concepto, que complicaría menos la estructura que la polisemia. En particular, los dos conceptos que se hacen necesarios son “elemento funcional” y “tipo de elemento funcional”. Actualmente se usan los conceptos objeto de rendimiento e instancia de objeto:

· objeto de rendimiento: elemento funcional si el objeto es simple.
· objeto de rendimiento: tipo de elemento funcional si el objeto es múltiple.
· instancia de objeto de rendimiento: elemento funcional del tipo representado por el objeto.

Esto podría haberse simplificarse si hubiera sido planificado del siguiente modo:

· objeto de rendimiento: tipo de elemento funcional
· instancia de objeto de rendimiento: elemento funcional del tipo representado por el objeto.

Si se visualiza la organización como un árbol, actualmente las ramas correspondientes a objetos de rendimiento múltiples son más largas que las que corresponden a objetos de rendimiento simples. Si se hubiera llegado a la definición propuesta por los autores, eso no ocurriría.

Falta de niveles claros

Debido precisamente a la doble definición de objeto de rendimiento, es difícil establecer niveles bien definidos en la organización. Lo que en el punto 1.d del capítulo 1 se llamó nivel de identificación de elemento funcional, en la organización utilizada por Microsoft contiene objetos de rendimiento simples e instancias de objetos de rendimiento múltiples.

Tal como se explicó en la sección 1 del capítulo 1, la organización natural utilizada por los autores permite establecer el nivel de identificación de objeto de

rendimiento, en el que están todos los objetos sean simples o múltiples, y también el nivel de identificación de instancia de objeto de rendimiento, en el que sólo hay instancias de objetos independientemente de si éstos son simples o múltiples.

Casos particulares de acceso

Hay objetos de rendimiento para los que la localización de instancia de objeto no es como la del caso general, con un único identificador. Estos objetos son Procesos y Subprocesos, y es posible solucionar el problema:

· En el caso de Procesos existe la opción de concatenar al nombre de proceso el identificador de proceso. Como éste es único, se estaría localizando exactamente el proceso. Para el caso de Subprocesos existe el identificador de subproceso.

· Para el caso de los Subprocesos, puede ser interesante incluir la información de qué proceso originó el subproceso. Para hacerlo, puede concatenarse el nombre de instancia del proceso (que incluirá nombre simbólico e identificador de proceso) al nombre de instancia del subproceso (que incluirá nombre simbólico e identificador de subproceso)

Con este método se dispone de información suficiente para localizar una instancia de cualquier objeto de rendimiento sólo con el nombre de instancia. Al mismo tiempo se dispone de información adicional que puede ser útil para el proceso de los datos de rendimiento.

d) Falta de paralelismo entre clasificación y localización

Debido a la falta de homogeneidad en la estructura de la organización, no puede establecerse un paralelismo claro y único entre la estructura y la localización de un dato de rendimiento. Esto dificulta la localización del dato por parte de los usuarios en comparación con la modelización propuesta en la sección 1 del capítulo 1. En la sección 2 del capítulo 1 se demuestra la existencia de un paralelismo entre la taxonomía propuesta por los autores en la sección 1 del mismo capítulo y el método de acceso que implicaría.

2. Características destacables de la organización en Linux

A continuación se desarrollan más las características poco adecuadas de la organización de los datos de rendimiento en el sistema operativo Linux. Estas características fueron introducidas en el capítulo 2 y recordadas en la introducción del capítulo 3.

a) Posibilidad de modificación y recompilación

Como se explicó en el capítulo dos, el núcleo del sistema operativo Linux puede ser modificado y recompilado. Este hecho supone que, en particular, el sistema de archivos virtuales */proc* puede ser modificado por el usuario. Por este motivo, no se puede garantizar que la estructura del directorio */proc* vaya a ser siempre la misma. No son características fijas el número de archivos existentes, ni cuáles son dichos archivos, ni la estructura del directorio, ni el contenido mismo de los archivos. Todo esto provoca que el estudio de los datos de rendimiento presentes en algunos archivos del directorio */proc* deba ser particular para cada equipo a analizar.

b) Heterogeneidad de los tipos de archivos en *<u>/proc</u>*

Considerada ya la falta de generalidad del sistema de archivos virtuales */proc*, se estudia ahora el contenido del mismo. Dentro de este directorio no sólo se encuentran datos acerca del rendimiento, sino multitud de archivos diferentes con información totalmente ajena al mismo. De este modo, dentro del directorio */proc* podemos encontrar, además de los datos de rendimiento presentes en algunos archivos, información estática referente a diversas características del hardware instalado en el equipo, así como archivos que contienen datos sobre la configuración del propio sistema operativo. Para agravar esta falta de homogeneidad en el contenido del directorio, existen otros archivos en el mismo que alojan datos de rendimiento mezclados con otra información totalmente ajena a ellos.

Todas estas características hacen que el estudio de los datos de rendimiento dentro del sistema de archivos virtuales */proc* requiera de un profundo conocimiento previo del mismo. Tan sólo en ciertos manuales específicos del sistema operativo se explica cuál es la estructura del directorio */proc* y el contenido de sus archivos. Además, hay que tener en cuenta que estos manuales presuponen un cierto grado de generalidad en el núcleo del sistema operativo, que como se explicó en el apartado anterior, no se puede garantizar.

Por todo ello, el estudio de los datos de rendimiento en equipos que operan bajo el sistema operativo Linux resultará en ocasiones lento y precisará de un estudio previo en profundidad de la estructura de cada archivo presente en el directorio */proc*.

c) Heterogeneidad de los archivos con datos de rendimiento

Una vez que nos centramos en aquellos archivos del directorio */proc* que sí contienen datos de rendimiento, observamos nuevas dificultades. El contenido de estos archivos no es en absoluto homogéneo, de manera que en cada uno de ellos el dato de rendimiento estará alojado en una parte concreta del archivo. Una vez más, esto supone que para acceder a un dato de rendimiento en particular es necesario conocer en detalle cómo está organizada la información dentro del archivo a estudiar.

En el caso de los archivos que contienen datos de rendimiento mezclados con otros que no lo son, este problema resulta evidente. Pero incluso en los archivos que sólo contienen datos de rendimiento, estos se encontrarán en una zona concreta del archivo dependiendo de cada caso particular. Esta disparidad en el formato y la semántica de los archivos del directorio */proc* exige que la aplicación que acceda a los datos de rendimiento requiera de código específico para cada manera de acceder a la información deseada dentro del archivo en estudio.

Además, el modo en que la información de los datos de rendimiento es presentada dentro de los archivos, está en ocasiones orientado a su procesamiento por parte de alguna aplicación informática, mientras que en otros casos parece más adecuado para una lectura personal de los mismos. Esta disparidad no favorece una dinámica homogénea de procesamiento de dichos datos de cara a un futuro análisis de los mismos.

Este problema se multiplica al combinarlo con los anteriores, puesto que el código implementado será particular del núcleo del sistema operativo en el que se desarrolle, ya que como se explicó en el primer apartado, no se puede garantizar la generalidad del sistema operativo en todos los casos.

d) Organización aparentemente no planificada

Las dificultades anteriormente expuestas, así como la observación de la estructura del sistema de archivos virtuales */proc*, hacen pensar en ocasiones que la organización de los datos de rendimiento en el sistema operativo Linux no ha sido demasiado planificada.

Los subdirectorios dentro del directorio */proc* están pensados, aparentemente, para agrupar dentro de ellos a los archivos que contengan información similar. Sin embargo, no se ha encontrado en la literatura existente ninguna definición de esta similitud. Además, se ha comprobado que en múltiples ocasiones los datos de rendimiento no se agrupan en subdirectorios de la manera en que su relación con los elementos funcionales que representan sugiere.

Por todo esto, la organización de los datos de rendimiento en el directorio */proc* da la impresión de haber sido improvisada ad hoc por diferentes equipos de desarrollo cuando la necesidad de ampliación del propio directorio así lo exigía. Esto provoca que la organización de la información resulte, en opinión de los autores, poco clara y usable, reforzando la idea de la necesidad de una nueva taxonomía para los datos de rendimiento.

3. Características deseables en una organización alternativa

A continuación se pasa a hablar de características que serían deseables, a juicio de los autores, en una nueva organización del acceso a los datos de rendimiento:

a) Definición natural de conceptos

Al definir los conceptos en los que se basará la taxonomía debe utilizarse cierto rigor. De ese modo, si en el futuro es necesario añadir nuevos datos de rendimiento, podrá hacerse y encontrar el puesto adecuado en la taxonomía sin necesidad de cambiar ésta. Esto significa que se tendrá al mismo tiempo la flexibilidad para introducir nuevos datos de rendimiento y la potencia de una estructura planificada.

Se comentan brevemente las definiciones que serían necesarias:

Elemento funcional: elemento de un computador que realiza una función. Un elemento funcional puede ser un componente hardware, software, o la combinación de hardware y software.

Dato de rendimiento: medición que puede hacerse sobre uno o varios elementos funcionales de un computador y proporcionar información sobre el rendimiento de éste.

Contador de rendimiento: representación de un dato de rendimiento.

Instancia de objeto de rendimiento: representación de un elemento funcional sobre el que se obtienen datos de rendimiento.

Objeto de rendimiento: representación de un tipo de elementos funcionales sobre los que se obtienen datos de rendimiento.

Una buena definición natural de los conceptos en los que se basa la taxonomía puede satisfacer la mayor parte de las demás características que se explican en esta sección.

b) Reglas concretas para las relaciones entre los conceptos

Entre los niveles de la taxonomía se establecen reglas claras. En la propuesta de los autores serían las que se comentan a continuación:

Un objeto de rendimiento se relaciona con un computador cuando el tipo de elemento funcional representado por el objeto de rendimiento está presente en el computador.

Una instancia de un objeto de rendimiento se relaciona con su objeto de rendimiento porque la instancia representa un elemento funcional del tipo representado por el objeto de rendimiento.

Un contador de rendimiento se relaciona con una instancia de objeto de rendimiento cuando el contador representa un dato de rendimiento que se obtiene del elemento funcional representado por la instancia de objeto

Un dato de rendimiento se relaciona con un contador de rendimiento cuando el contador representa al dato

c) Facilidad para la localización de datos de rendimiento

Al haber definido adecuadamente los conceptos en los que se basa la taxonomía, se establecen niveles en ésta. Esto hace que sea regular y por tanto que sea más fácilmente comprensible por un usuario humano y procesable por una aplicación, un servicio o función del sistema operativo, etc.

Al establecer niveles en la taxonomía resulta natural tener niveles en el acceso a un dato de rendimiento. Como la taxonomía ha sido realizada siguiendo la naturalidad del proceso de selección, la localización resulta natural:

Si se desea conocer el porcentaje del tiempo que el procesador invierte ejecutando un subproceso activo:

De red a computador: se selecciona el computador donde esté el procesador

De computador a objeto de rendimiento: se selecciona el objeto que representa el tipo de los procesadores.

De objeto de rendimiento a instancia de objeto: se selecciona el procesador concreto

De instancia de objeto a contador de rendimiento: se selecciona "% de tiempo de procesador"

d) Facilidad de proceso de los datos de rendimiento

Para satisfacer esta característica se debería disponer de la misma precisión en todos los datos de rendimiento, y por supuesto de precisión suficiente para todos y cada uno de ellos. La disponibilidad de la precisión, así como el nivel de ésta, debería ser información pública y de fácil obtención.

4. Método alternativo de acceso a los datos de rendimiento

Los autores son conscientes de que el implementar la taxonomía presentada de los datos de rendimiento implicaría realizar cambios en los sistemas operativos. También entienden que es posible que ni Microsoft ni los responsables del kernel de Linux estén dispuestos a realizar tales cambios, entre otros motivos por las posibles incompatibilidades con versiones anteriores. Debido a esto se ofrece un método

alternativo para el acceso a los datos de rendimiento de modo que satisfaga las siguientes características:
Facilidad de uso: Los objetivos perseguidos con la taxonomía de los datos de rendimiento presentada estaban dirigidos a facilitar el acceso y utilización de los datos de rendimiento tanto a los usuarios expertos como a los no expertos. El método alternativo para el acceso deberá dar esta facilidad.
Compatibilidad con las organizaciones actuales: Esta característica permitirá la utilización del método alternativo sin modificar los sistemas operativos.
Sin cambios en el sistema operativo: Si se acepta la idea de mantener intacto el sistema operativo, el método debe ser externo a él y utilizar las llamadas al sistema que sean necesarias. Esta característica, como las anteriores, es satisfactible.

Los autores ofrecen un método alternativo para acceder a los datos de rendimiento, de modo que se satisfagan las características anteriores. A pesar de ello, se debe recalcar que no es más que un método para facilitar el uso de los datos de rendimiento a los usuarios. Si se desea que los datos de rendimiento sigan una taxonomía adecuada, deben hacerse cambios en los sistemas operativos.

Cuando se considera la necesidad de facilitar el acceso a los datos de rendimiento sin cambiar la clasificación actual se llega a la conclusión de que no puede realizarse una organización adecuada de los datos de rendimiento de un sistema operativo, sin definir exactamente el objetivo que debe cumplir esa organización. Por supuesto, también habría que definir "organización adecuada" con respecto a ese objetivo. Pero ¿cómo decidir cuál es el objetivo que debe cumplir la organización? Puede responderse de dos modos:

a) Entrevistas a los posibles futuros usuarios

Se puede intentar averiguar cuál es el objetivo de más importancia que debe cumplir una organización alternativa, y después implementarla de este modo. El problema, por supuesto, es que en este caso la organización sólo sería óptima, como mucho, para los usuarios a los que se ha entrevistado y cuyas opiniones han sido evaluadas como más importantes.

Si se quiere seguir esta opción es necesario planificar las entrevistas. Como mínimo debería hacerse lo siguiente:

Selección de la población objetivo:

Para que los resultados de la entrevista sean fiables la población objetivo debe estar bien escogida. Las características de los miembros deben estar bien definidas y ser fáciles de evaluar, y la cantidad de miembros debe ser lo bastante alta como para que los resultados sean representativos. En particular, cuanto menores sean los requisitos que deben cumplir los miembros, mayor deberá ser el tamaño de la población.

Selección de las cuestiones:

No es lo mismo realizar preguntas a usuarios profesionales que a usuarios aficionados debido a la diferencia de conocimiento sobre el tema. El nivel de familiaridad de los miembros de la población con los computadores en general y el estudio de datos de rendimiento en particular debe ser un factor importante no sólo para decidir qué preguntas son adecuadas, sino también para el estilo de las preguntas. El estilo de las preguntas tenderá a ser más directo cuanto mayor sea la familiaridad del

objetivo con los temas tratados, debido a que su nivel de conocimientos le permite responder esas preguntas.

Por otro lado, las respuestas deben ser fácilmente evaluables. De nuevo se encuentra que la profesionalidad de los miembros objetivo es importante. Cuanto mayor sea esta profesionalidad, más precisas serán las respuestas. Igualmente, cuanto mayor sea el nivel de conocimientos sobre computadores y datos de rendimiento mayor será el detalle y precisión de las contestaciones.

b) No limitarse a una única organización

Con esta opción lo que se propone es el uso simultáneo de varias organizaciones de acceso a los datos de rendimiento. En lugar de decidir un único objetivo para la organización, se permiten varios objetivos distintos e incluso la combinación de éstos. Las características que se pretenden obtener con esta opción se explican a continuación:

a) Localización flexible

El hecho de disponer de varias organizaciones simultáneas significa que, como mínimo, podrá buscarse el mismo dato de rendimiento según varios criterios, como mínimo uno por objetivo. Además, según la implementación que se realizara, podría facilitarse más todavía la localización.

La creación de reglas deterministas para una organización conlleva el riesgo de tener que ampliar o redefinir las reglas, ya que no podemos imaginar todos los avances futuros y por tanto no podemos imaginar todos los posibles datos de rendimiento que pueden aparecer. Debido a esto es importante reflexionar y plantear bien esas reglas, de modo que permitan tanto una localización intuitiva de los datos de rendimiento como la extensibilidad de la organización.

b) Organización planificada

La planificación depende en realidad de la implementación de cada una de las organizaciones individuales que compondrán la organización múltiple. Si cada una de las organización está correctamente planificada, entonces la composición de éstas también lo estarán con sólo establecer unos requisitos mínimos para las organizaciones individuales.

c) Facilidad de selección de datos de rendimiento

De nuevo se ve una característica cuyo principal peso descansa en la implementación de las organizaciones individuales y no en la definición de la organización múltiple, en cuanto que ésta sólo necesitaría establecer unos requisitos mínimos para las organizaciones individuales admisibles.

d) Información de formato en los datos de rendimiento

Dado que no se puede asegurar que todos los datos de rendimiento vayan a ser de un tipo específico, en la implementación de la organización múltiple deberá disponerse de información computable sobre el formato de los datos de rendimiento. Esto puede conseguirse con una capa entre las organizaciones individuales y los puntos

de acceso a los datos de rendimiento, de tal modo que se incluya en esa capa información sobre el formato.

Asimismo, también se incluiría la información adicional de acceso que pudiera necesitarse para cada dato de rendimiento particular.

e) Compatibilidad con las organizaciones actuales

Este punto es el más sencillo de satisfacer, ya que basta con que una de las organizaciones individuales sea la organización actual de los datos de rendimiento.

En la figura 1 se ofrece un esquema representativo de la organización múltiple.

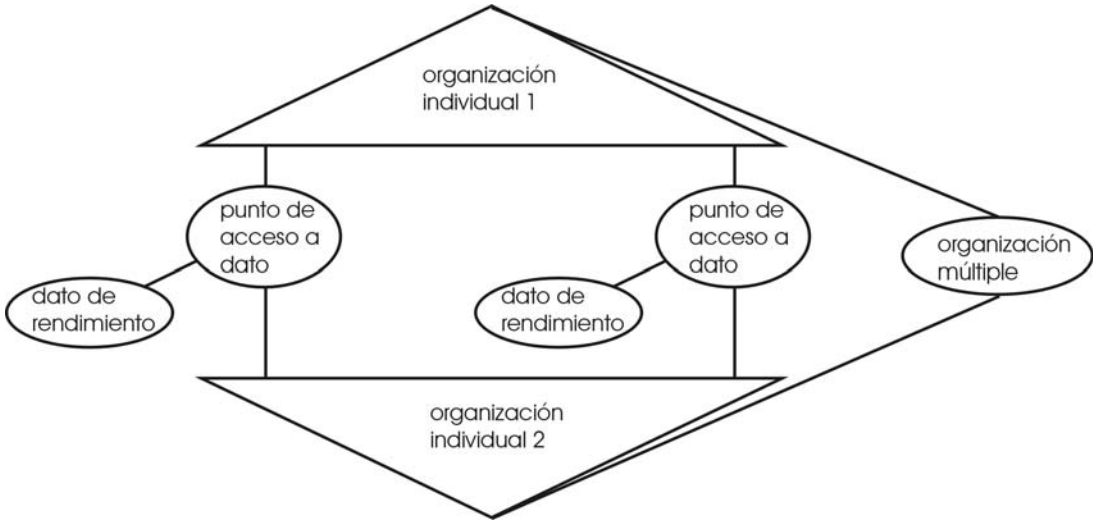

Figura 1

5. Estructura de clases para una implementación de la propuesta

Se presentan dos posibles estructuras de clases para el sistema de multiorganización:

a) Primera opción

Esta estructura se forma con una única capa entre las aplicaciones que vayan a utilizarla y la estructura necesaria para la localización y monitorización de datos de rendimiento. Las aplicaciones trabajarán con instancias de la clase CMultiOrg, que representan multiorganizaciones.

Se dispone de código implementado de esta opción con objetivo Windows. Este código se ofrece en el CD que acompaña a la memoria.

A continuación se muestran las características de las clases. Para una mayor legibilidad no se muestran los métodos accesores y modificadores de los atributos de las clases, ni tampoco sus constructores o destructores. Para una información más detallada, consultar el anexo 3.1:

Clase CDatoRendimiento

Las instancias de esta clase representan puntos de acceso a los datos de rendimiento, y por tanto deben contener toda la información necesaria para el acceso a los datos.

CDatoRendimiento
char* nombre
int compara (CDatoRendimiento*) static CDatoRendimiento** Intersección (CDatoRendimiento**, CDatoRendimiento**, unsigned int, unsigned int, unsigned int&)

Clase abstracta CMultiOrg

Las instancias de clases derivadas de esta representan una multiorganización con la que trabajarían directamente las aplicaciones.

Los atributos bosque, capacidad y nArboles permiten mantener la información sobre los puntos de acceso a los datos de rendimiento.

El atributo monitor lo que permite es realizar la monitorización de los valores de los datos de rendimiento.

CMultiOrg
COrgIndividual** bosque unsigned int capacidad unsigned int nArboles CControlMonitor* monitor
void AnadeDatoRendimiento (CDatoRendimiento*) void EliminaDatoRendimiento (CDatoRendimiento*) void EliminaDatoRendimiento (unsigned int) void EliminaDatoRendimiento (char*) void AnadeOrganizacion (COrgIndividual*) CDatoRendimiento** Busca (char**, unsigned int&) void CargaDatosRendimiento (char*) void GuardaDatosRendimiento (char*) int Consulta (char*, unsigned int&) unsigned int DameNumParametros () char** DameParametros (unsigned int&)

Clase abstracta CControlMonitor

Las instancias de clases derivadas de esta son capaces de monitorizar los valores de los datos de rendimiento incluidos en su lista de consulta. Esta lista de consulta es en realidad una serie de instancias de CDatoRendimiento, es decir, de puntos de acceso a los datos de rendimiento. Para el control de este conjunto es para lo que sirven los atributos capacidadNDatos, nDatos y cjtoConsulta.

CControlMonitor
CDatoRendimiento** cjtoConsulta unsigned int nDatos unsigned int capacidadNDatos
void anade (CDatoRendimiento*) void elimina (CDatoRendimiento*)

void elimina (unsigned int) void elimina (char*) void CargaDatos (char*) void GuardaDatos (char*) int consulta (char*, unsigned int&)

Clase abstracta COrgIndividual

Las instancias de esta clase representan organizaciones individuales de los accesos a los datos de rendimiento.

Su atributo char* permite una fácil identificación de una organización particular.

Sus atributos campos y nCampos contienen información sobre las opciones de búsqueda que permite la organización.

COrgIndividual
char* nombre char** campos int nCampos
CDatoRendimiento** Busca (char**, unsigned int&)

b) Segunda opción

Esta estructura se forma con dos capas entre las aplicaciones que vayan a utilizarla y la estructura necesaria para la localización y monitorización de datos de rendimiento. La capa adicional respecto a la primera opción presentada sirve para aislar más la especificación de los sistemas de localización y monitorización de los datos de rendimiento, de modo que aumenta la modularidad.

Como en la primera opción, las aplicaciones trabajarán con instancias de la clase CMultiOrg, que representan multiorganizaciones.

Se dispone de código implementado de esta opción con objetivo Windows y con objetivo Linux. Este código se ofrece en los anexos 3.4 y 3.5 respectivamente.

A continuación se muestran las características de las clases. Para una mayor legibilidad no se muestran los métodos accesores y modificadores de los atributos de las clases, ni tampoco sus constructores o destructores. Para una información más detallada, consultar el anexo 3.3 y el CD que acompaña a la memoria:

Clase CDatoRendimiento

Las instancias de esta clase representan puntos de acceso a los datos de rendimiento, y por tanto deben contener toda la información necesaria para el acceso a los datos.

CDatoRendimiento
char* nombre
int compara (CDatoRendimiento*) static CDatoRendimiento** Intersección (CDatoRendimiento**, CDatoRendimiento**, unsigned int, unsigned int, unsigned int&)

Clase abstracta CMultiOrg

Las instancias de clases derivadas de esta representan una multiorganización con la que trabajarían directamente las aplicaciones.

El atributo SisOrganización contiene los medios de localización de datos de rendimiento, mientras que el atributo SisConsulta contiene los medios de monitorización.

CMultiOrg
CSistCon* SisConsulta CSisOrg* SisOrganización
void AnadeDatoRendimiento (CDatoRendimiento*) void EliminaDatoRendimiento (CDatoRendimiento*) void EliminaDatoRendimiento (unsigned int) void EliminaDatoRendimiento (char*) void AnadeOrganizacion (COrgIndividual*) CDatoRendimiento** Busca (char**, unsigned int&) void CargaDatosRendimiento (char*) void GuardaDatosRendimiento (char*) int Consulta (char*, unsigned int&) unsigned int DameNumParametros () char** DameParametros (unsigned int&)

Clase abstracta CSistCon

Las instancias de esta clase representan sistemas de monitorización de datos de rendimiento. El atributo monitor es quien realiza realmente las tareas de monitorización. Como se ha dicho, esta clase es principalmente un elemento de separación de CMultiOrg para permitir una mayor modularización.

CSistCon
CControlMonitor* monitor
void AnadeDatoRendimiento (CDatoRendimiento*) void EliminaDatoRendimiento (CDatoRendimiento*) void EliminaDatoRendimiento (unsigned int) void EliminaDatoRendimiento (char*) void AnadeOrganizacion (COrgIndividual*) CDatoRendimiento** Busca (char**, unsigned int&) void CargaDatosRendimiento (char*) void GuardaDatosRendimiento (char*) int Consulta (char*, unsigned int&)

Clase abstracta CControlMonitor

Las instancias de clases derivadas de esta son capaces de monitorizar los valores de los datos de rendimiento incluidos en su lista de consulta. Esta lista de consulta es en realidad una serie de instancias de CDatoRendimiento, es decir, de puntos de acceso a los datos de rendimiento. Para el control de este conjunto es para lo que sirven los atributos capacidadNDatos, nDatos y cjtoConsulta.

CControlMonitor
CDatoRendimiento** cjtoConsulta unsigned int nDatos unsigned int capacidadNDatos
void anade (CDatoRendimiento*) void elimina (CDatoRendimiento*) void elimina (unsigned int) void elimina (char*) void CargaDatos (char*) void GuardaDatos (char*) int consulta (char*, unsigned int&)

Clase abstracta CSisOrg

Las instancias de esta clase soportan el sistema de localización de accesos a datos de rendimiento, y para ello sirven sus atributos, salvo nombre que tiene propósitos identificativos. Como se ha dicho, esta clase es principalmente un elemento de separación de CMultiOrg para permitir una mayor modularización.

CSisOrg
COrgIndividual** bosque unsigned int capacidad unsigned int nArboles
void AnadeOrganizacion (COrgIndividual*) CDatoRendimiento** Busca (char**, unsigned int&)

Clase abstracta COrgIndividual

Las instancias de esta clase representan organizaciones individuales de los accesos a los datos de rendimiento.

Su atributo char* permite una fácil identificación de una organización particular.

Sus atributos campos y nCampos contienen información sobre las opciones de búsqueda que permite la organización.

COrgIndividual
char* nombre char** campos int nCampos
CDatoRendimiento** Busca (char**, unsigned int&)

6. Estudio de las organizaciones individuales necesarias

La estructura propuesta está diseñada para ser abierta, en cuanto a que cualquier desarrollador puede implementar una clase descendiente de COrgIndividual que represente una nueva forma de organización individual. Esto implica que en caso de aparecer la necesidad de una nueva forma de búsqueda, podría seguir usándose la misma interfaz sin más que añadir información sobre la nueva organización.

Por supuesto, para una implementación adecuada de organizaciones individuales, destinadas a ser utilizadas dentro de la organización propuesta, sería aconsejable realizar un estudio sobre las necesidades de búsqueda de los posibles usuarios de estas nuevas organizaciones, de modo que pueda facilitarse la actividad de la mayor carga posible de trabajo de estas personas con un cierto control sobre la cantidad de organizaciones individuales existentes.

Lo primero que habría que hacer sería decidir un segmento adecuado de la población sobre el que realizar el estudio, de modo que la información adquirida a través de éste sirva realmente para encontrar las necesidades y objetivos que deben satisfacer las nuevas organizaciones que se creen.

Una vez seleccionada la población a la que se dirige el estudio, deberá prepararse éste de modo que pueda conseguirse la máxima cantidad de información.

A continuación se ilustran unas breves ideas sobre dos posibles estudios:

Primer posible estudio

Se pretende implementar organizaciones destinadas a su uso por parte de administradores de sistemas. Por ello, se escoge como muestra de la población objetivo a trabajadores en puestos de administración de sistemas informáticos.

Dado que la población objetivo es profesional del sector, puede presuponerse cierto nivel de conocimiento y experiencia. Sabiendo esto, podrá preguntarse de forma bastante directa en busca de conceptos como los siguientes:

· Necesidades de una organización de datos de rendimiento: facilidad de búsqueda, velocidad, uso de bajo nivel, uso de redes, posibilidad de acceder a datos de máquinas distintas, con distintos sistemas operativos, desde el mismo programa...

· Rasgos deseables en una nueva organización que se encuentren en otras organizaciones actuales, sean del sistema operativo que sea y para su implementación en el sistema operativo que sea.

Segundo posible estudio

Se pretende acercar el uso de monitores de rendimiento a los usuarios no profesionales. Por tanto, se escoge como muestra de la población objetivo a usuarios de ordenador, profesionales o no, pero que no tengan conocimientos profundos sobre monitorización del rendimiento de sistemas informáticos. La población seleccionada también debería estar interesada en el uso de monitores de rendimiento.

Como no puede asegurarse el conocimiento por parte de la población, la captura de información puede ser más laboriosa debido por un lado a que podría ser necesario explicar brevemente en qué consiste la monitorización, y después pasar a la búsqueda de conceptos.

· Cómo desearía encontrar los datos de rendimiento

· Cómo se desearía trabajar con datos de rendimiento: viéndolos el propio usuario, teniendo un programa que los examine y avise cuando ocurra algo en particular como escasez de recursos, con la posibilidad de usar ambos métodos.

· Qué tipo de sistema pretende monitorizar: Tanto el tipo de máquina – Pc, Mac... – como el sistema operativo que utiliza – Windows, Linux, Unix...

Como puede verse, el segundo concepto tiene más que ver con el diseño de una aplicación que con el diseño de las organizaciones individuales. Sin embargo, la información que se extraiga de dicho concepto también puede ser útil para el diseño de la organización.

7. Propuestas de organizaciones individuales

A continuación se ofrecen algunas ideas de organizaciones que pueden resultar de interés. En muchas de estas ideas se describe la organización como un árbol, siendo las hojas los puntos de acceso a los datos de rendimiento:

Organización propuesta #1: Organización actual

A pesar de que como ya se dijo la organización actual posee, a juicio de los autores, ciertos inconvenientes, es la que se utiliza actualmente y, como mínimo por razones de compatibilidad, debería poder utilizarse esta opción. Como ya se habla de esta organización en otros lugares, no se cree necesario explicar más aquí.

Organización propuesta #2: Áreas funcionales

El objetivo de esta organización es el de agrupar los datos de rendimiento en base al área funcional de la computadora a la que pertenecen, entendiendo como tal cada una de las partes del sistema operativo que se enumeran a continuación:

Recursos:

Los datos de rendimiento que se agrupen bajo este área funcional serán aquellos encargados de medir la utilización y consumo de los diferentes recursos del sistema, siendo por lo tanto los destinados a detectar dónde se encuentran localizados los posibles cuellos de botella que ralenticen los procesos.

Los hijos que partan de este nodo, que todavía no serán hojas del árbol, es decir, que aún no serán datos de rendimiento, pueden variar de un sistema a otro, dependiendo de cuáles sean los elementos de la computadora con capacidad para consumir recursos de la misma. En un sistema típico estos nodos incluirían el disco duro, la memoria, la CPU, la red, los servicios del sistema operativo, y los servicios de terceras aplicaciones.

De cada uno de estos nodos partirán los diferentes datos de rendimiento que conforman las hojas del árbol, y que serán los que midan, para cada elemento de los citados, el consumo de recursos sobre los mismos.

Rendimiento:

Los datos de rendimiento de este área funcional serán aquellos que permitirán conocer si el sistema está saturado o no. De esta manera, para mejorar el rendimiento de un sistema determinado, el procedimiento a seguir será observar primero, en los datos de rendimiento de este área, si el mismo está sobrecargado, y en caso afirmativo, modificar aquellos elementos en los que se encuentre localizado el cuello de botella, dato que nos proporcionarán los datos de rendimiento del área funcional anterior.

El primer nivel de nodos de este área funcional se compone de los diferentes indicadores de los que se dispone para medir el rendimiento de un sistema, y serían los tiempos de respuesta, el número de operaciones por segundo, los porcentajes de utilización de componentes, etc.

Concurrencia:

Bajo este área funcional están agrupados los datos de rendimiento que se encargan de medir el grado de paralelismo existente en los procesos del sistema.

Los nodos correspondientes al primer nivel de hijos de esta área funcional, de los que partirán los puntos de acceso a datos de rendimiento, serían el número de usuarios conectados, el número de operaciones simultáneas en el instante actual, la longitud media de las colas internas, y cualquier otro dato disponible adecuado.

Bloqueos:

Los datos de rendimiento asociados bajo esta área funcional serán los encargados de detectar problemas de acceso serializado a los recursos del sistema.

Los nodos hijos del primer nivel serían los tiempos de espera, los tiempos de espera a recursos frente a las peticiones totales, etc. De ellos salen los nodos hojas correspondientes a los datos de rendimiento.

Organización propuesta #3: Tipos de medición

En esta organización la división de los diferentes datos de rendimiento se basa en el tipo de mediciones que éstos realizan. Cada uno de estos modos de medir el rendimiento conforma un nodo del primer nivel de hijos que parten del nodo raíz identificador del árbol, y podrían ser los siguientes.

Datos instantáneos:

Los datos de rendimiento agrupados bajo este nodo son aquellos que devuelven como medida un valor numérico simple obtenido en un instante de tiempo concreto, correspondiente al extremo del intervalo de muestreo que se haya fijado.

Es conveniente usar estos datos con las debidas precauciones, puesto que con cierta frecuencia pueden ofrecer medidas confusas. Este hecho se debe fundamentalmente a que los valores corresponden únicamente al instante de tiempo indicado por el intervalo de muestreo, de manera que un cambio de valor brusco que genere un pico en mitad de dicho intervalo podría pasar desapercibido en la medición. Es por ello que se recomienda utilizar estos datos para medir variables que no varíen bruscamente en el tiempo, o que sean acumulativas (como valores máximos o sumatorios).

Datos media:

Los datos de rendimiento que parten de este nodo son, en general, más fiables que los que lo hacen del anterior, por las razones anteriormente mencionadas. Por ello, se recomienda la utilización de éstos antes que la de aquellos, siempre que sea posible y se intuya que la medición puede tener picos bruscos.

Los datos de este tipo se encargan de acumular en dos variables la suma de los valores de las medidas realizadas hasta el momento y el número de las mismas, respectivamente, mostrando como resultado de la consulta la división aritmética entre ambas. Para que un cambio brusco en los valores del Dato sea apreciable en las medidas tomadas, es conveniente que la media de los mismos se realice exclusivamente para el último período de muestreo, y no para el tiempo total desde el que se arrancó la aplicación. En cualquier caso, existen datos de rendimiento de ambos tipos,

dependiendo de la funcionalidad deseada, y los dos se agrupan bajo este nodo de la rama Tipos de medición.

Datos de frecuencia:

Los datos de rendimiento agrupados bajo este nodo miden la frecuencia con que se realizan determinadas operaciones sobre el sistema, es decir, el número de las mismas efectuado por segundo. Estas operaciones deberán ser bastante habituales para que las medidas obtenidas sean significativas, cosa que no ocurriría, por ejemplo, si se intentara medir el número de inicios de sesión por segundo.

Datos de tiempo medio:

Los datos de rendimiento de este tipo son una mezcla de los dos anteriores, y miden el tiempo medio de ejecución de un determinado proceso. Resultan especialmente útiles, puesto que proporcionan una medida muy aproximada del tiempo de respuesta del sistema.

Datos de porcentaje:

Bajo este nodo se asocian los datos de rendimiento de los que se pueden obtener medidas significativas sobre el uso de los recursos. El porcentaje al que se refiere la denominación estará habitualmente referido a medidas de tiempo o a la utilización de recursos.

Datos de longitud de colas:

Los datos de rendimiento que parten de este nodo de la rama se encargan de medir el número de peticiones que estarán alojadas en la cola de un recurso a la espera de la utilización del mismo.

Datos de diferencia:

Este nodo agrupa a los datos de rendimiento encargados de aquellas medidas en las que un valor instantáneo no es tan relevante como una variación de la misma.

En ocasiones podemos encontrarnos con que un mismo factor está expresado por un Dato de rendimiento instantáneo y por uno de diferencia, lo que nos permitirá un análisis más interesante del mismo.

4. Benchmarks o comparadores de rendimiento

0. Introducción

Los valores de los datos de rendimiento, por sí solos, no contienen información, ya que sólo son números. La información se consigue al analizar estos valores valiéndose de ciertos conocimientos:
La medida de los datos: Instrucciones por segundo, porcentaje de tiempo, número actual de procesos, etc
Las circunstancias de obtención de un valor determinado: Un valor alto en porcentaje de tiempo de escritura en disco debe alcanzarse cuando se escribe con alta frecuencia en el disco duro; un valor alto en datagramas IP recibidos se alcanzará cuando una conexión tiene un alto grado de actividad, etc.

Una vez que se puede acceder a los datos de rendimiento de un ordenador, una actividad que puede resultar de interés es el análisis del rendimiento del ordenador. Con tal fin, deberán planificarse baterías de pruebas que puedan repetirse. De ese modo, los resultados tendrán una validez científica. Habitualmente, para realizar estudios sistemáticos del rendimiento de un computador se utilizan los benchmarks.

Un benchmark es el resultado de la ejecución de uno o varios programas informáticos en un ordenador, con el objetivo de estimar el rendimiento de elementos concretos o el sistema completo, frecuentemente en comparación con algún parámetro de referencia. De ese modo pueden compararse los resultados con máquinas similares. Por extensión, se entiende como benchmark el programa ejecutado en la máquina para realizar dicha estimación. [MSS1, CEC1 y WIK1]

En origen, la tarea de ejecutar un benchmark se reducía a estimar el tiempo de proceso que llevaba la ejecución de un programa. Con paso del tiempo, la mejora en los compiladores y la gran variedad de arquitecturas y situaciones existentes convirtieron esta técnica en toda una especialidad. [WIK1]

La mayoría de los benchmarks hallados en la literatura consultada están diseñados para comparar el rendimiento de procesadores. Actualmente es más difícil la elección de las condiciones bajo las que dos sistemas informáticos distintos pueden compararse entre sí. La razón de esto está en las mejoras en compiladores, optimizadores de código, hardware especializado, etc. La consecuencia es que la publicación de los resultados de los benchmarks suele ser objeto de candentes debates cuando se abren a la comunidad. [WIK1]

Para intentar estandarizar el proceso de comparación nació el benchmark SPEC, pero la misma esencia del benchmarking, con infinidad de escenarios posibles, hace que la aplicación del mismo deba ser siempre puesta a consideración de quien deba interpretar los resultados. [WIK1]

1. Uso de un benchmark

Dos posibles aplicaciones del uso de la técnica de benchmark podrían ser las siguientes:

· Obtener datos que puedan indicar la superioridad de un sistema o arquitectura determinado sobre otro.
· Estudiar el rendimiento de una arquitectura de computadoras paralelas. Este tipo de benchmark se conoce también como workload o carga de trabajo.
[CEC1]

Un benchmark puede medir el rendimiento de un elemento funcional de un ordenador, o incluso más específicamente, atendiendo al rendimiento de una función concreta de un elemento, como por ejemplo las operaciones en coma flotante dentro de la CPU. Un benchmark puede evaluar el rendimiento de un sistema informático al completo, pero sólo en teoría. Actualmente no se utilizan para esto debido a dificultades del diseño y el análisis de los resultados que pudieran conseguirse. [MSS1 y WIK1]

La creación del benchmark suele tomar como parámetro de referencia un sistema determinado. De este modo, los resultados de la ejecución de dicho benchmark sobre otros sistemas pueden ser comparados entre sí a través de la relación de superioridad o inferioridad con el sistema de referencia del benchmark.

2. Clasificación de los benchmarks

En la literatura consultada han sido halladas varias clasificaciones de benchmarks, formando grupos de los cuales se comentan algunos a continuación, a título ilustrativo:

1. Pruebas basadas en aplicación (*application-based*).

Este tipo de benchmarks consisten en una aplicación que se ejecuta sobre el equipo del cual se desea medir el rendimiento. Esta aplicación, que dependiendo de qué tipo de análisis quiera realizar cargará unos recursos y otros del sistema, incorpora en su propio código las mediciones de tiempo oportunas para devolver ella misma, a su finalización, los resultados de rendimiento obtenidos. [MSS1]

2. Pruebas playback (*playback test*).

Los benchmarks de esta clase consisten en aplicaciones que realizan llamadas internas al sistema durante actividades específicas de una aplicación. Estas llamadas al sistema se ejecutarán en modo kernel, y por separado del resto de la aplicación. Estas llamadas internas provocarán una serie de retardos y otras medidas que serán las que se analicen como resultado de la medición. [MSS1]

3. Pruebas sintéticas (*synthetic test*).

Este tipo de benchmarks consisten en aplicaciones que ejecutan tareas específicas para un elemento concreto del sistema. De este modo el resultado que se obtiene está algo más acotado que el de los demás. Su uso suele estar complementado con un análisis estadístico y comparativo a posteriori, a través de la utilización de extensas bases de datos existentes para cada elemento de estudio sobre el que se realice la prueba. [MSS1]

4. Aplicación Real:

Es un tipo de benchmark que utiliza una aplicación, a la cual se le han agregado funciones para medir diversos parámetros del rendimiento de la aplicación o el sistema. [CEC1]

5. Kernels

Un benchmark kernel es un pequeño programa que encapsula un lazo interno de alguna aplicación. El objetivo aquí es gastar tiempo ejecutando un código kernel infinitas veces. [CEC1]

6. Sintéticos

Los benchmark sintéticos son programas que tienen ciertas propiedades estadísticas que se unen con extensas bases de datos de programas para tareas específicas. [CEC1]

7. Comerciales

La cantidad disponible de este tipo de benchmark es inmensa. La mayoría de estos programas han sido desarrollados para ayudar a tomar decisiones, por ejemplo los laboratorios BAPco y ZD poseen una serie de programas que tratan de medir el rendimiento de PC's compatibles en aplicaciones reales. Otros como AIM, tratan de asistir en la toma de decisiones cuando se hacen comparaciones cruzadas de arquitectura. [CEC1]

8. Aplicación Específica

A esta clase pertenecen los benchmarks que se construyen sobre la base de una especificación, y no sobre un programa real. Los desarrolladores tienen una gran libertad para elegir cómo construir el benchmark, pero están obligados a cumplir con ciertas especificaciones del cliente. [CEC1]

9. Benchmarks juguete

Este tipo de programas realiza pruebas sobre algunos aspectos del rendimiento el sistema, uno de los mas famosos es el benchmark de Integración de Stanford que fue usado para estudiar el rendimiento de máquinas RISC. [CEC 1]

3. Benchmarks conocidos

Existe un cierto número de benchmarks reconocidos internacionalmente. Sin embargo, sólo unos pocos guardan relación con programas reales. Es decir, se supone que los resultados arrojados miden algo intrínseco a la máquina, pero en realidad es difícil identificar con seguridad qué es lo que está siendo medido. Esta situación es análoga a lo que ocurre cuando se trata de medir la inteligencia humana con un test. [CEC1]

A continuación se comentan algunos de los benchmarks más populares:

1. Whetstone

Whetstone es un pequeño benchmark científico diseñado en el Laboratorio Nacional de Física de Inglaterra. Está considerado como uno de los padres de los benchmarks sintéticos por ser el primero diseñado específicamente con ese fin. Hoy en día forma parte de muchos benchmarks actuales. [WIK 1]

Historia y características generales

El Whetstone original fue diseñado en los años 60 por Brian Wichmann en el National Physical Laboratory, en Inglaterra, como un test para un compilador ALGOL 60 para una máquina hipotética. El sistema de compilación fue denominado Whetstone, localidad en la que fue diseñado, y el nombre parece haber pasado al benchmark en sí mismo. El programa fue diseñado basándose en el estudio de 949 programas Algol '60, por lo que constituye un benchmark “sintético”. [WIK1]

Debido a que sus creadores se basaron en programas científicos de la época, este benchmark puede considerarse como científico. Su importancia histórica radica en que fue el primer programa diseñado para benchmarking. La primera implementación práctica del Whetstone fue escrita por Harold Curnow en Fortran en 1972. Posteriormente fue traducido a varios lenguajes. [WIK1]

Fue diseñado para medir la velocidad de ejecución de una variedad de instrucciones de punto flotante (suma, producto, funciones trigonométricas, exponenciales, logaritmos y raíces) en datos escalares y vectoriales, aunque también contiene código de enteros, condicionales y llamadas a procedimientos. [WIK 1]

En sus comienzos demostró ser una buena medida de rendimiento, sin embargo con el tiempo su funcionalidad se vio reducida por su alta sensibilidad a optimizadores. A finales de los 80 y comienzos de los 90 se reconoció que el Whetstone no funcionaría adecuadamente para medir el rendimiento de supercomputadoras con multiprocesadores paralelos. Sin embargo el Whetstone aun se utiliza ampliamente, porque provee una medida muy razonable del rendimiento de monoprocesadores de aritmética flotante. Uno de sus mayores usos actualmente es como parte de otros benchmarks. [WIK1]

Características importantes

Su código fuente es corto y relativamente fácil de entender. El tiempo de ejecución es corto: cien segundos (por diseño). Los resultados son muy precisos (tiene pequeñas variaciones). [WIK1]

El bucle principal del Whetstone se ejecuta en unos pocos milisegundos en una máquina moderna promedio, así que sus diseñadores resolvieron proveer un procedimiento de calibración. Este método ejecuta varias series de repeticiones del bucle principal hasta que la calibración lleva más de dos segundos, y luego calcula un número de pasos extra para que la ejecución se realice en cien segundos. Luego ejecuta pasadas extra para cada una de las ocho secciones del bucle principal, midiendo el tiempo de ejecución de cada una y calculará los MWIPS. El Whetstone ejecuta once

bucles, cada uno de los cuales debe evaluar un tipo distinto de instrucciones: saltos condicionales, aritmética entera, funciones trigonométricas, etc. [WIK1]

Al comenzar el programa lo que se hace es establecer la cantidad de veces que se iterará cada bucle, en otras palabras, se asigna un peso a cada tipo de instrucción o bucle. El problema de este enfoque es que cada bucle tiene su propio tiempo de ejecución, por lo que la relación entre número de iteraciones y peso no es tan directa. Finalmente se llama a cada uno de estos bucles, se imprimen sus resultados y se calcula el rendimiento en MWIPS. [WIK1]

Problemas

· El mayor problema del Whetstone es que no cuenta con una versión oficial, por lo que hay que tener mucho cuidado al comparar los resultados de distintas máquinas y asegurarse de que en ambas evaluaciones se utilizó la misma versión del programa, y sobre todo la misma cantidad de iteraciones por bucle. [WIK1]

· El benchmark Whetstone consume entre el 40 y 50% del tiempo de ejecución en subrutinas matemáticas, por lo que las librería usadas para las pruebas pueden alterar los resultados significativamente. Por ejemplo, algunos sistemas tienen dos versiones de subrutinas de punto flotante: una cumple con el estándar IEEE de punto flotante y la segunda es mucho más ligera pero también mucho menos precisa. [WIK1]

· La mayoría de sus variables son globales y no mostrarán las ventajas de las arquitecturas como RISC donde la presencia de un gran número de registros del procesador optimizan el uso de variables locales. Por esta misma razón colocar las variables globales en los registros del procesador incrementa su velocidad de ejecución. [WIK 1]

· Las funciones de librerías matemáticas están sobrerrepresentadas. [WIK1]

2. Dhrystone

El Dhrystone es un pequeño benchmark sintético que pretende ser representativo de programación entera de sistemas. Está basado en estadísticas publicadas sobre uso de particularidades de los lenguajes de programación, sistemas operativos, compiladores, editores, etc. [WIK1]

Historia y características generales

La publicación original de este benchmark se puede ver en CACM 27,10 (Oct. 1984), 1013-1030, por Reinkol P. Weicker de Siemens-Nixford Information System. Originalmente fue publicado en ADA, aunque hoy en día la mayoría utiliza la versión en C distribuida por Rick Richardson. La versión 2 fue publicada en SIGPLAN Notices 23,8 (Ago. 1988), 49-62, junto con las reglas de medición y en tres lenguajes: Ada, C y Pascal. Meses después salió la última versión: la 2.1. La versión 1 ya no se recomienda debido a que los compiladores actuales pueden eliminar mucho del "código muerto" del benchmark (sin embargo, los números MIPS citados son frecuentemente extraídos de la versión 1). Su nombre representa un juego de palabras con su antecesor: el Whetstone (“wet” = mojado, “dry” = seco). Es un benchmark sintético y comercial. [WIK1]

Características importantes

Contiene muchas instrucciones simples, llamadas a procedimiento y condicionales, y pocas de coma flotante y bucles. No realiza llamadas al sistema. Usa pocas variables globales y ejecuta operaciones con punteros. Está compuesto por doce procedimientos incluidos en un bucle de medida con 94 sentencias. No se puede variar su tamaño. Está compuesto por un 53% de instrucciones de asignación, 32% de instrucciones de control y un 15% de llamadas a procedimiento. Dhrystone es bastante compacto (no más de 1,5 KB), y por ello cabe completamente en la caché interna del procesador. De esta manera no mide el resto del sistema pero presenta la ventaja de que mide solamente la capacidad del procesador.

El Dhrystone compara el rendimiento del procesador usando una máquina de referencia: la VAX 11/780 es la máquina que corre a 1 MIP (logra 1757 Dhrystones por segundo). A diferencia del Whetstone, donde los distintos tipos de instrucción están en bucles distintos, en Dhrystone no existen tales divisiones, por lo que no se puede cambiar la importancia de cada tipo de instrucción. [WIK1]

Problemas

· El problema principal es su escasa representatividad y su alta sensibilidad frente a compiladores, lo cual desalienta su uso.

· Un considerable porcentaje del tiempo se gasta en funciones de manejo de cadenas de caracteres, lo que hace al test muy dependiente de la manera en que estas operaciones son ejecutadas. En pocas palabras, es muy susceptible a optimizaciones de rutinas críticas por los fabricantes. [WIK1]

· En la VAX 11/785 con el Berkeley UNIX (4.2) los compiladores consumieron 23% del tiempo en estas funciones de cadenas de caracteres. En máquinas RISC (donde se gasta menor tiempo en secuencias de llamadas a procedimiento que en la VAX) y con mejores compiladores y optimizadores, el porcentaje varía del 21% al 65%, lo que está por encima de las aplicaciones típicas de hoy día. Es muy difícil que existan aplicaciones que posean una proporción tan importante de uso de librerías de C o incluso de llamadas a librerías específicas. [WIK1]

· No existe el proceso de certificación necesario para citar los resultados del Dhrystone. Esto es una ventaja en cuanto a la facilidad de trabajar con los resultados, pero una desventaja por el hecho de que uno debe examinar bien estos resultados cuando son externos antes de llegar a alguna conclusión. [WIK1]

· Hay que tener en cuenta es la dependencia del compilador, ya que es un aspecto clave en el Dhrystone junto con la tecnología de librería usada - mucho más que en cualquier otro benchmark. Se han encontrado diferencias de hasta un 100% con la misma máquina y diferentes compiladores. [WIK1]

· En lugar de usar el formato normal de Kernighan & Ritchie (K&R) muchas veces el código es alterado para cumplir con los formatos ANSI. Este cambio puede otorgar mucha más información para el compilador, lo cual redunda en una mejor optimización. [WIK1]

· Combinar los dos archivos fuentes en un único módulo habilita al compilador a ejecutar optimizaciones adicionales en todas las funciones simultáneamente. [WIK1]

· Habilitar el inline de procedimientos puede mejorar el rendimiento. Con la disponibilidad del ANSI C la definición de la función de cadena de caracteres no era parte del lenguaje; ahora sí, con lo que se permite explícitamente poner inline a las

funciones de librería. La regla Dhrystone "No usar inline en procedimientos para Dhrystone" se refiere sólo al nivel de procedimientos del usuario y no al de rutinas de librería. [WIK1]

· La forma de medir el resultado se basa en el reloj del procesador y la cantidad de tiempo que se ejecuta es demasiado corta (algo más de 2 segundos). [WIK1]

· Adaptar las librerías de C para acelere el código de Dhrystone puede alterar los resultados del benchmark de forma drástica, aunque esto pueda influir negativamente en la performance de código típico. [WIK1]

3. Linpack

Historia y características generales

El benchmark Linpack fue desarrollado en el Argone National Laboratory por Jack Dongarra en 1.976, y es uno de los más usados en sistemas científicos y de ingeniería. Su uso como benchmark fue accidental, ya que originalmente fue una extensión del programa Linpack -cuyo propósito era resolver sistemas de ecuaciones- que otorgaba el tiempo de ejecución del programa en 23 máquinas distintas. Luego fueron agregándose cada vez mayor cantidad de máquinas (según sus mismo autores, principalmente como un pasatiempo). Hoy en día, el programa Linpack ha sido reemplazado por el paquete Lapack, el cual hace mejor uso de las características de la arquitectura RISC. El benchmark Linpack puede conseguirse en versión Fortran, C y como un applet de Java. [WIK 1]

Características importantes

La característica principal de Linpack es que hace un uso muy intensivo de las operaciones de punto flotante, por lo que sus resultados son muy dependientes de la capacidad de la FPU que tenga el sistema. Además pasa la mayor parte del tiempo ejecutando unas rutinas llamadas BLAS (Basic Linear Algebra Subroutines o Subrutinas de Álgebra Lineal Básica). Como hay dos tipos de estas librerías, el resultado dependerá mucho de esto. Por lejos, el mayor tiempo de ejecución se consume en la rutina DAXPY de la librería BLAS (casi el 90%). DAXPY realiza el siguiente cálculo

y(i) := y(i) + a * x(i).

Casi se puede decir que lo que mide Linpack es la performance del sistema para DAXPY. Por otra parte, al ser esencialmente cálculo con matrices fácilmente vectorizables, se puede llegar a medir la eficiencia de sistemas con multiprocesador. [WIK1]

El reporte del benchmark describe el rendimiento para resolver un problema de matrices generales Ax = b a tres niveles de tamaño y oportunidad de optimización: problemas de 100 x 100 (optimización del bucle interno), problemas de 1.000 por 1.000 (optimización de tres bucles - el programa entero) y un problema paralelo escalable. Estas matrices son generadas usando un generador de números pseudoaleatorios, pero forzando los números para que pueda ejecutarse un pivoteo parcial con Eliminación Gaussiana. Ejecuta dos rutinas básicas: una que descompone la matriz, y otra que resuelve el sistema de ecuaciones basándose en la descomposición. Para una matriz de n

x n la primera rutina lleva n^3 operaciones de coma flotante, mientras que la segunda ejecuta n^2 operaciones de coma flotante. [WIK1]

Linpack está compuesto de tres benchmarks: los ya mencionados de orden 100 y orden 1.000, y un tercero de Computación Altamente Paralela (un algoritmo especialmente diseñado para buscar los beneficios de los multiprocesadores). En el primero sólo se permite cambiar las opciones del compilador para hacerlo más eficiente; sin embargo debe especificarse antes de ejecutarlo una función SECOND que devuelva el tiempo de ejecución en segundos del programa. [WIK1]

Los resultados se expresan en MFlops (millones de operaciones de punto flotante por segundo), siempre referidas a operaciones de suma y multiplicación de doble precisión (64 bits, aunque para algunos sistemas esta cantidad de bits es la precisión simple). [WIK1]

Problemas

· El Linpack es un programa que ejecuta cálculos con matrices, y como tal, existen numerosas técnicas para optimizarlo, como el loop-unrolling. Esto implica que los resultados de la ejecución son muy susceptibles a la utilización de técnicas que mejoren la velocidad de los cálculos con matrices. [WIK1]

4. Ciusbet

Historia y características generales

Ciusbet Hardware BenchMark, creado por Ciusbet en 2003, es un importante benchmark para analizar varios componentes de un ordenador. Ha sido diseñado consultando con gente especializada en hardware y overclockers, para garantizar que el análisis es fiable y útil. Es un desarrollo freeware, y está considerado uno de los aplicativos de benchmarking más útiles de ese sector. Es también uno de los primeros benchmarks creados por un programador español. [WIK1]

Características importantes

Este software cuenta con un gran abanico de pruebas, CPU y Caché, disco duro, tarjeta gráfica, memoria, y diversas pruebas aisladas para chequear el funcionamiento de distintos componentes. El benchmark se actualiza con las nuevas versiones, gracias a un actualizador automático. Así, el usuario se asegura de que analiza su equipo aprovechando las tecnologías más recientes. El programa fue desarrollado y probado con la colaboración de la página de Coyotes Hardware que se ha convertido en la página oficial del benchmark. [WIK1]

Problemas

· EL programa ejecutable es bastante grande, así que es posible que genere tráfico con la caché, lo que puede influir sobre la medida de otros recursos. Por otro lado, el programa está diseñado para ser utilizado específicamente en Windows, lo que limita su uso a ordenadores que funcionen bajo este sistema operativo.

5. Livermore

Livermore es un benchmark que consiste en una serie de rutinas computacionales desarrolladas por el Laboratorio Nacional de Lawrence Livermore (Livermore, California), y es utilizado para probar el rendimiento de sistemas en cuanto a aritmética de coma flotante escalares y vectoriales. Fue introducido originalmente en 1970, cuando estaba compuesto de 14 kernels de aplicaciones numéricas de coma flotante escritas en Fortran (aunque hoy existe una traducción a C); luego este número fue elevado a 24 en los años 80. [WIK1]

Historia y características generales

El motivo de su desarrollo fue que en este laboratorio les era imposible realizar benchmarks con programas que formen parte de su carga de trabajo normal. Estas aplicaciones reales normalmente excedían las 100.000 líneas de código, tenían muchas rutinas orientadas a hardware específico (lo que complica su portabilidad) y además la gran mayoría de este software es secreto. Dada la situación, resolvieron crear un benchmark con aplicaciones que fueran representativas de su carga de trabajo. [WIK1]

El resultado fue un benchmark que sirve para medir la capacidad del procesador en supercomputadoras. No es un benchmark sintético, ya que sus módulos provienen de partes de computaciones reales (a diferencia del Whetstone y el Dhrystone, que intentaban reproducir proporcionalmente las instrucciones más usadas por los programas de la época). [WIK1]

Características importantes

El código fuente es un programa de un solo nivel contenido en un único archivo fuente que es fácilmente compilado y enlazado. La única función dependiente del sistema es la que llama al reloj para obtener la medición del tiempo de ejecución. Si bien es necesaria la librería de funciones matemáticas estándar, la ejecución de estas no consume la mayor parte del tiempo como sucede con otros benchmarks. [WIK1]

Ciertos kernels son vectorizables, y la reprogramación para aprovechar esta característica es permitida siempre y cuando su uso sea solamente para la demostración de las capacidades de una determinada tecnología. Los resultados deben ser enviados al ASCI (Accelerated Strategic Computing Iniciative o Iniciativa de Computación Estratégica Acelerada) del Laboratorio Lawrence Livermore, quienes se encargan de poner las reglas y verificar los resultados que son publicados. [WIK1]

La medida de rendimiento está en término de Millones de Operaciones de Punto Flotante Por Segundo o MFLOPS. El programa además chequea la precisión computacional de los resultados (realiza una suma de comprobación y determina cuánto se aleja del estándar de punto flotante IEEE 754). Una de sus características es que no produce comparaciones de rendimiento con un solo número, sino que ejecuta los 24 kernels en tres distintos bucles que producen sendas distintas medidas de rendimiento: para vectores cortos, medios y largos. Otorga también una serie de medidas generales, que consisten en la media aritmética, media geométrica y media armónica, mínimo y máximo de los resultados. [WIK1]

6. iComp

Historia y características generales

iCOMP es un benchmark desarrollado por Intel para poder medir el rendimiento de sus procesadores, cuando cambios arquitectónicos impedían una comparación únicamente por la frecuencia de reloj. iCOMP es un acrónimo de Intel COmparative Microprocessor Performance.

Este benchmark no ha tenido la repercusión que Intel esperaba. [WIK1]

El tipo de computadoras que intenta medir son las computadoras personales y servidores de red basados en microprocesadores Intel. [WIK1]

Su resultado expresa el rendimiento relativo de los procesadores Intel respecto a algún procesador Intel que se considera base (el mismo fue cambiando con las distintas versiones del iCOMP). Para esto calcula la media geométrica ponderada de los distintos benchmarks que lo componen. [WIK 1]

Estas medidas fueron escogidas entre una serie de benchmarks públicamente disponibles. Entre las razones esgrimidas para su elección se tienen que cada uno mide una característica especial de la arquitectura Intel; además utilizan una mezcla de aplicaciones que se encuentran hoy en día en la mayoría de los sistemas. Al otorgarle distintos pesos a cada uno Intel también establece cuáles son de esas características las más importantes para las aplicaciones del momento. [WIK1]

Posteriormente salió al público la versión 3.0, que tomó como base al Pentium II de 350 MHz y nuevamente cambió la mezcla que conforma el benchmark general. [WIK1]

Obviamente, al ser un benchmark definido por el mismo Intel responde a sus propios intereses, particularmente se nota que la relación entre puntajes del iCOMP siempre supera al del reloj. Además, a medida que fue cambiando su arquitectura fue cambiando este benchmark para que dé mejores resultados según las nuevas características que iba incorporando. [WIK1]

7. SPEC

Historia y características generales

Standard Performance Evaluation Corporation (SPEC), es un consorcio sin fines de lucro que incluye a vendedores de computadoras, integradores de sistemas, universidades, grupos de investigación, publicadores y consultores de todo el mundo.

Sus objetivos son crear benchmark estándar para medir el rendimiento de computadores y controlar y publicar los resultados de estos tests. Actualmente incluyen tres grupos: OSG, HPG y GPC: [WIK1]

· Open Systems Group (OSG, Grupo de Sistemas Abiertos) realiza benchmarks de nivel de componentes y sistemas en ambientes UNIX / NT / VMS. Es el grupo original a partir del cual surgió el SPEC, y sus benchmarks son probablemente los más conocidos. Además del conocido SPEC CPU2000 suite, el grupo ha desarrollado

benchmarks para Java, para servidores web, un benchmark para servidores de correos y uno de Sistema de Archivos de Red. [WIK1]

Por otra parte, tiene otra serie de benchmarks en desarrollo; entre los cuales se puede encontrar un nuevo benchmark de servidor de correos que incluye el protocolo iMAP. [WIK1]

· High Performance Group (HPG, Grupo de Alto Rendimiento) realiza benchmarks en un ambiente de computación numérico, con énfasis en computación numérica de alto rendimiento. El nicho que intenta cubrir este benchmark incluye sistemas con multiprocesadores simétricos, clusters de estaciones de trabajo, sistemas paralelos con memoria distribuida y las tradicionales supercomputadoras vectoriales y vectoriales paralelas. [WIK1]

· Graphics Performance Characterization (GPC, Caracterización del Rendimiento de Gráficos) desarrolla benchmarks para subsistemas gráficos, OpenGL y Xwindows. [WIK1]

SPEC CPU2000 es un benchmark producido por la SPEC. Fue creado con el fin de proveer una medida de rendimiento que pueda ser usado para compara cargas de trabajo intensivas en cómputo en distintos sistemas de computadora. SPEC CPU2000 contiene dos benchmark suites: CINT2000 para medir y comparar el rendimiento de computación intensiva de enteros, y CFP2000 para medir y comparar el rendimiento de computación intensiva en flotantes. La “C” en CINT2000 y CFP2000 denotan que son benchmarks a nivel de componentes, en oposición a benchmarks de todo el sistema.

Al ser intensivos en cómputos, miden el rendimiento del procesador de la computadora, la arquitectura de la memoria y el compilador. El CINT2000 y el CFP2000 no fuerzan la E/S (unidades de disco), trabajo en red o gráficos. Si bien pueden utilizarse estos benchmariks para probar un sistema de forma tal que alguno de estos componentes afecten el rendimiento del equipo, no es ese el objetivos de estas suites.

SPEC provee lo siguiente en el paquete CPU2000:

· Las herramientas del CPU2000 para compilar, ejecutar y validar los benchmarks en una variedad de sistemas operativos.

· El código fuente de las herramientas, de manera que puedan ser compiladas para los sistemas no cubiertos por las herramientas pre-compiladas

· El código fuente de los benchmarks herramientas para la generación de informes de rendimiento

· Reglas de ejecución e informes que definen cómo deberían ser usados los benchmarks para producir resultados estándar la documentación.

SPEC CPU2000 incluye herramientas para la mayoría de los sistemas operativos UNIX y para Windows NT. Productos adicionales para otros sistemas operativos serán lanzados si SPEC detecta suficiente demanda. CINT2000 contiene 11 aplicaciones escritas en C y una en C++ que son usadas como benchmarks. [WIK1]
CFP2000 contiene 14 aplicaciones (seis en FORTRAN77, cuatro en FORTRAN90 y cuatro en C) que son usadas como benchmarks: [WIK1]

Resultados

El SPEC2000 provee cuatro medidas para cada componente (consistente en la media geométrica de los tiempos de ejecución de los programas). Esto es porque tiene dos clasificaciones. La primera es en cuanto a la optimización del compilador por el usuario:

· Base (“conservadora”): sirve para encuadrar a los usuarios que compilan con las opciones de rendimiento generales ofrecidas por el compilador, de manera que tiene una serie de referencias establecidas para usar las opciones del compilador.

· No base (“agresiva”): intenta encuadrar a los usuarios que intentan lograr el mejor rendimiento de sus programas. Son opcionales y tienen requerimientos mucho menos estrictos (por ej. pueden usarse diferentes opciones de compilación para distintos programas escritos en el mismo lenguaje)

La segunda clasificación se refiere a la cantidad de tareas que se corren al mismo tiempo:

· No rate (monotarea): corresponde a la velocidad de ejecución de una sola tarea.

· Rate (multitarea): corresponde a la capacidad de la máquina de llevar a cabo un cierto número de tareas similares. Tradicionalmente se usa para medir el rendimiento de multiprocesadores.

La combinación de estas dos clasificaciones nos otorga 4 medidas distintas para el CINT2000 y 4 para el CFP2000. SPEC usa una Sun Ultra5_10 con un procesador de 300MHz como máquina de referencia. Toma aproximadamente dos días hacer una corrida conforme a SPEC (por lo menos tres repeticiones de cada benchmark para asegurar la validez de los resultados) de CINT2000 y CFP2000 en esta máquina.

4. Problemas en el uso de benchmarks

El uso de benchmarks para generar información sobre el rendimiento de un sistema posee ciertas características que pueden resultar problemáticas:

a) Orientados a subsistemas

Los benchmark actuales están orientados al estudio de una parte relativamente pequeña de un ordenador. Esto es lógico, ya que es necesario definir los aspectos relevantes del sistema que el benchmark usa como referencia: Cuantos más elementos del sistema pretendan examinarse, más datos será necesario especificar. Esto implica que aumentarán las diferencias entre el ordenador de referencia y aquellos que se estudien. Este aumento de diferencias puede restar validez a los resultados del análisis, o como mínimo aumentar las discusiones sobre las conclusiones a las que lleve dicho análisis.

Si se considera un ordenador completo es necesario determinar un modelo de las piezas de hardware, estado de los sistemas de memoria e incluso programas instalados, ya que un antivirus o un cortafuegos podrían afectar a mediciones del rendimiento de red, por ejemplo.

b) Resultados que requieren interpretación

Dado que cada benchmark está diseñado según ciertos objetivos, los resultados que cada uno devuelva deberán ser interpretados con arreglo a esos objetivos. Si el código de un benchmark trata de estimar la velocidad del procesador en operaciones matemáticas, pero da más importancia a las operaciones en punto flotante, será necesario tenerlo en cuenta a la hora de interpretar los resultados.

c) Dependencia de compiladores y avances tecnológicos

Para contribuir a agravar el punto anterior, el código objeto que generan los benchmark puede depender del compilador que se utilice, y el comportamiento de ese código depender de los avances tecnológicos.

El ejemplo más claro se tiene en el Dhrystone, pero en realidad es un problema que afecta a todos, ya que el uso de una librería de matemáticas optimizada, o la presencia en el procesador de instrucciones orientadas a aritmética vectorial, etc, pueden afectar al resultado del benchmark.

d) Evaluación a través de la comparación

Los benchmarks encontrados en la investigación realizada evalúan el rendimiento de los elementos funcionales analizados en comparación con una referencia en particular. Esto implica que debe definirse correcta y completamente dicha referencia, lo cual impone una limitación al alcance del análisis como se veía en (a).

Asimismo, el hecho de partir de un elemento de referencia permite una mayor discutibilidad de los resultados de los benchmarks. Como se ha visto en el ejemplo del iComp, la elección de los parámetros de referencia puede responder, o ser sospechoso de responder, a intenciones publicitarias. Además, los resultados de distintos benchmarks deben compararse entre sí a través de la comparación entre sus sistemas de referencia, lo cual se haría mediante otros benchmarks y por tanto aumentaría la discutibilidad de la comparación, pues se consideraría la falta de precisión de los resultados originales con la de la comparación entre los parámetros de referencia.

5. Diseño de benchmarks

No hay suficientes benchmark como para evaluar el rendimiento de cada elemento funcional de un ordenador. Por tanto, es posible que haya que diseñar programas propios de evaluación. Una vez decidido el rendimiento de qué elemento funcional debe ser evaluado, los dos rasgos más importantes a considerar serían los siguientes:

a) Cómo evaluar el elemento funcional

Lo primero que hay que realizar es un estudio sobre el funcionamiento del elemento funcional para saber qué operaciones realiza y, a ser posible, cómo. De este modo se averigua qué servicios debe solicitar el programa de benchmarking que se pretende diseñar.

b) Cómo cargar adecuadamente el elemento funcional

Hay que ser consciente de las mejoras específicas que tienen los modelos, al menos los más populares, del tipo de elemento funcional que debe estudiarse. Entre estas mejoras se contarían circuitos o repertorios de órdenes específicas para determinadas funciones. Es natural pensar que si un modelo concreto es capaz de realizar ciertas operaciones mejor que otros, no por ello debe impedirse que el rendimiento de dichas operaciones se tenga en cuenta al evaluar el modelo. Sin embargo, si es algo que hay que tener en cuenta a la hora de interpretar los resultados: Si un modelo tiene un rendimiento mejor en un 50% con un grupo concreto de

funciones, pero estas funciones sólo se presentan en el uso real del ordenador un 1% del tiempo, es necesario tenerlo en cuenta.

c) Cómo no cargar otros elementos funcionales

Como en otros aspectos de la informática, es importante que el programa haga aquello para lo que se diseña, pero también que no más que eso. En realidad, en el diseño de benchmarking es especialmente importante, ya que si el programa actúa sobre elementos funcionales distintos al de aquel para el que ha sido diseñado, el rendimiento de esos otros elementos influye sobre los resultados.

Otro motivo para evitar la carga de otros elementos funcionales se encuentra en la posibilidad de evaluación combinada de elementos. Es decir, en analizar los efectos sobre el sistema de la carga de varios de sus elementos funcionales en lugar de sólo uno de ellos.

5. Evaluación y conclusiones del trabajo

1. Organización de los datos de rendimiento

1.1 Organización de los datos de rendimiento en Microsoft Windows

El sistema de organización de datos de rendimiento que utiliza Windows muestra varias características que pueden hacer dudar de su optimalidad.

Las aplicaciones que Windows incluye en sus distribuciones para analizar el rendimiento del ordenador no parecen estar siendo ofrecidas al usuario corriente, sino casi estar siendo ocultas en espera de alguien que ya sepa dónde debe buscar lo que quiere encontrar. Cualquier usuario de Windows no tiene más que abrir el menú Inicio para encontrar sus programas, pero no únicamente esto, sino también herramientas que actúan sobre el funcionamiento del sistema operativo como el defragmentador de disco, el scandisk, asistentes para eliminar datos inútiles... Sin embargo, cuando se trata de analizar el rendimiento del equipo por parte del usuario, ¿Dónde están esas solícitas utilidades, que parecen tan dispuestas a ayudar? Los equipos de políticas de Microsoft parecen considerar que el usuario no tiene como tarea común saber cómo funciona su ordenador. Sin embargo, si parece que sea una tarea común arreglar problemas de su sistema, como poco espacio libre en disco o la fragmentación excesiva.

Si se toma la defragmentación como ejemplo particular, se encuentra otro dato del que merece la pena hablar. ¿Dónde se informa al usuario común de los problemas de la fragmentación de disco? Es casi como si regalaran un taller de reparación de coches a cada persona que comprase un coche. Los usuarios pueden arreglar los problemas de sus coches. Siempre y cuando hubiera mecánicos asignados al taller, o información de contacto con mecánicos, o siquiera libros de instrucciones claros. Asimismo un usuario actual de Microsoft Windows puede arreglar algunos problemas de su ordenador, o podría, si tuviera los conocimientos necesarios. ¿Qué utilidad tiene un taller para alguien que no es mecánico? ¿Qué utilidad tienen aplicaciones informáticas para los que no saben cómo o porqué utilizarlas? Algunos podrían pensar que no es más que una táctica de empresa para proyectar en sus clientes una imagen de compañía amable y dispuesta a ayudar, cuando en realidad no se ayuda tanto como parece.

Lo mismo ocurre con las aplicaciones de análisis de funcionamiento del sistema. Peor aún. Es como darle al conductor un taller, pero no decirle dónde está y ni siquiera que lo tiene. No hay accesos tan directos y fáciles de encontrar a las aplicaciones de análisis, y por supuesto, la información sobre porqué utilizarlas es más difícil de encontrar que el porqué alguien querría defragmentar el disco duro de su ordenador. Al mismo tiempo, aunque encuentre las aplicaciones de análisis y el porqué querría usarlas, la información para interpretar los resultados requiere algo más de búsqueda e investigación. Sin esta información, los resultados del análisis sólo son una colección de números que no proporcionan información real.

¿Por qué dificultar tanto el acceso al análisis del ordenador? Es obvio que, comercialmente, debe mantenerse un cierto nivel de secreto sobre la implementación, pero en ocasiones, al estudiar el sistema operativo Windows, se percibe más una superprotección que una protección razonable. Y sólo en cuanto al alcance del estudio realizado en este trabajo.

Supóngase que el usuario ya sabe que puede analizar el funcionamiento de su ordenador, e incluso cómo interpretar los datos del análisis para que realmente cuente con información sobre el funcionamiento de su equipo. Supóngase que también sabe qué datos de rendimiento examinar y con que frecuencia tomar muestras. Aún así, todavía tiene que localizar esos datos de rendimiento, que en Windows equivale a localizar los contadores de rendimiento correspondientes. Sin embargo, la organización actual de los contadores de rendimiento no es tan homogénea como podría, lo que dificulta la comprensión de la estructura, y por tanto la localización de los contadores de rendimiento. Esto lleva a pensar que facilitar el análisis del rendimiento del ordenador no es una prioridad para los equipos de políticas de Microsoft, ni para usuarios comunes ni para los propios administradores. Es posible que debido a esa falta de prioridad haya sido tan, aparentemente, poco planificada la organización, acceso y localización de contadores de rendimiento. Esta organización parece haber sido implementada primero, y luego explicada mientras se improvisaban sus ampliaciones. Como hacer los planos después de la casa.

En la documentación sobre la organización de los contadores aparecen definiciones poco claras, imprecisas en ocasiones, e incluso conceptos que varían de significado según ciertas circunstancias, aunque dentro del mismo contexto. La estructura, como ya se ha dicho, tampoco es homogénea. Todo ello son rasgos que dificultan la comprensión de una metodología de clasificación. No sólo eso, al resultar imprecisa, dificulta la inclusión de nuevos elementos del dominio, como nuevos contadores de rendimiento. Es decir, la falta de rigor en la planificación de una organización lleva a dificultades, tanto en el uso, como en la implementación, mejora y complementación de dicha organización.

Debido a los rasgos explicados en los párrafos anteriores, se puede llegar a pensar que la documentación publicada sobre los contadores de rendimiento no es la que utilizan los desarrolladores en Microsoft, debido a la falta de rigor científico y precisión. Eso supone un aumento del nivel de secretismo sobre sus productos, un nivel que ya parecía superar lo lógico. Ahora bien, si se tiene en cuenta que la documentación publicada está, al menos en teoría, orientada a desarrolladores; y más importante aún, que parece corresponder con la realidad de la organización, surge otra posible explicación, probablemente peor que la anterior: los desarrolladores de Microsoft no siempre planifican sus diseños tan exhaustivamente como sería científicamente deseable.

Por último, hay otro aspecto de Microsoft que se deduce de la investigación realizada para este trabajo. A pesar de que, a diferencia de otros sistemas operativos, Microsoft tiene la capacidad de establecer estándares y controles sobre el software que se desarrolla para sus sistemas operativos, existe una considerable libertad para añadir (o eliminar el acceso a) contadores de rendimiento de un ordenador que funcione bajo Microsoft Windows. Aunque requiere de ciertos conocimientos y cierta investigación, el hecho de que una empresa, que se ha probado tan secretista, ceda tales libertades a sus usuarios resulta inconsistente y contradictorio con otras muestras de su política.

1.2 Organización de los datos de rendimiento en Linux

La organización de los datos de rendimiento en Linux ni siquiera es propiamente una organización de los datos de rendimiento, sino que parece más una organización de

datos del sistema, ya que pueden encontrarse datos de rendimiento, de configuración, etc. Por supuesto, este tipo de organización dificulta el acceso y uso de los datos de rendimiento para analizar el funcionamiento del sistema operativo. Sin embargo, a diferencia del caso de Windows, es sencillo encontrar una explicación satisfactoria para este hecho: Linux no estandariza su código, no al menos tan férreamente como Microsoft. Es cierto que existe un núcleo "oficial", pero los usuarios tienen libertad para replanificar y rehacer los núcleos que utilizan, así como distribuirlos. Tal cantidad de libertad hace imposible que alguien pueda obligar a los usuarios de Linux a utilizar una determinada organización para los datos de rendimiento.

Quizá tendría cierto interés reflexionar sobre la posibilidad de un equilibrio entre el superproteccionismo de los sistemas operativos Microsoft y la libertad de modificación del sistema operativo Linux. Quizá que los cambios en el núcleo sean enviados a un equipo de supervisión y adaptación que tuvieran control sobre el núcleo "estándar" de Linux. Esto elevaría la importancia de este equipo y disminuiría la libertad de los usuarios, pero también es cierto que facilitaría el análisis de los datos de rendimiento, que es lo que se estudia en este trabajo.

1.3 Una única organización

En este trabajo se propone un modelo de taxonomía basado en ideas intuitivas sobre los datos de rendimiento. Estas ideas parten de la definición utilizada de los datos de rendimiento: mediciones que pueden realizarse en un computador y proporcionar información acerca del funcionamiento y aprovechamiento de sus elementos funcionales; entendiendo como "elemento funcional" un elemento que realiza una función, independientemente de si es un componente hardware, software, o combinación de ambos.

La taxonomía se basa en las propias definiciones de dato de rendimiento y elemento funcional, y está orientada a una metodología de localización de los datos de rendimiento a través de conceptos intuitivos.

La taxonomía propuesta podría resumirse en las siguiente ideas:

· Un ordenador está conectado a otros ordenadores
· Un ordenador tiene varios tipos de elementos funcionales
· Un elemento funcional tiene varios representantes en un ordenador
· Sobre el funcionamiento de cada elemento funcional pueden tomarse mediciones.

Esta taxonomía resulta más clara y fácil de entender que la actual, tanto en Windows como en Linux. Su uso facilitaría el acceso a los datos de rendimiento y, por tanto, su utilización.

La utilización de la taxonomía propuesta exigiría hacer cambios en los sistemas operativos actuales, algo que no parece posible en el caso de Linux y que puede considerarse improbable en el caso de Windows, teniendo en cuenta la prioridad aparente del tema que ocupa este trabajo. Sin embargo, ¿no hay otros motivos para revisar el diseño de los sistemas operativos actuales? ¿No habría otros aspectos que se beneficiarían de tal revisión? Como mínimo, este trabajo aporta nuevos motivos en favor de esa revisión.

A pesar de lo explicado en el párrafo anterior, es necesario tener en cuenta que esos cambios en los sistemas operativos puede que no se hagan, al menos a corto plazo o quizá nunca. Por ello se ha buscado otro método para facilitar el uso de los datos de rendimiento: una estructura que recubra las organizaciones actuales, y que permita a los usuarios localizar datos de rendimiento según sus necesidades específicas. ¿Cómo?

Partiendo de una idea simple: “La optimalidad de una organización depende del objetivo que se pretenda con dicha organización.” La estructura propuesta se basa entonces en permitir el uso, diseño e inclusión de varios criterios y métodos de localización, en lugar de escoger uno sólo como si fuera la solución a todas las necesidades. La estructura no afecta a los sistemas operativos, por lo que su utilización no está condicionada al cambio de éstos.

2. Utilización de los datos de rendimiento

2.1 ¿Para qué utilizar los datos de rendimiento?

Hay varias razones para querer usar estos datos y la información que pueda extraerse de ellos. A continuación se indican algunos de estos usos, desde los más simples y cercanos al uso común actual, hasta los más lejanos y teóricos.

La información puede utilizarse para localizar “cuellos de botella” y determinar así a dónde deben dirigirse los recursos para mejorar el sistema. Por ejemplo, dónde invertir el dinero necesario para la sustitución de un elemento funcional por otro mejor.

Se puede extraer información sobre el funcionamiento de una aplicación, servicio, controlador, elemento funcional, etc. y utilizarla para ajustar parámetros de configuración o detectar comportamientos no esperados. Por supuesto, pueden compararse los valores de los datos de rendimiento en distintas circunstancias, para estudiar cómo influyen éstas en el funcionamiento de una parte o la totalidad del sistema.

Un desarrollador de software puede preparar sus aplicaciones para que se autoconfiguren según el valor de los datos del rendimiento del computador en el que se ejecuta. Esta autoconfiguración podría perseguir el objetivo que prefiriera el desarrollador, o disponerse de varios objetivos entre los que pudiera escoger el usuario. Por ejemplo, considérese un gestor de descarga. Muchos disponen de opciones para limitar la cantidad de ancho de banda que consumen. De ese modo los usuarios pueden ajustarlo para permitir el funcionamiento adecuado de otras aplicaciones como los navegadores de internet. Esto, sin embargo, implica que el usuario debe saber cuáles son las capacidades de su ordenador y cómo configurarlo, un conocimiento que no puede suponerse. Sin embargo, el desarrollador es de esperar que sepa cómo configurar una aplicación así, y del mismo modo puede preparar su aplicación para que utilice sólo el ancho de banda que no está utilizando otra aplicación, y que muestree el estado de uso de la conexión cada cierto tiempo para mantener la configuración adecuada, de modo que el usuario pueda navegar por internet sin notar retardos debidos al gestor de descarga, y éste use el máximo del ancho de banda que haya disponible para acelerar las descargas. En resumen, mejorar el rendimiento global desde el punto de vista del usuario.

A través del análisis de los datos de rendimiento pueden detectarse situaciones de conflicto por los recursos, ese es un uso actual. Pero, ¿pueden preveerse esas situaciones? Responder a esta idea requiere más investigación, pero no resulta descabellado pensar que así es, ya que de un modo parecido se actúa en muchas redes informáticas para preveer problemas de conexión.

Cuando se unen las ideas de aplicaciones autoconfiguradas y predicción de situaciones de conflicto, nace la idea de sistemas autoconfigurables. Al igual que los sistemas de control de redes usan ciertas mediciones sobre sí mismas para ajustar su funcionamiento, de modo que se asegure la comunicación, ¿por qué no podrían hacerlos

los computadores? Resulta interesante imaginar la posibilidad de ordenadores capaces de autoconfigurarse para satisfacer los criterios de optimalidad de sus responsables. Estos criterios podrían ser disminuir el tiempo de respuesta para un usuario, aumentar la productividad para un administrador, etc.

Esta última idea podría llevar al diseño de sistemas capaces de contener y aplicar, al menos parcialmente, el conocimiento del campo de administración de sistemas que posee un ser humano. Además, los sistemas resultantes no obligarían a los usuarios a poseer el conocimiento necesario para configurarlos correctamente, pero siempre estarían adecuadamente configurados respecto a las necesidades de aquellos.

Llegados a semejante punto, ¿por qué no aumentar la escala? Actuar de forma modular es una herramienta muy utilizada en el campo de la informática. Se usan piezas pequeñas para crear otras mayores, y éstas otras mayores, etc. ¿Por qué detenerse en la administración de un ordenador? ¿Y todos los ordenadores de una red? ¿Y combinaciones de redes?

Es obvio que el estudio de los datos de rendimiento ofrece incentivos actuales, pero también abre la posibilidad de futuros que pueden tener gran interés.

2.2 Problemas en el uso de los datos de rendimiento

Existen varias dificultades que es necesario estudiar en las investigaciones sobre el uso de los datos de rendimiento. Los requisitos de tiempo y espacio necesarios para el muestreo de los datos de rendimiento y su almacenamiento para análisis posteriores son, sin lugar a dudas, límites sobre los estudios que puedan realizarse. Existen formas para disminuir el impacto de tales requisitos, pero probablemente jamás puedan anularse por completo estos límites. Esto significa que siempre habrá cierto grado de imprecisión en los datos que se midan, ya que el hecho de obtener estas mediciones estará variando el sistema que se mide, y por tanto, podrá variar los valores obtenidos, con respecto a los que serían si no se estuviera midiendo.

Estas dificultades no impiden ni mucho menos la investigación, sólo imponen ciertos matices en la interpretación de resultados y crean cierta necesidad de trabajos y estudios previos.

Otra dificultad estriba en las diferentes plataformas de funcionamiento de los sistemas informáticos actuales. Esta dificultad es, posiblemente, salvable por completo, a diferencia de las explicadas anteriormente. Sería útil un modo de acceso a los datos de rendimiento que sea independiente de la plataforma, especialmente para la posibilidad de sistemas autoadministrados. La solución ideal para administrar en la misma red varios computadores no es necesariamente el uso del mismo sistema operativo para todas las máquinas conectadas, sino que bastaría con el uso de una estandarización adecuada de la información de rendimiento o, al menos, de la existencia de estructuras independientes de plataforma, como la que se propone en el capítulo 3 de este trabajo.

3. Evaluación del rendimiento

Actualmente los estudios sobre el rendimiento de los ordenadores se realiza a través de metodologías comparativas. Estos análisis poseen importancia, más aún cuando han llevado a avances importantes en la informática. Los adelantos en procesadores, por ejemplo, han sido impulsados precisamente por la capacidad de comparar unos con otros y el deseo de mejorarlos. El problema, sin embargo, es que un computador no consiste sólo en sus procesadores. Por supuesto, pueden diseñarse

pruebas comparativas para analizar otros elementos funcionales e impulsar del mismo modo avances en las tecnologías de éstos. Sin embargo, a pesar de toda la modularización presente en los ordenadores, lo cierto es que son elementos que trabajan en conjunto y, por tanto, que influyen unos en otros. La influencia más clara está en los cuellos de botella. Un disco duro puede enviar decenas de megabytes de información por segundo, y un procesador puede ser capaz de procesar toda esa información, pero ¿qué ocurre cuando el bus que los conecta tiene una capacidad de siete megabytes por segundo? Puede haber otras influencias menos claras, ¿cómo detectarlas y estudiarlas?

Los métodos de análisis comparativo requieren de la definición adecuada de un parámetro de comparación. Si este parámetro fuera un ordenador completo, implica muchísimas variables cuyos valores es necesario establecer. Cuantas más variables haya, más podrán diferenciarse los sistemas estudiados del sistema de referencia, y esto disminuye la precisión de los resultados.

Por otro lado, ha habido ocasiones en las que los métodos de evaluación comparativa han permitido maniobras y diseños comerciales, orientados específicamente a mejorar los resultados de un elemento respecto a sus competidores.

Ahora considérese la posibilidad de análisis cuantitativos. Suponen otra nueva dimensión en cuanto a los análisis de rendimiento. Lo más importante es que no necesitan de la definición de parámetros de referencia como los métodos actuales, lo que elimina los problemas de ésta necesidad. Es decir, permitiría mediciones de la totalidad del ordenador y no únicamente de aspectos concretos del mismo. Este es el resultado principal de este trabajo.

6. Evaluación y líneas futuras

Este trabajo puede evaluarse positivamente porque los objetivos descritos en la introducción han sido alcanzados.

Este proyecto provee de infraestructura suficiente para analizar los datos de rendimiento de sistemas informáticos, haciendo necesaria una investigación continuada a partir de los resultados obtenidos. Las líneas de trabajo que se proponen no pretenden agotar las posibilidades de investigación y desarrollo, sino sólo establecer las direcciones más naturales, desde el punto de vista de los autores, que podrían seguirse en el futuro.

1. Líneas de investigación

1.1 Estudio sobre la futura utilización de los resultados del proyecto

Este trabajo proporciona medios para realizar análisis cuantitativos del rendimiento de sistemas informáticos, así como facilitar el uso de los datos de rendimiento presentes en los sistemas operativos Microsoft Windows y Linux. No obstante, se debería seguir investigando con el fin de realizar las siguientes actividades:

Ampliar los puntos de vista sobre el análisis de rendimiento. Para esto sería necesario entrevistar a profesionales que trabajen de forma especializada con análisis del rendimiento, buscando obtener guías acerca de la mejora o creación de los criterios de selección de datos de rendimiento.

Estas entrevistas también permitirían **confirmar las necesidades de los usuarios** tanto como encontrar necesidades no previstas en este trabajo.

Una vez especificadas tales necesidades debería utilizarse la información obtenida para **verificar** la existencia de necesidades no solventables con los métodos actuales así como **validar** de forma exhaustiva los métodos propuestos en este trabajo.

El estudio que se propone en este punto se llevaría a cabo en fases claramente diferenciadas:

a) Selección de la población objetivo: Es necesario delimitar la población objetivo para definir adecuadamente sus características, como por ejemplo su nivel de conocimientos, su actividad profesional, la naturaleza de su interés (profesional o personal), etc.

b) Preparación de las entrevistas: Éstas deben adecuarse a la población estudiada, siguiendo la definición de las características que se haya especificado en la fase anterior. Al mismo tiempo, la evaluación de las respuestas a las entrevistas debe ser lo más fácilmente realizable, en lo que influye la preparación de las preguntas, etc.

c) Realización de las entrevistas: Una vez disponible las entrevistas deben realizarse sobre una cantidad suficientemente representativa de la población seleccionada.

d) Evaluación de los resultados: Con las respuestas a las entrevistas debe extraerse información acerca de puntos de vista sobre el análisis del rendimiento que no se hayan tenido en cuenta en el presente trabajo. Al mismo tiempo, debe realizarse una extracción de necesidades presentes en la población objetivo cuyo tratamiento no haya sido realizado ya.

e) Confirmación de las necesidades: A partir de los datos sobre puntos de vista y necesidades no contempladas es necesaria una segunda serie de entrevistas para

determinar la generalidad de tales necesidades y puntos de vista, de modo que puedan seleccionarse los más interesantes en cuanto a la proporción de población coincidente en su valoración.

f) Validar los métodos que se presentan: Los resultados de este trabajo no deberían tomarse como dogmáticos en cuanto a que una mentalidad científica debería siempre permitir una revisión de resultados anteriores. Al fin y al cabo, resultados más aceptados por la comunidad científica se han puesto en duda con anterioridad, y esto ha llevado a nuevas teorías y descubrimientos. Así pues, una vez que se sepa qué necesidades existen en la población objetivo y qué puntos de vista les resultan interesantes, debe estudiarse la validez de los métodos presentados en este trabajo para asegurar la satisfacción de las necesidades y la utilización de los puntos de vista.

1.2 Selección de los contadores de rendimiento

En el trabajo se explican ciertos hechos de interés:

· La cantidad de contadores de rendimiento es existentes es bastante grande.

· Pueden encontrarse contadores de rendimiento distintos que hacen referencia al mismo dato de rendimiento

· Existen problemas en la obtención de datos de rendimiento. Estos problemas son más graves cuanto mayor sea la cantidad de contadores de rendimiento que se monitorizan.

Los tres hechos se combinan para establecer que los problemas de monitorización del rendimiento tienen un impacto potencialmente grande.

Debido a estas ideas se considera de interés estudiar exhaustivamente cómo debe realizarse una selección de los contadores de rendimiento a monitorizar, de modo que se maximice la información obtenida y se minimice el impacto de los problemas generados. Para estudiar esta selección se atravesarían varias fases:

a) Estudio de los contadores de rendimiento y los datos que contienen: Conociendo el significado de cada dato de rendimiento puede determinarse que no es necesaria su monitorización. Por ejemplo, en un determinado análisis puede no añadir información el monitorizar los contadores del objeto de rendimiento “Objetos de WMI”. Por tanto, no deberían ser considerados como contadores de rendimiento a examinar. Para tomar estas decisiones se debe estudiar cada dato de rendimiento lo que, debido a las limitaciones de tiempo, se encuentra fuera del alcance de este trabajo.

b) Encontrar contadores que hagan referencia al mismo dato: En el mismo estudio que se realiza en la fase (b) se obtendrá información suficiente para determinar si existen grupos de contadores de rendimiento que accedan al mismo dato de rendimiento. Obviamente, no añade información el monitorizar más de uno de ellos, pero sí que agrava la problemática mencionada. Por tanto, conocer qué contadores de rendimiento hacen referencia al mismo dato es de utilidad para la selección. Por ejemplo, en los dos objetos de rendimiento “Sistema” y “Objetos” existen los contadores “Procesos” y “Subproceso o subproceso”, y hacen referencia al mismo dato de rendimiento.

c) Encontrar datos de rendimiento cuyos valores puedan deducirse de otros: En la fase anterior se ve que existen contadores distintos que acceden al mismo valor. De ahí surge la pregunta: ¿Podría el valor de un contador de rendimiento deducirse de los valores de otros? Para responder a esto se realizarían análisis del comportamiento de los contadores de rendimiento en situaciones controladas para estudiar las relaciones entre sus valores, aplicando técnicas de datamining. De este modo podrían obtenerse reglas que permitieran, como mínimo, estimar el valor de un contador de rendimiento a partir

del valor de otros. Si esta estimación es lo bastante buena para el uso que se le quiera dar, podría monitorizarse un conjunto menor de contadores de rendimiento.

2. Líneas de desarrollo

2.1 Fiabilidad de los contadores

Este punto ha sido estudiado en este trabajo, pero merece un análisis mucho más profundo que incluya todos los elementos funcionales y posteriormente combinaciones de éstos. Tal profundidad está fuera del alcance de este trabajo, superando el tiempo asignado a la asignatura Sistemas Informáticos por el plan de estudios.

Hay que continuar el trabajo iniciado aquí resolviendo, para todo un sistema y cada una de sus partes, dos cuestiones:

· ¿Los valores de los datos de rendimiento son los que en teoría deberían ser?
· ¿Cuál es la precisión de los valores de los datos de rendimiento?

Estas cuestiones se responden en tres fases:

a) Análisis del elemento funcional: Se requiere conocer la teoría del funcionamiento de un elemento funcional para poder diseñar el código encargado de actuar sobre sus recursos. Se debe estudiar qué código o método debe utilizarse para observar cada una de las capacidades del elemento funcional concreto. Por ejemplo, para evaluar el rendimiento de un procesador se deberá estudiar qué código provoca uso de la unidad aritmético-lógica, de la gestión de saltos, de la unidad de punto flotante, etc.

b) Preparación de bancos de pruebas: Una vez estudiado suficientemente el funcionamiento de un elemento funcional deben implementarse medios para realizar las pruebas. En este trabajo se considera que debería utilizarse código compilado y no estímulos hardware artificiales, ya que el objetivo es conocer la fiabilidad de los contadores de rendimiento ante la ejecución de un código determinado. El código obtenido deberá analizarse para saber qué grado de uso de los recursos debe esperarse.

En el anexo 4.1 se ofrece una estructura diseñada para estudiar la ejecución de semejantes bancos de pruebas.

c) Análisis de los resultados: Tras la ejecución de los bancos de pruebas se debe determinar si los resultados obtenidos han sido los esperados. En caso contrario deben analizarse las diferencias para averiguar a qué pueden deberse: errores en el análisis teórico del elemento funcional o el código, fallos en el sistema de medición, falta de precisión de los contadores de rendimiento, interferencia por parte de otros procesos, interferencias por actividad del sistema operativo, etc.

2.2 Recubrimiento de la organización actual

Esta línea de desarrollo sería más fácil de seguir tras la realización de la investigación descrita en el punto 1.1, ya que entonces se conocerán las necesidades y criterios de búsqueda de datos de rendimiento que realmente son necesarios para una población determinada.

En caso de que los métodos propuestos en este trabajo hayan sido validados adecuadamente para satisfacer esas necesidades, podría utilizarse la estructura ofrecida en el capítulo 3 para implementar cada criterio de búsqueda y sumarlo a las

posibilidades actuales de localización de datos de rendimiento. Cada inclusión debería haber pasado por dos fases:

a) Planificación detallada de la implementación y especificación exhaustiva

b) Comprobación de la satisfacción real de las especificaciones

2.3 Posibilidades predictivas de los contadores

Tras realizar la investigación propuesta en el punto 1.2 y el desarrollo explicado en el punto 2.1 sería más fácil realizar la propuesta en esta sección 2.3. Resulta de interés poder predecir en qué condiciones pueden aparecer situaciones de conflicto por los recursos de un sistema informático. Al igual que en el caso de las redes se realizan ciertas mediciones para asegurar el funcionamiento de la red, en el caso de un sistema informático puede resultar de interés la posibilidad de seleccionar un conjunto de mediciones suficiente para asegurar su correcto funcionamiento. Para cumplir este objetivo se deberán atravesar las siguientes fases:

a) Definir con precisión qué se considera una situación de conflicto: Lo primero es, naturalmente, entender qué es lo que se desea evitar. Basándose en esta definición, se realizarían unas primeras propuestas de definición de situaciones de riesgo alto, medio, bajo, etc. Con estas definiciones y las reglas lógicas que se obtengan en los siguientes puntos, un sistema informático podría programarse para actuar de determinado modo según el riesgo de conflicto.

b) Desarrollo de métodos sistemáticos para producir situaciones de conflicto: Una vez definido qué quiere predecirse, deberán desarrollarse los medios para provocar, de modo controlado, las situaciones de conflicto. Durante estas situaciones deberá monitorizarse un conjunto de datos de rendimiento adecuadamente seleccionado. Los bancos de pruebas desarrollados en el punto 2.1 podrían ser de utilidad como punto de partida para el desarrollo de esta fase.

c) Análisis de los resultados: A través del análisis de los datos de rendimiento monitorizados en el punto anterior, podrían determinarse reglas que indiquen el riesgo de un estado determinado del ordenador. A través de estas reglas, el propio sistema podría ser capaz de percibir el nivel de competición por sus recursos y tomar las medidas necesarias para evitar que llegue a producirse una situación de conflicto que lleve al bloqueo del computador.

2.4 Estudio de la factibilidad de sistemas autoadministrados

Este desarrollo se propone como continuación del anterior, ya que hará uso de los resultados de aquel. El objetivo es probar las reglas de predicción obtenidas en el desarrollo del punto 2.3, y estudiar qué métodos son apropiados para evaluar esas reglas y actuar sobre el sistema. Asimismo, se estudiaría la propia factibilidad de un sistema autoadministrado, o si es necesario realizar más avances para que sea posible su realización.

Este desarrollo seguiría las siguientes fases:

a) Determinar los objetivos de la administración: La correcta administración de un sistema depende de los objetivos de las personas responsables. A un usuario le importará más disminuir los tiempos de respuesta; a un administrador le puede preocupar más aumentar la productividad; a un gestor financiero le importará que el intervalo de tiempo entre tareas acabadas sea el mínimo posible para poder rentabilizar

antes los resultados, aunque se acaben las tareas una por una... Estos objetivos deben ser tenidos en cuenta a la hora de diseñar un sistema autoadministrado.

b) Diseño de aplicación de monitorización-administración: Este diseño buscaría los siguientes objetivos:

· Monitorizar el conjunto adecuado de datos de rendimiento (ver propuesta de investigación 1.2)

· Generar conclusiones sobre el estado, basándose en las reglas obtenidas en la propuesta de desarrollo 2.3

· Generar respuestas de prevención o reparación de las situaciones no deseadas. Estas respuestas deberían registrarse para permitir su examen por parte de supervisores humanos.

· Opcionalmente, la aplicación sería capaz de poner en práctica sus decisiones

c) Experimentación con la aplicación desarrollada: Para una correcta evaluación deberían utilizarse tanto métodos que lleven demostradamente a situaciones de conflicto, como las desarrolladas en la fase (b) del punto 2.3, como con métodos que intenten "engañar" a la aplicación, de modo que no lleven a situaciones de conflicto pero puedan hacerle creer que sí lo harán. Así mismo, debería experimentarse con una cantidad adecuada de métodos que no lleven a situaciones de conflicto ni lo parezca.

d) Evaluación de los experimentos: Esta fase es doble. Por un lado, los propios desarrolladores de la propuesta podrían evaluar si la aplicación ha podido clasificar adecuadamente los métodos provocadores de conflicto, los engañosos y los inofensivos. Por otro lado, se podría entrevistar a expertos que examinasen las situaciones a las que ha respondido la aplicación y evaluasen su reacción.

e) Calibración de la aplicación o conclusiones sobre la factibilidad: La modificación de la sensibilidad de la aplicación, entre otros factores, puede afectar al comportamiento de ésta, y cada intento sería seguido de la repetición de las fases (c) y (d). Se debería registrar para cada configuración los resultados de la evaluación, de modo que se pueda buscar un equilibrio entre reacciones acertadas y erróneas del sistema. Finalmente, el estudio y desarrollo debería permitir estimar la factibilidad de sistemas autoadministrados con las tecnologías disponibles.

ANEXO 1.1:

Web-Based Enterprise Management (WBEM) y Common Information Model (CIM)

1. Web-Based Enterprise Management (WBEM)

WBEM (Web-Based Enterprise Management: Gestión Empresarial Basada en Red) es un conjunto de tecnologías que buscan definir un estándar para unificar la gestión de sistemas de computación distribuida. WBEM provee a la industria de la capacidad para entregar un conjunto bien integrado de herramientas estándar de gestión, facilitando el intercambio de datos entre plataformas y tecnologías distintas. [DTMF1]

El conjunto central de estándares que forman WBEM incluyen:

a) Modelo Común de Información (CIM, Common Information Model)

El modelo de datos para WBEM, CIM provee de una definición común de información de gestión para sistemas informáticos, redes, aplicaciones y servicios, y permite extensiones por parte de los vendedores. [DTMF1]

Hay disponible más información sobre CIM en este anexo.

b) WBEM Discovery y WBEM Universal Resource Identifier

Dos estándares que proveen a las aplicaciones de un modo de identificar e interactuar con sistemas gestionados a través de WBEM, centrándose en estándares y protocolos existentes para permitir un rápido desarrollo y despliegue de soluciones de gestión. [DTMF1]

c) Lenguaje de Consulta de CIM

Un lenguaje de consulta utilizado para extraer datos de una infraestructura de gestión basada en CIM. [DTMF1]

2. Common Information Model (CIM)

CIM provee de una definición de información de gestión para sistemas, redes, aplicaciones y servicios, mientras permite extensiones por parte de otros fabricantes. Las definiciones de CIM permiten a los usuarios intercambiar información de gestión entre sistemas distintos a través de red. [DTMF1]

CIM se compone de una Especificación y un Esquema. El esquema incluye las descripciones reales de los modelos, mientras que la Especificación define los detalles necesarios para la integración con otros modelos de gestión. [DTMF1]

El modelo CIM está orientado a objetos: utiliza clases como las de un lenguaje informático para representar la información. En particular, se desarrollan clases que representen discos duros, aplicaciones, routers e incluso tecnologías particulares

definidas por los usuarios como el sistema de climatización inteligente de un edificio. Así como las clases representan tipos de elementos dentro del sistema, se hace uso de la herencia para crear subclases que representen tipos, modelos, etc específicos. Las instancias de estas clases representan a un elemento particular del sistema gestionado. [MSDN1]

A través de la consulta de datos en instancias de las clases CIM es como se obtiene información sobre los elementos reales que representan dichas instancias. Asimismo, actuando sobre dichas instancias pueden controlarse las funciones del elemento particular representado. [MSDN1]

CIM usa tres niveles de herencia para describir tanto aspectos generales como específicos del sistema gestionado. También utiliza una técnica denominada "asociación" para enlazar diferentes partes del modelo de la empresa y usa esquemas para distinguir distintos medios de gestión. Los tres niveles de clases que define CIM son: [MSDN1]

a) Clases del núcleo
b) Clases comunes
c) Clases extendidas

a) Clases del núcleo:
Las clases del núcleo representan objetos gestionados que se aplican a todas las áreas de gestión. Estas clases proporcionan un vocabulario básico para analizar y describir sistemas gestionados. Por ejemplo, las clases __Parameters y __SystemSecurity son ejemplos de clases del núcleo. __Parameters define los parámetros de entrada y salida para métodos. También se usa para pasar valores de parámetros de entrada y salida entre un cliente de WMI y un proveedor de método. __SystemSecurity contiene métodos para acceder y modificar los parámetros de seguridad de un namespace.
[MSDN1]

b) Clases comunes:
Las clases comunes representan objetos gestionados que se aplican a áreas de gestión específicas. Sin embargo, son independientes de una implementación o tecnología particular. Las clases comunes son una extensión de las clases del núcleo. Un ejemplo de clase común sería CIM_UnitaryComputerSystem. Esta clase representa un sistema de computadoras de un único nodo, ya sea un ordenador de sobremesa, un ordenador portátil, un servidor, etc.
[MSDN1]

c) Clases extendidas:
Las clases extendidas representan objetos gestionados que son adiciones de tecnología específica a las clases comunes. Una clase extendida típicamente se aplica a una plataforma específica como UNIX o el ambiente Microsoft Win32. Un ejemplo de clase extendida es Win32_ComputerSystem, que representa un sistema de computadoras operando bajo una versión de Microsoft Windows.
[MSDN1]

WMI también soporta **asociaciones**. Una asociación es una relación entre dos o más clases WMI diferentes. Por ejemplo, una estación de trabajo en funcionamiento normalmente tiene un procesador. La clase Win32_ComputerSystemProcessor es una clase de asociación de WMI que asocia la clase de estación Win32_ComputerSystem con la clase de procesador Win32_Procesor. Sin embargo, una clase de asociación no

tiene que enlazar dos clases dependientes. De hecho, el objetivo principal de una clase de asociación es mostrar las relaciones entre clases que no son necesariamente dependientes unas de otras. [MSDN1]

Finalmente, WMI soporta el concepto de **esquema**. En el contexto de WMI un esquema es un grupo de clases que describen un ambiente de gestión particular. Por ejemplo, la plataforma SDK de Microsoft Windows usa dos esquemas: el esquema CIM y el esquema Win32. El primero contiene las definiciones para las clases del núcleo y comunes mientras que la segunda contiene las definiciones para las clases extendidas que son comunes dentro del ambiente Win32. Otros desarrolladores pueden crear sus propios esquemas, y dado que éstos son infinitamente extensibles, siempre pueden añadirse nuevas clases que describen nuevos objetos gestionados en un ambiente existente. [MSDN1]

Anexo 1.2:
Código desarrollado para la obtención de datos de rendimiento a través del Interfaz del Registro de Windows

1. Procedimiento de obtención de contadores de rendimiento a través del Interfaz del Registro:

Los registros del Registro de Windows tienen distintos tipos. En particular los que se utilizan para las claves que contienen información sobre los contadores de rendimiento son del tipo REG_MULTI_SZ (Multi-String, si una string es una cadena de caracteres terminada en null, multi-string es una cadena de strings terminada en null.)

Las rutas de acceso a los contadores de rendimiento están siempre en inglés y, si el ordenador usa otro idioma como predeterminado, en ese otro idioma. En particular, la entrada del Registro de Windows HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib\009 contiene las rutas en inglés y la HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib\00A las rutas en español (cuando éste idioma está instalado).

El proceso detallado para obtener las rutas en español sería como sigue:

Paso 1

- Usar la función RegOpenKeyEx () para abrir la entrada del Registro de Windows correspondiente a HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib
- Usar la función RegQueryValueEx para averiguar cuánto espacio se requiere, y reservar ese espacio. Es importante cerrar la entrada abierta en el punto anterior con RegCloseKey.
- Usar RegOpenKeyEx () para abrir la entrada HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib\00A y preparar el espacio necesario
- Usar RegQueryValueEx () para leer el valor Counter. Esta entrada contiene los nombres de objetos y contadores de rendimiento. El valor Help contiene textos explicativos de los objetos y contadores de rendimiento.

Paso 2

- Se usa RegQueryValueEx () para leer el valor Global en HKEY_PERFORMANCE_DATA y cargar así los datos sobre los contadores de rendimiento.
- Se va objeto por objeto buscando los nombres de instancia y de contadores de rendimiento y construyendo con ellos las posibles rutas.

3. Se usa la función RegQueryValueEx () consultando el valor “Global” dentro de la clave HKEY_PERFORMANCE_DATA. Con ello se obtiene una estructura del tipo PERF_DATA_BLOCK que describe el sistema y los datos de rendimiento.

Esta estructura es seguida por datos sobre los objetos de rendimiento, datos que incluyen una estructura de tipo PERF_OBJECT_TYPE por cada objeto. Los datos sobre los objetos pueden tener dos formatos, según se refieran a objetos con múltiples instancias o no: [WIN32]

- Los objetos con una **única instancia** tienen en sus datos una estructura PERF_OBJECT_TYPE que está seguida por una lista de estructuras del tipo PERF_COUNTER_DEFINITION, una por cada contador del objeto.
La estructura PERF_OBJECT_TYPE describe los datos de rendimiento del objeto. Tras la lista de estructuras PERF_COUNTER_DEFINITION hay una estructura del tipo PERF_COUNTER_BLOCK con los datos del contador correspondiente. [MSDN1]

- Los objetos con **múltiples instancias** también tienen en sus datos una estructura PERF_OBJECT_TYPE, y también ésta está seguida por una lista de estructuras PERF_COUNTER_DEFINITION, una por cada contador del objeto. La diferencia está en que en este caso, tras la lista de PERF_COUNTER_DEFINITION hay una lista de bloques de información de las instancias. Cada uno de estos bloques contiene una estructura PERF_INSTANCE_DEFINITION, el nombre de la instancia, y una estructura PERF_COUNTER_BLOCK.

Es a través de este bloque de datos de rendimiento como accedemos a los contadores. En las estructuras PERF_COUNTER_BLOCK y PERF_OBJECT_TYPE no aparecen los nombres de contadores y objetos de rendimiento, sino que lo que aparecen son índices a la colección de cadenas que se obtuvieron en el paso 2. [MSDN1]

4. Ahora ya pueden tomarse los valores de cada contador. Sin embargo, debe tenerse en mente que dependiendo de qué contador se trate, el valor puede no tener sentido en sí mismo sino que necesita de otros datos: Algunos contadores necesitan dos muestras, relacionadas por una determinada formula, para ofrecer datos que contengan información. Otros contadores están basados en tiempo y necesitan por tanto de ese dato. Por último, hay contadores que no necesitan ninguno de esos dos elementos (sus valores son valores con información) y contadores que necesitan de ambos (contadores basados en tiempo y que además necesitan dos muestras) [MSDN1]

2. Código desarrollado

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string.h>

#define TOTALBYTES    8192
#define BYTEINCREMENT 1024

LPSTR lpNameStrings;
LPSTR *lpNamesArray;
unsigned long tamano;

void reconoce (PDH_STATUS PdhStatus)
/*
      Muestra un mensaje de explicación sobre un error PDH
*/
{
      switch (PdhStatus) {
            case ERROR_SUCCESS: ShowMessage ("ERROR_SUCCESS"); break;

            case PDH_CSTATUS_BAD_COUNTERNAME:ShowMessage ("PDH_CSTATUS_BAD_COUNTERNAME"); break;
            case PDH_CSTATUS_INVALID_DATA:ShowMessage ("PDH_CSTATUS_INVALID_DATA"); break;
            case PDH_CSTATUS_ITEM_NOT_VALIDATED:ShowMessage ("PDH_CSTATUS_ITEM_NOT_VALIDATED"); break;
            case PDH_CSTATUS_NEW_DATA:ShowMessage ("PDH_CSTATUS_NEW_DATA"); break;
            case PDH_CSTATUS_NO_COUNTER:ShowMessage ("PDH_CSTATUS_NO_COUNTER"); break;
            case PDH_CSTATUS_NO_COUNTERNAME:ShowMessage ("PDH_CSTATUS_NO_COUNTERNAME"); break;
            case PDH_CSTATUS_NO_INSTANCE:ShowMessage ("PDH_CSTATUS_NO_INSTANCE"); break;
```

```
case PDH_CSTATUS_NO_MACHINE: ShowMessage ("PDH_CSTATUS_NO_MACHINE"); break;
case PDH_CSTATUS_NO_OBJECT:  ShowMessage ("PDH_CSTATUS_NO_OBJECT"); break;
case PDH_ACCESS_DENIED:ShowMessage ("PDH_ACCESS_DENIED"); break;
case PDH_CALC_NEGATIVE_DENOMINATOR:ShowMessage ("PDH_CALC_NEGATIVE_DENOMINATOR"); break;
case PDH_CALC_NEGATIVE_TIMEBASE:ShowMessage ("PDH_CALC_NEGATIVE_TIMEBASE"); break;
case PDH_CALC_NEGATIVE_VALUE:ShowMessage ("PDH_CALC_NEGATIVE_VALUE"); break;
case PDH_CANNOT_CONNECT_MACHINE:ShowMessage ("PDH_CANNOT_CONNECT_MACHINE"); break;
case PDH_CANNOT_READ_NAME_STRINGS:ShowMessage ("PDH_CANNOT_READ_NAME_STRINGS"); break;
case PDH_DATA_SOURCE_IS_LOG_FILE:ShowMessage ("PDH_DATA_SOURCE_IS_LOG_FILE"); break;
case PDH_DATA_SOURCE_IS_REAL_TIME:ShowMessage ("PDH_DATA_SOURCE_IS_REAL_TIME"); break;
case PDH_DIALOG_CANCELLED:ShowMessage ("PDH_DIALOG_CANCELLED"); break;
case PDH_END_OF_LOG_FILE:ShowMessage ("PDH_END_OF_LOG_FILE"); break;
case PDH_ENTRY_NOT_IN_LOG_FILE:ShowMessage ("PDH_ENTRY_NOT_IN_LOG_FILE"); break;
case PDH_FILE_ALREADY_EXISTS:ShowMessage ("PDH_FILE_ALREADY_EXISTS"); break;
case PDH_FILE_NOT_FOUND:ShowMessage ("PDH_FILE_NOT_FOUND"); break;
case PDH_FUNCTION_NOT_FOUND:ShowMessage ("PDH_FUNCTION_NOT_FOUND"); break;
case PDH_INVALID_ARGUMENT:ShowMessage ("PDH_INVALID_ARGUMENT"); break;
case PDH_INSUFFICIENT_BUFFER:ShowMessage ("PDH_INSUFFICIENT_BUFFER"); break;
case PDH_INVALID_BUFFER:ShowMessage ("PDH_INVALID_BUFFER"); break;
case PDH_INVALID_HANDLE:ShowMessage ("PDH_INVALID_HANDLE"); break;
case PDH_INVALID_INSTANCE:ShowMessage ("PDH_INVALID_INSTANCE"); break;
case PDH_INVALID_PATH:ShowMessage ("PDH_INVALID_PATH"); break;
case PDH_LOG_FILE_CREATE_ERROR:ShowMessage ("PDH_LOG_FILE_CREATE_ERROR"); break;
case PDH_LOG_FILE_OPEN_ERROR:ShowMessage ("PDH_LOG_FILE_OPEN_ERROR"); break;
case PDH_LOG_TYPE_NOT_FOUND:ShowMessage ("PDH_LOG_TYPE_NOT_FOUND"); break;
case PDH_LOGSVC_QUERY_NOT_FOUND:ShowMessage ("PDH_LOGSVC_QUERY_NOT_FOUND"); break;
case PDH_LOGSVC_NOT_OPENED:ShowMessage ("PDH_LOGSVC_NOT_OPENED"); break;
case PDH_MEMORY_ALLOCATION_FAILURE:ShowMessage ("PDH_MEMORY_ALLOCATION_FAILURE"); break;
case PDH_MORE_DATA:ShowMessage ("PDH_MORE_DATA"); break;
case PDH_NO_DATA:ShowMessage ("PDH_NO_DATA"); break;
```

```
            case PDH_NO_DIALOG_DATA:ShowMessage ("PDH_NO_DIALOG_DATA"); break;
            case PDH_NO_MORE_DATA:ShowMessage ("PDH_NO_MORE_DATA"); break;
            case PDH_NOT_IMPLEMENTED:ShowMessage ("PDH_NOT_IMPLEMENTED"); break;
            case PDH_RETRY:ShowMessage ("PDH_RETRY"); break;
            case PDH_STRING_NOT_FOUND:ShowMessage ("PDH_STRING_NOT_FOUND"); break;
            case PDH_UNABLE_MAP_NAME_FILES:ShowMessage ("PDH_UNABLE_MAP_NAME_FILES"); break;
            case PDH_UNABLE_READ_LOG_HEADER:ShowMessage ("PDH_UNABLE_READ_LOG_HEADER"); break;
            case PDH_UNKNOWN_LOG_FORMAT:ShowMessage ("PDH_UNKNOWN_LOG_FORMAT"); break;
            case PDH_UNKNOWN_LOGSVC_COMMAND:ShowMessage ("PDH_UNKNOWN_LOGSVC_COMMAND"); break;
            case PDH_WBEM_ERROR:ShowMessage ("PDH_WBEM_ERROR"); break;

            default:ShowMessage ("Error desconocido");
        }
};

/*****************************************************************
 *                                                               *
 * Functions used to navigate through the performance data.      *
 *                                                               *

 *****************************************************************/

PPERF_OBJECT_TYPE FirstObject( PPERF_DATA_BLOCK PerfData )
{
   return( (PPERF_OBJECT_TYPE)((PBYTE)PerfData +
      PerfData->HeaderLength) );
}

PPERF_OBJECT_TYPE NextObject( PPERF_OBJECT_TYPE PerfObj )
{
```

```
    return( (PPERF_OBJECT_TYPE)((PBYTE)PerfObj +
        PerfObj->TotalByteLength) );
}

PPERF_INSTANCE_DEFINITION FirstInstance( PPERF_OBJECT_TYPE PerfObj )
{
    return( (PPERF_INSTANCE_DEFINITION)((PBYTE)PerfObj +

        PerfObj->DefinitionLength) );
}

PPERF_INSTANCE_DEFINITION NextInstance(
    PPERF_INSTANCE_DEFINITION PerfInst )
{
    PPERF_COUNTER_BLOCK PerfCntrBlk;

    PerfCntrBlk = (PPERF_COUNTER_BLOCK)((PBYTE)PerfInst +
        PerfInst->ByteLength);

    return( (PPERF_INSTANCE_DEFINITION)((PBYTE)PerfCntrBlk +
        PerfCntrBlk->ByteLength) );
}

PPERF_COUNTER_DEFINITION FirstCounter( PPERF_OBJECT_TYPE PerfObj )
{
    return( (PPERF_COUNTER_DEFINITION) ((PBYTE)PerfObj +

        PerfObj->HeaderLength) );
}
```

```
PPERF_COUNTER_DEFINITION NextCounter(
   PPERF_COUNTER_DEFINITION PerfCntr )
{
   return( (PPERF_COUNTER_DEFINITION)((PBYTE)PerfCntr +
      PerfCntr->ByteLength) );
}

/*************************************************************
 *                                                           *
 * Load the counter and object names from the registry to the    *
 * global variable lpNamesArray.                             *

 *                                                           *
 *************************************************************/

void GetNameStrings( )
{
   HKEY hKeyPerflib;      // handle to registry key
   HKEY hKeyPerflib009;   // handle to registry key
   DWORD dwMaxValueLen;   // maximum size of key values
   DWORD dwBuffer;        // bytes to allocate for buffers
   DWORD dwBufferSize;    // size of dwBuffer
   LPSTR lpCurrentString; // pointer for enumerating data strings

   DWORD dwCounter;       // current counter index

// Get the number of Counter items.

   RegOpenKeyEx( HKEY_LOCAL_MACHINE,
      "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Perflib",
```

```
        0,
        KEY_READ,
        &hKeyPerflib);

    dwBufferSize = sizeof(dwBuffer);

    RegQueryValueEx( hKeyPerflib,
        "Last Counter",
        NULL,
        NULL,
        (LPBYTE) &dwBuffer,
        &dwBufferSize );

    RegCloseKey( hKeyPerflib );

// Allocate memory for the names array.

    lpNamesArray = (char**) malloc( (dwBuffer+1) * sizeof(LPSTR) );
    tamano = dwBuffer+1;

// Open key containing counter and object names.

    RegOpenKeyEx( HKEY_LOCAL_MACHINE,
    "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Perflib\\009",
        0,
        KEY_READ,
        &hKeyPerflib009);

// Get the size of the largest value in the key (Counter or Help).
```

```
    RegQueryInfoKey( hKeyPerflib009,
       NULL,
       NULL,
       NULL,

       NULL,
       NULL,
       NULL,
       NULL,
       NULL,
       &dwMaxValueLen,
       NULL,
       NULL);

// Allocate memory for the counter and object names.

    dwBuffer = dwMaxValueLen + 1;

    lpNameStrings =(char*) malloc( dwBuffer * sizeof(CHAR) );

// Read Counter value.

    RegQueryValueEx( hKeyPerflib009,
       "Counter",
       NULL,
       NULL,
       lpNameStrings, &dwBuffer );

// Load names into an array, by index.
```

```
   for( lpCurrentString = lpNameStrings; *lpCurrentString;
        lpCurrentString += (lstrlen(lpCurrentString)+1) )
   {
      dwCounter = atol( lpCurrentString );

      lpCurrentString += (lstrlen(lpCurrentString)+1);

      lpNamesArray[dwCounter] = (LPSTR) lpCurrentString;
   }
}

/*****************************************************************
 *                                                               *
 * Devuelve la lista con los contadores                          *
 *                                                               *
 *****************************************************************/

void disp2(TStringList *lista)
{
   PPERF_DATA_BLOCK PerfData = NULL;
   PPERF_OBJECT_TYPE PerfObj;
   PPERF_INSTANCE_DEFINITION PerfInst;
   PPERF_COUNTER_DEFINITION PerfCntr, CurCntr;
   PPERF_COUNTER_BLOCK PtrToCntr;
   DWORD BufferSize = TOTALBYTES;
   DWORD i, j, k;
   PDH_STATUS PdhStatus;
   LPCTSTR szWildCardPath;
   LPTSTR mszExpandedPathList = NULL;
   LPDWORD pcchPathListLength = new unsigned long;
```

```
    AnsiString Objeto;
    AnsiString otro;
    AnsiString contador;
    AnsiString wildCard;
    bool fallo;


// Consigue los nombre de los contadores a través del registro

    GetNameStrings( );

// Reserva espacio para los datos de rendimiento

    PerfData = (PPERF_DATA_BLOCK) malloc( BufferSize );

    while( RegQueryValueEx( HKEY_PERFORMANCE_DATA,
                            "Global",
                            NULL,
                            NULL,
                            (LPBYTE) PerfData,
                            &BufferSize ) == ERROR_MORE_DATA )
    {
       BufferSize += BYTEINCREMENT;
       PerfData = (PPERF_DATA_BLOCK) realloc( PerfData, BufferSize );
    }

// Obtiene el primer objeto

    PerfObj = FirstObject( PerfData );
```

```
// Procesa todos los objetos

    for( i=0; i < PerfData->NumObjectTypes; i++ )
    {
                fallo = false;
            Objeto = lpNamesArray[PerfObj->ObjectNameTitleIndex];

    // Obtiene el primer contador
        PerfCntr = FirstCounter( PerfObj );

        *pcchPathListLength = 1;
       wildCard = "\\";
       Objeto = wildCard + Objeto;
       // esta parte de la ruta va tanto si tiene instancias como si no

        if( PerfObj->NumInstances > 0 )
        {
                // el Objeto tiene instancias.
                // Expandimos la lista correspondiente
                contador = "(*/";
                otro = "*#*)\\*";
                otro = contador + otro;      // segunda parte lista
                wildCard = Objeto + otro;  //"\\IP\\*";    // unión lista
                // ya está preparada la ruta con wildCards
        }
        else
        // sólo una instancia
        {
                contador = "\\*";
                wildCard = Objeto + contador;
```

```
            // ya está preparada la ruta con wildCards
        };

        // expandimos la lista con wildCards
        szWildCardPath = (char*) wildCard.data();
        mszExpandedPathList = (char*) GlobalAlloc (GPTR, *pcchPathListLength);
        PdhStatus = PdhExpandCounterPath (
            szWildCardPath,         // in Puntero a una cadena acabada en null que especifica el nombre de la path a expandir
            mszExpandedPathList,    // out Puntero a un buffer que recibe la MULTI_SZ lista de counter path names que encajan con
la especificación de la wildcard
            pcchPathListLength      // inout Puntero a una variable que especifica el tamaño del mszExpandedPathList buffer, en
TCHARs
                                    // recibe el tamaño que se ha usado realmente
        );

        if (PdhStatus == PDH_MORE_DATA)
        {
            GlobalFree (mszExpandedPathList);
            *pcchPathListLength = *pcchPathListLength+1;
            mszExpandedPathList = (char*) GlobalAlloc (GPTR, *pcchPathListLength);
            PdhStatus = PdhExpandCounterPath (szWildCardPath,
                mszExpandedPathList,pcchPathListLength);
        };
        if (PdhStatus != ERROR_SUCCESS)
        {
            ShowMessage (wildCard);
            reconoce (PdhStatus);
            fallo = true;
        };
        if (!fallo)
```

```
        {
                // añadimos la lista de contadores
                contador = "";
                for (unsigned long j=0; j<*pcchPathListLength; j++)
                {
                        if (mszExpandedPathList[j]==NULL)
                        {
                                lista->Add (contador);
                                contador = "";
                        }
                        else contador = contador + mszExpandedPathList[j];
                }
                if (*pcchPathListLength > 0)
                    lista->Delete (lista->Count -1);
                    // la última línea es NULL
        }

     // Pasamos al siguiente objeto
        GlobalFree (mszExpandedPathList);
        PerfObj = NextObject( PerfObj );
    }
}

void capturaContadores (TStringList* lista, AnsiString nArchivo)
{
   disp2 (lista);
   // ahora están todos los contadores
   lista->SaveToFile (nArchivo);
}
```

Anexo 1.3: Código desarrollado para la obtención de datos de rendimiento a través de Performance Data Helper

1. Proceso de acceso a datos de rendimiento a través de PDH

Para poder acceder a los datos de rendimiento utilizando funciones de la librería Performance Data Helper es necesario seguir una serie de pasos que se detallan a continuación:

1. Obtener las rutas de los contadores de rendimiento.

Existen varias funciones en PDH que sirven para obtener los elementos necesarios para construir la ruta de un contador de rendimiento:

- PdhEnumMachines () devuelve la lista de nombres o direcciones de las máquinas conectadas a la máquina local. Como para identificar un contador de la máquina local no es necesario especificar el nombre de ésta, PdhEnumMachines () no lo incluye en la lista de nombres que devuelve.
- PdhEnumObjects () devuelve la lista de los nombres de los objetos de rendimiento presentes en el ordenador que se le especifique.
- PdhEnumObjectItems () devuelve la lista de los nombres de contadores de rendimiento, e instancias en su caso, que pertenecen a un objeto cuyo nombre se proporcione. En los nombres de las instancias que se devuelven se incluye en su caso el nombre de instancia del padre, aunque no el índice de instancia.
- PdhExpandCounterPath () recibe como parámetro una ruta de contador de rendimiento en la que se permite la presencia de caracteres "comodín" en lugar de algunos de los elementos de la ruta. La función devuelve una lista con todas las rutas válidas que se obtienen de sustituir esos caracteres comodín por cada elemento adecuado. Por ejemplo, si se llama a PdhExpandCounterPath () y se le entrega como ruta una en la que se especifique el nombre de objeto y de instancia, pero en el lugar del nombre de contador aparezca un asterisco, la lista que devolverá la función (suponiendo que el nombre de instancia corresponda a ese objeto y que éste este presente en el sistema) contendrá todas las rutas con el nombre de objeto e instancia entregados, y en las que en lugar del asterisco aparece el nombre de un contador válido.
- PdhValidatePath () comprueba la validez de una ruta que se le pase como parámetro.

[PDHREF1]

2. Crear una consulta:

La creación de una consulta se realiza a través de la función PdhOpenQuery (). Esta función devuelve un manejador de la consulta que se usará más adelante. [PDHREF1]

3. Añadir contadores a la consulta:

Por cada contador de rendimiento que desee añadirse a la consulta debe usarse la función PdhAddCounter(), que requiere como parámetro la ruta del contador de rendimiento que se desee monitorizar, debiendo estar aquella completamente cualificada. [PDHREF1]

4. Obtener los datos de rendimiento:

Llamando a la función PdhCollectQueryData() y pasándole como parámetro el manejador de una consulta, se obtienen los datos de todos los contadores presentes en dicha consulta. Como ya se dijo en la sección sobre el interfaz del registro, los valores de los contadores no siempre bastan por sí mismos. Por esta razón, al llamar a PdhCollectQueryData(), no se desechan los valores obtenidos en la última vez que se realizó la llamada (si la hubo) con la misma consulta, sino que se guardan para poder usarlos en los contadores necesarios. [PDHREF1]

4. Procesar los datos de rendimiento

Para dar un formato a los datos obtenidos en el paso anterior se usa PdhGetFormattedCounterValue(). Esta función convierte los datos del contador seleccionado a formato integer, float o double según se le indique en uno de sus parámetros. La función almacena el resultado de las operaciones en una estructura del tipo PDH_FMT_COUNTERVALUE. [PDHREF1]

En las siguientes páginas se presenta el código que fue utilizado para el uso de PDH

```
#ifndef FREGH
#define FREGH

#include <iostream.h>   // para usar inserción y extracción con AnsiString
#include <fstream.h>    // para manejar archivos
#include <pdh.h>        // acceso a librería PDH
#include <pdhmsg.h>     // acceso a los mensajes de la librería PDH
#include <windows.h>    // funciones estándar windows
#include <vcl.h>        // componentes VCL como TStringList
#include <malloc.h>     // acceso a reserva y liberación de memoria

//---------------------------------------------------------------------------
/*
IN:    - serieCadenas es una multistring
INOUT: - TStringList. Contendrá las cadenas de serieCadenas como AnsiString
NOTA:  Las funciones de la librería PDH suelen dejar salida como multistrings
*/
void charToStringList (char* serieCadenas, TStringList* lista);
//---------------------------------------------------------------------------

/*
  La clase monitor incluye métodos y atributos para ayudar a consultar los
contadores de rendimiento
*/

class monitor {
private:
        // ATRIBUTOS DE LA CLASE
        ofstream salida;        // Fichero donde se introducen los datos
        HQUERY hQuery;          // Manejador de la Query PDH
```

```
HCOUNTER *hCounter;     // Manejador de contadores PDH
TStringList* contadores;// Estructura que almacena las rutas de los
                        // contadores de rendimiento

// MÉTODOS PRIVADOS
/*
IN: - l tiene las rutas de los contadores que deseamos incluir
    - f es la ruta de acceso al archivo donde deben dejarse los datos
NOTAS:
    - El archivo de salida será sobreescrito
    - El archivo es modificado, en este orden, por las funciones:
      a) iniciaCabecera (): una vez
      b) contadorAnadido (): una vez por cada contador válido
      c) acabaCabecera (): una vez
*/
void creaQuery (TStrings* l, AnsiString f);

/*
IN: f es el fichero donde dejar los datos obtenidos
NOTAS:
    - Se toman los datos que haya en los contadores de hQuery
    - Se les da cierto formato y se introducen en el archivo f
    - Dicho archivo es modificado, en este orden, por las funciones:
      a) ponHora (): una vez
      b) formateaDatos (): una vez
*/
void cuentaP (ofstream* f);

/*
IN: f es el fichero donde dejar los datos obtenidos
```

```
NOTAS:
    - Los datos son formateados como double
    - f es modificado por la función ponMediciones ()
*/
virtual void formateaDatos (ofstream* f);

/*
IN: f es una cadena que sea necesaria para el cometido de la función
NOTA:
    El fichero salida es modificado iniciándose la cabecera
*/
virtual void iniciaCabecera (AnsiString f);

/*
IN: n es una cadena que sea necesaria para el cometido de la función
NOTA:
    El fichero salida es modificado del modo adecuado a la inclusión
de un contador de rendimiento.
*/
virtual void contadorAnadido (AnsiString n);

/*
NOTA: Modifica el fichero salida terminando su cabecera
*/
virtual void acabaCabecera ();

/*
IN: f es el fichero que debe modificarse
NOTA:
    El fichero salida debe ser modificado como corresponda a la hora
```

```
        */
        virtual void ponHora (ofstream* f);

        /*
        IN: - m es una serie de double con los valores que hay que incluir
            - f es el fichero en donde hay que incluirlos
        NOTA:
            El fichero salida es actualizado con los datos en m
            En m hay tantos valores como contadores en hQuery
        */
        virtual void ponMediciones (double* m, ofstream* f);


public:
        // MÉTODOS PÚBLICOS
        monitor () {contadores = new TStringList ();};

        /*
        IN: Descripción de un código de error PDH
        NOTAS:
             Devuelve un AnsiString con un mensaje adecuado al error descrito
        */
        static AnsiString codigoError (PDH_STATUS PdhStatus);

        /*
        IN: - l tiene las rutas de los contadores que deseamos incluir
            - f es la ruta de acceso al archivo donde deben dejarse los datos
        NOTAS:
            Prepara hQuery, hCounter y salida para la captura de datos
        */
```

```
void inicia (TStrings* l, AnsiString f);

/*
NOTAS: Hace una estimación del tiempo que tardará cuenta en terminar
*/
Cardinal estimaTiempo ();

/*
NOTAS: hace una llamada a cuentaP con el fichero salida
*/
void cuenta ();

/*
Finaliza la toma de datos con los parámetros actuales
*/
void para ();

/*
INOUT: l es una estructura de AnsiString donde serán devueltos los
       nombres de los objetos de rendimiento en la máquina local
*/
static void devuelveObjetos (TStringList *l);

/*
IN: - objeto contiene el nombre del objeto de rendimiento del que
    queremos saber los nombres de sus instancias y sus contadores
INOUT: - li es una estructura de AnsiString donde serán devueltos los
    nombres de las instancias del objeto de rendimiento indicado
       - lc es una estructura de AnsiString donde serán devueltos los
    nombres de los contadores del objeto de rendimiento indicado
```

```
        */
        static void devuelveInstanciasContadores (AnsiString objeto, TStringList *li, TStringList* lc);

        /*
        IN: - objeto contiene el nombre del objeto de rendimiento
            - contador es un contador de dicho objeto
        INOUT: instancias contiene los nombres de las instancias conocidas del
            objeto de rendimiento al hacer la llamada a la función. En la salida
            contiene la lista completa de instancias existentes incluyendo, en
            su caso, los índices de instancia
        */
        static void extiendeInstancias (AnsiString objeto,
                                        TStringList* instancias,
                                        AnsiString contador);

        /*
        IN: - objeto contiene el nombre del objeto de rendimiento
            - instancia contiene el nombre de la instancia de objeto deseada, ""
            en caso de que el objeto no tenga instancias
            - contador contiene el nombre del contador de objeto deseado.
        OUT: - Devuelve una cadena vacía si la ruta formada no es válida.
             - En otro caso devuelve la ruta formada
        */
        static AnsiString hazPathContador (AnsiString objeto, AnsiString instancia, AnsiString contador);
};

#endif
```

```
//---------------------------------------------------------------------------
// IMPLEMENTACIONES
//---------------------------------------------------------------------------

void charToStringList (char* serieCadenas, TStringList* lista)
{
	// una variable auxiliar
	AnsiString temp = "";

	// vaciamos lista
	lista->Clear();

	for (unsigned long i = 0; // desde el inicio de la cadena
	     serieCadenas[i]; // hasta no encontrar el final de la colección
	     i++)
	{
		// hasta no encontrar el final de la cadena
		for (;serieCadenas[i];i++)
			// actualizamos la cadena temp
			temp = temp + serieCadenas[i];
		// terminada la cadena actual, la incluimos en lista
		lista->Add (temp);
		temp = "";
	}
};

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

// IMPLEMENTACIÓN DE LAS FUNCIONES DE MONITOR
```

```
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

void monitor::creaQuery (TStrings* l, AnsiString f)
{
	// variable para controlar problemas con funciones de PDH
	PDH_STATUS PdhStatus;

	// inicialmente no hay ningún contador en la query
	contadores ->Clear();

	// escribe la cabecera
	iniciaCabecera (f);

	// abrimos la query
	PdhStatus = PdhOpenQuery (0, 0, &hQuery);

	// metemos los contadores uno por uno
	for (int i = 0; i < l->Count; i++)
	{
		PdhStatus = PdhAddCounter(hQuery,		// query en la que queremos el contador
			(const char*) l->Strings[i].data(),	// ruta al contador
			i,				// índice del contador
			&(hCounter[i]));			// manejador del contador
		if (PdhStatus == ERROR_SUCCESS) {
			// si todo está correcto tenemos que introducir nuevos datos en salida
			// y en la estructura con las rutas de los contadores
			contadores->Add (l->Strings[i]);
			contadorAnadido (l->Strings[i]);
```

```
            }
        }

        // terminamos de escribir la cabecera
        acabaCabecera ();
};

//---------------------------------------------------------------------------

void monitor::cuentaP (ofstream* f)
{
        // introduce la fecha y hora
        ponHora (f);

        // ejecutamos la query
        PdhCollectQueryData(hQuery);

        // formateamos e introducimos en f los datos obtenidos
        formateaDatos (f);
};

//---------------------------------------------------------------------------

void monitor::formateaDatos (ofstream* f)
{
        // reservamos memoria para los datos tanto en la estructura PDH como en double
        PDH_FMT_COUNTERVALUE *valor;
        valor = (PDH_FMT_COUNTERVALUE *) GlobalAlloc (GPTR, sizeof(PDH_FMT_COUNTERVALUE));
        double *mediciones = new double[contadores->Count];
```

```
	// para cada contador obtenemos su valor
	for (int i=0; i < contadores->Count; i++) {
		PdhGetFormattedCounterValue(
			hCounter[i],		// manejador del contador
			PDH_FMT_DOUBLE,	// queremos que el valor sea un doble
			NULL,			// tipo del contador
			valor);		// dónde guardamos el valor
		// en valor hay 3 campos, longValue, doubleValue, largeValue
		mediciones[i] = valor->doubleValue;
	}

	// introducimos los datos en la salida
	ponMediciones (mediciones, f);

	// liberamos el espacio reservado en la función
	GlobalFree (valor);
	delete mediciones;
};

//---------------------------------------------------------------------------

void monitor::iniciaCabecera (AnsiString f)
{
	// f hace de nombre del log
	salida << "@RELATION " << f << "\n" << "\n";
	salida << "@ATTRIBUTE time DATE\n";
};

//---------------------------------------------------------------------------
```

```
void monitor::contadorAnadido (AnsiString n)
{
	// n es la ruta del contador añadido
	salida << "@ATTRIBUTE " << n << " NUMERIC\n";
};

//---------------------------------------------------------------------------
void monitor::acabaCabecera ()
{
	// @DATA es el marcador de comienzo de los datos
	salida << "\n@DATA\n\n";
};

//---------------------------------------------------------------------------

void monitor::ponHora (ofstream* f)
{
	// cadena que describa fecha y hora
	AnsiString tiempo;

	DateTimeToString(tiempo,		// dónde dejar la cadena
		"dd/mm/yyyy//hh:nn:ss:zzz",	// cómo formatear fecha y hora
		Time());			// de dónde obtener los datos

	// introducimos los datos en f
	(*f) << tiempo;
};

//---------------------------------------------------------------------------
```

```
void monitor::ponMediciones (double* m, ofstream* f)
{
	// para cada contador introducimos el valor asociado separando con comas
	for (int i=0; i<contadores->Count; i++)
		(*f) << "," << m[i];

	// al acabar con los contadores, cambiamos de línea
	(*f) << "\n";
};

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

AnsiString monitor::codigoError (PDH_STATUS PdhStatus)
{
	AnsiString respuesta;
	switch (PdhStatus) {
		case ERROR_SUCCESS:			respuesta = "Todo bien"; break;

		case PDH_CSTATUS_BAD_COUNTERNAME:	respuesta = "PDH_CSTATUS_BAD_COUNTERNAME"; break;
		case PDH_CSTATUS_INVALID_DATA:		respuesta = "PDH_CSTATUS_INVALID_DATA"; break;
		case PDH_CSTATUS_ITEM_NOT_VALIDATED:	respuesta = "PDH_CSTATUS_ITEM_NOT_VALIDATED"; break;
		case PDH_CSTATUS_NEW_DATA:		respuesta = "PDH_CSTATUS_NEW_DATA"; break;
		case PDH_CSTATUS_NO_COUNTER:		respuesta = "PDH_CSTATUS_NO_COUNTER"; break;
		case PDH_CSTATUS_NO_COUNTERNAME:	respuesta = "PDH_CSTATUS_NO_COUNTERNAME"; break;
		case PDH_CSTATUS_NO_INSTANCE:		respuesta = "PDH_CSTATUS_NO_INSTANCE"; break;
		case PDH_CSTATUS_NO_MACHINE:		respuesta = "PDH_CSTATUS_NO_MACHINE"; break;
		case PDH_CSTATUS_NO_OBJECT:		respuesta = "PDH_CSTATUS_NO_OBJECT"; break;
		case PDH_ACCESS_DENIED:			respuesta = "PDH_ACCESS_DENIED"; break;
		case PDH_CALC_NEGATIVE_DENOMINATOR:	respuesta = "PDH_CALC_NEGATIVE_DENOMINATOR"; break;
```

```
case PDH_CALC_NEGATIVE_TIMEBASE:        respuesta = "PDH_CALC_NEGATIVE_TIMEBASE"; break;
case PDH_CALC_NEGATIVE_VALUE:           respuesta = "PDH_CALC_NEGATIVE_VALUE"; break;
case PDH_CANNOT_CONNECT_MACHINE:        respuesta = "PDH_CANNOT_CONNECT_MACHINE"; break;
case PDH_CANNOT_READ_NAME_STRINGS:      respuesta = "PDH_CANNOT_READ_NAME_STRINGS"; break;
case PDH_DATA_SOURCE_IS_LOG_FILE:       respuesta = "PDH_DATA_SOURCE_IS_LOG_FILE"; break;
case PDH_DATA_SOURCE_IS_REAL_TIME:      respuesta = "PDH_DATA_SOURCE_IS_REAL_TIME"; break;
case PDH_DIALOG_CANCELLED:              respuesta = "PDH_DIALOG_CANCELLED"; break;
case PDH_END_OF_LOG_FILE:               respuesta = "PDH_END_OF_LOG_FILE"; break;
case PDH_ENTRY_NOT_IN_LOG_FILE:         respuesta = "PDH_ENTRY_NOT_IN_LOG_FILE"; break;
case PDH_FILE_ALREADY_EXISTS:           respuesta = "PDH_FILE_ALREADY_EXISTS"; break;
case PDH_FILE_NOT_FOUND:                respuesta = "PDH_FILE_NOT_FOUND"; break;
case PDH_FUNCTION_NOT_FOUND:            respuesta = "PDH_FUNCTION_NOT_FOUND"; break;
case PDH_INVALID_ARGUMENT:              respuesta = "PDH_INVALID_ARGUMENT"; break;
case PDH_INSUFFICIENT_BUFFER:           respuesta = "PDH_INSUFFICIENT_BUFFER"; break;
case PDH_INVALID_BUFFER:                respuesta = "PDH_INVALID_BUFFER"; break;
case PDH_INVALID_HANDLE:                respuesta = "PDH_INVALID_HANDLE"; break;
case PDH_INVALID_INSTANCE:              respuesta = "PDH_INVALID_INSTANCE"; break;
case PDH_INVALID_PATH:                  respuesta = "PDH_INVALID_PATH"; break;
case PDH_LOG_FILE_CREATE_ERROR:         respuesta = "PDH_LOG_FILE_CREATE_ERROR"; break;
case PDH_LOG_FILE_OPEN_ERROR:           respuesta = "PDH_LOG_FILE_OPEN_ERROR"; break;
case PDH_LOG_TYPE_NOT_FOUND:            respuesta = "PDH_LOG_TYPE_NOT_FOUND"; break;
case PDH_LOGSVC_QUERY_NOT_FOUND:        respuesta = "PDH_LOGSVC_QUERY_NOT_FOUND"; break;
case PDH_LOGSVC_NOT_OPENED:             respuesta = "PDH_LOGSVC_NOT_OPENED"; break;
case PDH_MEMORY_ALLOCATION_FAILURE:     respuesta = "PDH_MEMORY_ALLOCATION_FAILURE"; break;
case PDH_MORE_DATA:                     respuesta = "PDH_MORE_DATA"; break;
case PDH_NO_DATA:                       respuesta = "PDH_NO_DATA"; break;
case PDH_NO_DIALOG_DATA:                respuesta = "PDH_NO_DIALOG_DATA"; break;
case PDH_NO_MORE_DATA:                  respuesta = "PDH_NO_MORE_DATA"; break;
case PDH_NOT_IMPLEMENTED:               respuesta = "PDH_NOT_IMPLEMENTED"; break;
case PDH_RETRY:                         respuesta = "PDH_RETRY"; break;
```

```
            case PDH_STRING_NOT_FOUND:              respuesta = "PDH_STRING_NOT_FOUND"; break;
            case PDH_UNABLE_MAP_NAME_FILES:         respuesta = "PDH_UNABLE_MAP_NAME_FILES"; break;
            case PDH_UNABLE_READ_LOG_HEADER:        respuesta = "PDH_UNABLE_READ_LOG_HEADER"; break;
            case PDH_UNKNOWN_LOG_FORMAT:            respuesta = "PDH_UNKNOWN_LOG_FORMAT"; break;
            case PDH_UNKNOWN_LOGSVC_COMMAND:        respuesta = "PDH_UNKNOWN_LOGSVC_COMMAND"; break;
            case PDH_WBEM_ERROR:                    respuesta = "PDH_WBEM_ERROR"; break;

            default:                                respuesta = "Mensaje desconocido";
        }
        return respuesta;
};

//---------------------------------------------------------------------------

void monitor::inicia (TStrings* l, AnsiString f)
{
        // establecemos el separador decimal
        DecimalSeparator = '.';

        // inicializamos los manejadores de los contadores
        hCounter = new HCOUNTER[l->Count];

        // abre el fichero destino
        salida.open ((const char*)f.data());

        // creamos una query y la rellenamos mientras completamos la cabecera del log
        creaQuery (l, f);
};

//---------------------------------------------------------------------------
```

```
Cardinal monitor::estimaTiempo ()
{
	// variables para calcular el tiempo que tardaremos
	TDateTime tiempo;
	unsigned short tReal [4];
	Cardinal respuesta;
	// un fichero de pruebas
	ofstream prueba;

	// abrimos el fichero de prueba
	prueba.open ((const char*)"prueba.txt");

	// iniciamos el cálculo del tiempo
	tiempo = Time();

	// acciones que se toman en un cuenta()
	cuentaP (&prueba);

	// calculamos el tiempo tardado
	tiempo = Time() - tiempo;

	// calculamos el intervalo en el tipo adecuado para responder
	tiempo.DecodeTime(&(tReal[3]),  // horas
			&(tReal[2]),    // minutos
			&(tReal[1]),    // segundos
			&(tReal[0]));   // milisegundos
	respuesta = 3600000 * tReal[3] + 60000 * tReal[2] + 1000 * tReal[1] + tReal[0];

	// cerramos el archivo de prueba
```

```
	prueba.close ();

	return respuesta;
};

//---------------------------------------------------------------------------
void monitor::cuenta ()
{
	cuentaP (&salida);
};

//---------------------------------------------------------------------------
void monitor::para ()
{
	// cerramos la consulta y la salida
	PdhCloseQuery (hQuery);
	salida.close();

	// liberamos la memoria ocupada por los manejadores de los contadores
	delete hCounter;
};

//---------------------------------------------------------------------------

void monitor::devuelveObjetos (TStringList *l)
{
	// variable para controlar los errores PDH
	PDH_STATUS PdhStatus;

	// capacidad de la lista de cadenas que devolverá la función PDH
```

```
LPDWORD capacidadBuffer = new unsigned long;
// lista de cadenas que devolverá la función PDH
LPTSTR listaObjetos;
// nivel de detalle máximo.
DWORD nivelDetalle = PERF_DETAIL_WIZARD;

// reservamos la memoria inicial
*capacidadBuffer = 1;
listaObjetos = (char*) GlobalAlloc (GPTR, *capacidadBuffer);

PdhStatus = PdhEnumObjects(
	NULL,			// reservado, debe ser NULL
	NULL,			// nombre de la máquina, NULL para coger la local
	listaObjetos,		// buffer para los nombres de objetos.
	capacidadBuffer,	// tamaño del buffer
	nivelDetalle,		// nivel de detalle
	FALSE			// para refrescar la lista de máquinas conectadas
);

if (PdhStatus == PDH_MORE_DATA)
// no teníamos espacio suficiente
// la cantidad de espacio necesario está en capacidadBuffer
{
	// liberamos el espacio que habíamos ocupado
	GlobalFree (listaObjetos);

	// reservamos suficiente memoria
	*capacidadBuffer = *capacidadBuffer+1;
	listaObjetos = (char*) GlobalAlloc (GPTR, *capacidadBuffer);
```

```
            // volvemos a hacer la llamada
            PdhStatus = PdhEnumObjects(NULL, NULL, listaObjetos,
                  capacidadBuffer, nivelDetalle, FALSE);
      };

      // ponemos la lista de objetos en l
      // será una lista ordenada, sin duplicados
      l->Clear();
      l->Sorted = true;
      l->Duplicates = dupIgnore;
      charToStringList (listaObjetos, l);

      // liberamos la memoria reservada en esta función
      delete capacidadBuffer;
      GlobalFree (listaObjetos);
};

//---------------------------------------------------------------------------

void monitor::devuelveInstanciasContadores (AnsiString objeto,
                              TStringList *li,
                              TStringList* lc)
{
      // variable para controlar los errores PDH
      PDH_STATUS PdhStatus;

      // buffers para los nombres de los contadores y las instancias
      LPTSTR listaContadores;
      LPTSTR listaInstancias;
```

```
// tamaños de los buffers anteriores
LPDWORD tamBufferContadores = new unsigned long;
LPDWORD tamBufferInstancias = new unsigned long;

// nivel de detalle máximo
DWORD nivelDetalle = PERF_DETAIL_WIZARD;

// reservamos el espacio inicial
*tamBufferContadores = 1;
*tamBufferInstancias = 1;
listaContadores = (char*) GlobalAlloc (GPTR, *tamBufferContadores);
listaInstancias = (char*) GlobalAlloc (GPTR, *tamBufferInstancias);

PdhStatus = PdhEnumObjectItems(
        NULL,                   // reservado, debe ser NULL
        NULL,                   // NULL para que use la máquina local
        (char*) objeto.data(),  // nombre del objeto
        listaContadores,// buffer para nombres de contadores
        tamBufferContadores,  // tamaño del buffer para nombres de contadores
        listaInstancias,   // buffer para nombres de instancias
        tamBufferInstancias,    // tamaño del buffer para nombres de instancias
        nivelDetalle,             // nivel de detalle
        0);               // reservado, debe ser 0

if ((PdhStatus == PDH_MORE_DATA)
   || (PdhStatus == PDH_INSUFFICIENT_BUFFER))
   // en tamBufferContadores y tamBufferInstancias está el tamaño
   // necesario de listaContadores y listaInstancias respectivamente
{
        // liberamos la memoria que habíamos ocupado
```

```
        GlobalFree (listaContadores);
        GlobalFree (listaInstancias);

        // reservamos una cantidad adecuada de memoria
        *tamBufferContadores = *tamBufferContadores + 1;
        *tamBufferInstancias = *tamBufferInstancias + 1;
        listaContadores = (char*) GlobalAlloc (GPTR, *tamBufferContadores);
        listaInstancias = (char*) GlobalAlloc (GPTR, *tamBufferInstancias);

        // volvemos a hacer la llamada
        PdhStatus = PdhEnumObjectItems(NULL, NULL, (char*) objeto.data(),
              listaContadores, tamBufferContadores, listaInstancias,
              tamBufferInstancias, nivelDetalle, 0);
};

// ponemos los datos recogidos en las TStringList
// tanto li como lc están ordenadas y sin duplicados
lc->Clear(); lc->Sorted = true; lc->Duplicates = dupIgnore;
li->Clear(); li->Sorted = true; li->Duplicates = dupIgnore;
charToStringList (listaContadores, lc);

// si tenemos instancias, las extendemos para no perder los índices de instancia
if (*tamBufferInstancias > 1) {
        charToStringList (listaInstancias, li);
        extiendeInstancias (objeto, li, lc->Strings[0]);
}

// liberamos la memoria utilizada
delete tamBufferContadores;
delete tamBufferInstancias;
```

```
	GlobalFree (listaContadores);
	GlobalFree (listaInstancias);
};

//---------------------------------------------------------------------------

void monitor::extiendeInstancias (AnsiString objeto,
				TStringList* instancias,
				AnsiString contador)
{
	// número de instancias
	int n = instancias->Count;

	// ruta del contador
	AnsiString o;

	// combinación de instancia e índice de instancia
	AnsiString asp;

	// índice de instancia
	int ext;

	// variable para controlar los errores PDH
	PDH_STATUS PdhStatus;

	// para cada instancia
	for (int i = 0; i<n; i++) {
		ext = 0;
		// comprobamos qué índices de instancia admite
		do {
```

```
                // actualizamos el índice
                ext++;

                // combinamos nombre de instancia e índice
                asp = instancias->Strings[i] + "#" + IntToStr (ext);

                // preparamos la ruta
                o = "\\" + objeto + "(" + asp +")\\" + contador;

                // comprobamos la ruta
                PdhStatus = PdhValidatePath ((char *)o.data());

                // si es correcta, la añadimos
                if (PdhStatus == ERROR_SUCCESS)
                        instancias->Add (asp);
        } while (PdhStatus == ERROR_SUCCESS);
    }
};

//---------------------------------------------------------------------------

AnsiString monitor::hazPathContador (AnsiString objeto,
                        AnsiString instancia,
                        AnsiString contador)
{
    // variable para controlar los errores PDH
    PDH_STATUS PdhStatus;
    AnsiString respuesta;

    // preparamos la ruta al contador de rendimiento
```

```
	if (instancia != "")
		respuesta = "\\" + objeto + "(" + instancia + ")\\" + contador;
	else
		respuesta = "\\" + objeto + "\\" + contador;

	// comprobamos la ruta
	PdhStatus = PdhValidatePath((char*) respuesta.data());

	// si es incorrecta devolvemos la cadena vacía
	if (PdhStatus != ERROR_SUCCESS)
		respuesta = "";

	// si no, devolvemos la ruta
	return respuesta;
};

//---------------------------------------------------------------------------
```

Anexo 2.1: Explicación sobre una selección de ficheros del sistema de archivos */proc*

En esta sección del documento se comentan diversos datos sobre los archivos en */proc* más comúnmente utilizados. Si se desea más información sobre los datos presentados, puede acudirse a la referencia [RHM4]

/proc/apm

Este fichero proporciona información acerca del estado de Advanced Power Management (APM) y las opciones en el sistema. Esta información es usada por el kernel para proporcionar información para el comando apm.

La salida de este fichero en un sistema sin una bateria que está conectado a una fuente de energía AC se parece a la siguiente:

1.16 1.2 0x03 0x01 0xff 0x80 -1% -1 ?

Al ejecutar un comando apm -v en estos sistemas el resultado es parecido a:

APM BIOS 1.2 (kernel driver 1.16)
AC on-line, no system battery

Para estos sistemas, apm será capaz de hacer más que poner una máquina en modo standby, comúnmente se lo conoce como “poner el sistema a dormir”. El comando apm es mucho más útil en portátiles y otros sistemas Linux portables. Esto queda reflejado en los ficheros /proc/apm. Ésta es la salida de datos desde un fichero de muestra en un portátil que está ejecutando Linux, mientras que está conectado a una toma de corriente:

1.16 1.2 0x03 0x01 0x03 0x09 100% -1 ?

Cuando la misma máquina está desconectada de su fuente de energía y funcione con batería durante algunos minutos, el contenido del fichero apm cambiará:

1.16 1.2 0x03 0x00 0x00 0x01 99% 1792 min

En este estado, el comando apm presentaría estos datos:

APM BIOS 1.2 (kernel driver 1.16)
AC off-line, battery status high: 99% (1 day, 5:52)
[RHM4]

/proc/cmdline

Este fichero muestra los parámetros pasados al kernel de Linux en el momento en que éste es iniciado. Ejemplo de fichero /proc/cmdline:

ro root=/dev/hda2
[RHM4]

/proc/cpuinfo

Este fichero contiene información sobre el procesador o los procesadores instalados en el sistema. La información que se proporciona es:
· Número de identificación para cada procesador. Si el sistema sólo tiene instalado un procesador, éste tendrá asignado el número cero.
· Familia de CPU: Da de forma autorizada el tipo de procesador que tiene en el sistema. Ponga el número delante del "86" para calcular el valor. Esto le servirá de ayuda si se está preguntando sobre el tipo de arquitectura de un sistema antiguo (686, 586, 486 o 386). Ya que los paquetes RPM están compilados de forma ocasional para arquitecturas particulares, este valor le indica qué paquete instalar en el sistema.
· Nombre del modelo: Indica el nombre conocido del procesador, incluyendo el nombre de proyecto.
· Velocidad del procesador en megahercios
· Tamaño de caché: Indica la cantidad de nivel 2 de la caché de memoria disponible en el procesador.
· Flags de configuración: Define un número de cualidades diferentes del procesador, como la presencia de una unidad de coma flotante o la habilidad para procesar instrucciones MMX.
[RHM4]

/proc/devices
Este fichero visualiza los diversos dispositivos de caracteres y dispositivos de bloque actualmente configurados para el uso con el kernel. No incluye módulos que estén disponibles pero sin cargar en el kernel.

La salida de datos desde /proc/devices incluye el mayor número y nombre de dispositivos y se divide en dos secciones, una para los dispositivos de caracteres y otra para los dispositivos de bloque.

Los dispositivos de caracteres son parecidos a los dispositivos de bloque, a excepción de dos diferencias básicas.
- En primer lugar, los dispositivos de bloque disponen de un buffer para peticiones, permitiendo que éstas sean tratadas por orden. Esto es muy práctico con dispositivos diseñados para guardar información, tales como discos duros, porque la habilidad de ordenar la información antes de escribirla en el dispositivo permite que ésta se almacene de forma más eficiente. Los dispositivos de caracteres no necesitan dicha memoria.
- En segundo lugar, los dispositivos de bloque pueden enviar y recibir información en bloques de un tamaño particular, que pueden ser configurada para cumplir los requisitos de un dispositivo en particular. Los dispositivos de caracteres envían datos en los bytes necesarios, sin un tamaño preconfigurado.
[RHM4]

/proc/dma

Este fichero contiene una lista de los canales de acceso de memoria directos (DMA) ISA registrados en uso.
[RHM4]

/proc/execdomains

Este fichero lista los dominios de ejecución soportados en la actualidad por el kernel de Linux junto con la gama de personalidades que soportan.

Se piensa en los dominios de ejecución como en una clase de “personalidad” de un sistema operativo en particular. Otros formatos binarios, como Solaris, UnixWare y FreeBSD pueden usarse con Linux. Al cambiar la personalidad de una tarea ejecutada bajo Linux, un programador puede cambiar el modo en el que el sistema operativo trata las llamadas del sistema particulares desde un binario. A excepción del dominio de ejecución PER_LINUX, el resto pueden ser implementados como módulos cargables de forma dinámica.
[RHM4]

/proc/fb

Este fichero contiene una lista de dispositivos frame buffer, con el número del dispositivo frame buffer y el driver que lo controla.
[RHM4]

/proc/filesystems

Este fichero visualiza una lista de los tipos del sistema de ficheros actuales soportados por el kernel. A continuación tiene un ejemplo de salida de datos desde un fichero /proc/filesystems del kernel genérico:

```
nodev   rootfs
nodev   bdev
nodev   proc
nodev   sockfs
nodev   tmpfs
nodev   shm
nodev   pipefs
        ext2
nodev   ramfs
        iso9660
nodev   devpts
        ext3
nodev   autofs
nodev   binfmt_misc
```

La primera columna implica que el sistema de ficheros está montado en un dispositivo de bloque, con aquellos que contienen nodev en dicha columna implicando que éstos no están montados en un dispositivo de bloque. La segunda columna lista el nombre de los sistemas de ficheros soportados.

El comando mount utiliza la información para pasar por los posibles sistemas de ficheros cuando uno no está especificado como un argumento.
[RHM4]

/proc/interrupts

Este ficheros graba el número de interrupciones por IRQ en la arquitectura x86. Ejemplo de un fichero interrupts estándar:

```
        CPU0
0:      80448940    XT-PIC timer
1:      174412      XT-PIC keyboard
2:      0           XT-PIC cascade
8:      1           XT-PIC rtc
10:     410964      XT-PIC eth0
12:     60330       XT-PIC PS/2 Mouse
14:     1314121     XT-PIC ide0
15:     5195422     XT-PIC ide1
NMI:    0
ERR:    0
```

Para una máquina multiprocesadora, el fichero aparecerá de forma diferente:

```
        CPU0        CPU1
0:      1366814704  0           XT-PIC timer
1:      128         340         IO-APIC-edge keyboard
2:      0           0           XT-PIC cascade
8:      0           1           IO-APIC-edge rtc
12:     5323        5793        IO-APIC-edge PS/2 Mouse
13:     1           0           XT-PIC fpu
16:     11184294    15940594    IO-APIC-level Intel EtherExpress Pro 10/100 Ethernet
20:     8450043     11120093    IO-APIC-level megaraid
30:     10432       10722       IO-APIC-level aic7xxx
31:     23          22          IO-APIC-level aic7xxx
NMI:    0
ERR:    0
```

La primera columna se refiere al número de IRQ. Cada CPU del sistema tiene su propia columna y su propio número de interrupciones por IRQ. La columna siguiente le indica el tipo de interrupción y la última contiene el nombre del dispositivo que está localizado en ese IRQ.

Cada uno de los tipos de interrupciones vistos en este fichero, específicos para el tipo de arquitectura, significan algo diferente. Los siguientes valores son comunes para las máquinas x86:

· XT-PIC: Interrupciones del ordenador AT antiguo que se han producido por un largo periodo de tiempo.

· IO-APIC-edge: Señal de voltaje de las transacciones interrumpidas de abajo arriba, creando una edge, en la que la interrupción IO-APIC-level, tan sólo se dan a partir de procesadores 586 y superiores.

· IO-APIC-level: Genera interrupciones cuando su señal de voltaje se alza hasta que la señal desciende de nuevo.

[RHM4]

/proc/iomem

Este fichero muestra el mapa actual de la memoria del sistema para los diversos dispositivos:

```
00000000-0009fbff : System RAM
0009fc00-0009ffff : reserved
000a0000-000bffff : Video RAM area
000c0000-000c7fff : Video ROM
000f0000-000fffff : System ROM
00100000-07ffffff : System RAM
00100000-00291ba8 : Kernel code
00291ba9-002e09cb : Kernel data
e0000000-e3ffffff : VIA Technologies, Inc. VT82C597 [Apollo VP3]
e4000000-e7ffffff : PCI Bus #01
e4000000-e4003fff : Matrox Graphics, Inc. MGA G200 AGP
e5000000-e57fffff : Matrox Graphics, Inc. MGA G200 AGP
e8000000-e8ffffff : PCI Bus #01
e8000000-e8ffffff : Matrox Graphics, Inc. MGA G200 AGP
ea000000-ea00007f : Digital Equipment Corporation DECchip 21140 [FasterNet]
ea000000-ea00007f : tulip
ffff0000-ffffffff : reserved
```

La primera columna muestra los registros de memoria utilizados para cada uno de los diferentes tipos de memoria. La segunda columna indica el tipo de memoria de dichos registros. En particular, esta columna le mostrará qué registros de memoria son usados por el kernel den del sistema RAM o, si tiene puertos Ethernet múltiples en su NIC, los registros de memoria para cada puerto.
[RHM4]

/proc/ioports

De forma similar a /proc/iomem, /proc/ioports proporciona una lista de puertos registrados actualmente utilizados para la comunicación de entrada y salida con un dispositivo. Este fichero puede ser muy largo y empezaría de la siguiente manera:

```
0000-001f : dma1
0020-003f : pic1
0040-005f : timer
0060-006f : keyboard
0070-007f : rtc
0080-008f : dma page reg
00a0-00bf : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : ide1
01f0-01f7 : ide0
02f8-02ff : serial(auto)
0376-0376 : ide1
```

```
03c0-03df : vga+
03f6-03f6 : ide0
03f8-03ff : serial(auto)
0cf8-0cff : PCI conf1
d000-dfff : PCI Bus #01
e000-e00f : VIA Technologies, Inc. Bus Master IDE
e000-e007 : ide0
e008-e00f : ide1
e800-e87f : Digital Equipment Corporation DECchip 21140 [FasterNet]
e800-e87f : tulip
```

La primera columna indica la extensión de la dirección del puerto de entrada y salida para el dispositivo listado en la segunda columna.
[RHM4]

/proc/isapnp

Este fichero lista las tarjetas Plug and Play (PnP) en espacios ISA del sistema. Esto es mucho más habitual con las tarjetas de sonido incluyendo, en ese caso, cualquier número de dispositivos.

Este fichero puede ser bastante largo dependiendo del número de dispositivos y de los requisitos o peticiones de recursos.
Cada tarjeta lista su nombre, número de versión PnP y versión del producto. Si el dispositivo está activo y configurado, este fichero revelará el puerto y los números de IRQ para el dispositivo. Además, para asegurar una mejor compatibilidad, la tarjeta especificará los valores preferred y acceptable para un número de parámetros diferentes. El objetivo es el de permitir que las tarjetas PNP funcionen en base a otra y evitar los problemas de IRQ y puertos.
[RHM4]

/proc/kcore

Este fichero representa la memoria física del sistema y se almacena en el formato del fichero central. A diferencia de la mayoría de ficheros /proc, kcore muestra un tamaño. Este valor se da en bytes y es igual al tamaño de la memoria física (RAM) utilizada más 4KB.

Advertencia:
Evite visualizar el fichero kcore en file /proc. Los contenidos de este fichero se saldrán del terminal. Si accidentalmente lo visualiza, pulse [Intro]-[C] para parar el proceso y luego escriba reset para volver a la línea de comandos del prompt en la estaba.
Sus contenidos están diseñados para que los examine un depurador, como por ejemplo gdb, el depurador de GNU.
[RHM4]

/proc/ksyms

Este fichero contiene las definiciones del símbolo exportado del kernel usadas por las herramientas de módulos para enlazar dinámicamente módulos cargables.

```
e003def4    speedo_debug      [eepro100]
e003b04c    eepro100_init     [eepro100]
e00390c0    st_template       [st]
e002104c    RDINDOOR          [megaraid]
e00210a4    callDone          [megaraid]
e00226cc    megaraid_detect   [megaraid]
```

La segunda columna se refiere al nombre de una función del kernel y la primera columna lista la dirección de la memoria para dicha función. La última columna revela el nombre del módulo cargado para proporcionar dicha función.
[RHM4]

/proc/loadavg

Este fichero ofrece una vista preliminar al promedio de carga o al uso del procesador, del mismo modo que ofrece datos adicionales utilizados por uptime y otros comandos. El contenido puede ser parecido a este:

```
0.20  0.18  0.12  1/80  11206
```

Las primeras tres columnas miden el uso de una CPU en los últimos periodos de 1, 5 y 10 minutos. La cuarta columna muestra el número de procesos en ejecución en la actualidad y el número total de los mismos. La última columna visualiza la última ID de proceso usada.
[RHM4]

/proc/locks

Estos ficheros muestran los ficheros bloqueados en la actualidad por el kernel. El contenido de este fichero contiene datos internos de depuración y puede variar enormemente, dependiendo del uso del sistema. Los datos que aparecen en el archivo son, además de datos para depuración, los siguientes:

· Número único del bloqueo.

· Clase de bloqueo utilizado.

· Indicación de si el bloqueo no impide que otras personas puedan acceder a los datos y tan sólo previene de otros intentos de bloqueo o bien si mientras dura el bloqueo no se permite ningún otro acceso a los datos.

· Indicación de si el bloqueo permite al responsable del mismo el acceso de lectura o escritura al fichero

· ID del proceso de bloqueo.

· ID del fichero bloqueado.

· Inicio y el final de la región bloqueada del fichero.

[RHM4]

/proc/mdstat

Este fichero contiene información actual sobre las configuración del disco múltiple, de RAID.
[RHM4]

/proc/meminfo

Éste fichero contiene mucha información importante sobre el uso actual de RAM en el sistema. Un sistema con 256MB de RAM y 384MB de espacio swap tendrá un fichero /proc/meminfo parecido al siguiente:

```
        total:    used:    free:  shared: buffers:  cached:
Mem:  261709824 253407232  8302592       0 120745984 48689152
Swap: 402997248      8192 402989056
MemTotal:    255576 kB
MemFree:       8108 kB
MemShared:        0 kB
Buffers:     117916 kB
Cached:       47548 kB
Active:      135300 kB
Inact_dirty:  29276 kB
Inact_clean:    888 kB
Inact_target:     0 kB
HighTotal:        0 kB
HighFree:         0 kB
LowTotal:    255576 kB
LowFree:       8108 kB
SwapTotal:   393552 kB
SwapFree:    393544 kB
```

El comando top utiliza la mayoría de información. De hecho, la salida de datos del comando free es parecida, aparentemente, al contenido y estructura de meminfo. Si lee directamente meminfo conocerá muchos detalles sobre la memoria:
· Mem: Muestra el estado actual de RAM física en el sistema, incluyendo el uso en bytes de memoria total usada, libre, compartida, buffer y caché.
· Swap: Muestra la cantidad total de espacio swap libre y usado en bytes.
· MemTotal: Cantidad total de RAM física en kilobytes.
· MemFree: Cantidad de RAM física, en kilobytes, sin utilizar por el sistema.
· MemShared: No se utiliza con 2.4 y kernels superiores pero se deja por motivos de compatibilidad con versiones del kernel precedentes.
· Buffers: Cantidad de RAM física, en kilobytes, usada para los ficheros de buffers.
· Cached: Cantidad de RAM física en kilobytes usada como memoria caché.
· Active: Cantidad total de buffer o memoria caché de página, en kilobytes, que está en uso activo.
· Inact_dirty: Cantidad total de buffer y páginas de la caché, en kilobytes, que podrían quedar libres.
· Inact_clean: Cantidad total de buffer y páginas de la caché en kilobytes que definitivamente están libres y disponibles.

· Inact_target: Cantidad de red de asignaciones por segundo, en kilobytes, con un promedio de un minuto.
· HighTotal y HighFree: Cantidad total de memoria libre, que no está implantada en el espacio del kernel. El valor HighTotal puede variar dependiendo del tipo de kernel utilizado.
· LowTotal y LowFree: Cantidad total de memoria libre implantada directamente en el espacio del kernel. El valor LowTotal puede cambiar dependiendo del tipo de kernel utilizado.
· SwapTotal: Cantidad total de swap disponible, en kilobytes.
· SwapFree: Cantidad total de swap libre, en kilobytes.
[RHM4]

/proc/modules

Este fichero visualiza una lista de todos los módulos que han sido cargados en el sistema. Su contenido variará dependiendo de la configuración y uso de su sistema.

El fichero contiene:

· Nombre del módulo.

· Tamaño de memoria del módulo en bytes.

· Si el módulo está cargado o descargado en la actualidad.

· Si el módulo puede descargarse por si mismo automáticamente tras un periodo en el que no ha usado o si no se está utilizando.

· Si un módulo depende de que otro esté presente para que funcione.
[RHM4]

/proc/mounts

Este fichero proporciona una lista rápida de todos los montajes en uso del sistema. El archivo contiene los siguientes datos:

· Dispositivo montado.

· Punto de montaje.

· Tipo de sistema de ficheros

· Si está montado en modo sólo lectura o sólo escritura.
[RHM4]

/proc/mtrr

Este fichero se refiere a la memoria actual Memory Type Range Registers (MTRRs) en uso con el sistema. Si la arquitectura de su sistema soporta MTRRs, su mtrr será algo parecido a:

```
reg00: base=0x00000000 ( 0MB), size= 64MB: write-back, count=1
```

Los MTRRs se usan con procesadores de tipo Intel P6 (Pentium Pro y superiores) para controlar el acceso del procesador a la capacidad de la memoria. Cuando utilice una tarjeta de vídeo en un PCI o un bus AGP, un fichero mtrr adecuadamente configurado puede incrementar la ejecución en un 150%.
[RHM4]

/proc/partitions

La mayoría de la información no es relevante para los usuarios, a excepción de las siguientes líneas:
· major: Número principal del dispositivo con esta partición.
· minor: Número menor del dispositivo con esta partición. Separa las particiones en diferentes dispositivos físicos y los relaciona con el número al final del nombre de la partición.
· #blocks: Lista el número de bloques de disco físicos contenidos en una partición particular.
· name: Nombre de la partición.
[RHM4]

/proc/pci

El fichero contiene una lista completa de cada dispositivo PCI de su sistema. Dependiendo del número de dispositivos PCI que posea, /proc/pci puede ser bastante largo.

Esta salida de datos muestra una lista de todos los dispositivos PCI, en orden de bus, dispositivo y función. Además de proporcionar el nombre y versión del dispositivo, que va muy bien saber si no se acuerda de la marca de su tarjeta de interfaz de red, esta lista le proporciona información de IRQ detallada para que pueda dar con los problemas rápidamente.
[RHM4]

/proc/slabinfo

Este fichero contiene información sobre el uso de memoria a nivel slab. Los kernels de Linux superiores a 2.2 utilizan slab pools para gestionar la memoria superior al nivel de la página. Los objetos utilizados habitualmente, tienen sus propios slab pools.

Los valores en este fichero acontecen en el siguiente orden: nombre de la caché, nombre de objetos activos, nombre de objetos totales, tamaño del objeto, nombre de slabs activos (bloques) de los objetos, número total de slabs de los objetos y el número de páginas por slab.

Cabe remarcar que la palabra activo en este caso significa estar en uso. Un objeto activo es un objeto en uso y un slab activo es el que contiene cualquier objeto usado.
[RHM4]

/proc/stat

Este fichero le aporta diferentes estadísticas sobre el sistema desde que fue reiniciado por última vez.

Algunas de las estadísticas más usadas incluyen:

· cpu: Mide el número de jiffies (centésimas de segundo) que el sistema ha estado en modo usuario, modo usuario con prioridad baja, modo del sistema y tarea inactiva respectivamente. El total de todas las CPUs se da al inicio y cada CPU individual se lista debajo con sus propias estadísticas.
· page: Número de páginas que el sistema ha cargado o suprimido del disco.
· swap: Número de páginas swap que el sistema ha introducido o sacado.
· intr: Número de interrupciones que ha experimentado el sistema.
· btime: Tiempo de arranque, medido por el número de segundos desde el 1 de enero de 1970, conocido con el nombre de epoch.
[RHM4]

/proc/swaps

Este fichero mide el espacio swap y su uso. Mientras que alguna de esta información se puede encontrar en otros ficheros /proc/swaps proporciona una instantánea rápida de cada nombre de fichero swap, tipo de espacio swap y el tamaño total usado (en kilobytes). La columna prioritaria es útil cuando múltiples ficheros swap están en uso y se prefieren algunos de ellos a otros, como si estuvieran en discos duros más rápidos. Cuanto más baja es la prioridad, más probable es que se use un fichero swap.
[RHM4]

/proc/uptime

El fichero contiene información sobre el tiempo que lleva encendido el sistema desde el último reinicio. La salida de datos de /proc/uptime consiste en dos números. El primero indica el número total de segundos que el sistema ha estado en funcionamiento. El segundo indica cuánto de ese tiempo, la máquina ha estado inactiva.
[RHM4]

Anexo 2.2: Explicación sobre una selección de subdirectorios del sistema de archivos */proc*

En esta sección se comentan ciertas características de los directorios utilizados con más frecuencia.
[RHM4]

Directorios de proceso

Cada directorio /proc/ contiene unos cuantos directorios nombrados con un número. Un listado de los mismos podría empezar de la siguiente manera:

```
dr-xr-xr-x 3 root    root    0 Feb 13 01:28 1
dr-xr-xr-x 3 root    root    0 Feb 13 01:28 1010
dr-xr-xr-x 3 xfs     xfs     0 Feb 13 01:28 1087
dr-xr-xr-x 3 daemon  daemon  0 Feb 13 01:28 1123
dr-xr-xr-x 3 root    root    0 Feb 13 01:28 11307
dr-xr-xr-x 3 apache  apache  0 Feb 13 01:28 13660
dr-xr-xr-x 3 rpc     rpc     0 Feb 13 01:28 637
dr-xr-xr-x 3 rpcuser rpcuser 0 Feb 13 01:28 666
```

A estos directorios se les llama directorios de proceso, ya que pueden hacer referencia a un ID de proceso y contener información específica para ese proceso. El propietario y grupo de cada directorio de proceso está configurado para que el usuario ejecute el proceso. Cuando se finaliza el proceso, el directorio del proceso /proc desaparece. Sin embargo, mientras que se está ejecutando el proceso, una gran cantidad de información específica a ese proceso está contenida en varios ficheros del directorio de procesos.
Cada uno de los directorios de procesos contiene los siguientes ficheros:
· cmdline: Contiene los argumentos de la línea de comandos que iniciaron el proceso.
· cpu: Proporciona información específica sobre el uso de cada uno de las CPUs del sistema.
· cwd: Enlace al directorio actual en funcionamiento para el proceso.
· environ: Le da una lista de variables de entorno para el proceso. La variable de entorno en mayúsculas y el valor en minúsculas.
· exe: Enlace al ejecutable de este proceso.
· fd: Directorio que contiene todos los descriptores de ficheros para un proceso en particular. Vienen dados en enlaces numerados.
· maps: Contiene mapas de memoria para los diversos ejecutables y ficheros de librería asociados con este proceso. Este fichero puede ser bastante largo, dependiendo de la complejidad del proceso.
· mem: Memoria del proceso.
· root: Enlace al directorio root del proceso.
· stat: Estado del proceso.
· statm: Estado de la memoria en uso por el proceso.
· status: Proporciona el estado del proceso en una forma mucho más legible que stat o statm. Además del nombre y el ID del proceso, tiene a su disposición el estado (por ejemplo S (sleeping) o R (running)) y el ID de usuario/grupo al ejecutar el proceso, así como datos muy detallados sobre el uso de la memoria.

[RHM4]

/proc/self

El directorio /proc/self es un enlace al proceso en ejecución. Esto le permite verse a si mismo sin tener que conocer su ID de proceso.
[RHM4]

/proc/bus/

Este directorio contiene información específica sobre los diversos buses disponibles en el sistema.

Los contenidos de los subdirectorios y ficheros disponibles varían según la configuración del sistema. No obstante, cada uno de los directorios para cada uno de los tipos de bus contiene al menos un directorio para cada bus de ese tipo. Estos directorios bus individuales, habitualmente comunicados con números, tales como 00 contienen ficheros binarios que se refieren a los varios dispositivos disponibles en ese bus.

El directorio /proc/bus/usb contiene ficheros que siguen los diferentes dispositivos en cualquier bus USB, así como los controladores requeridos para su uso. El directorio 001 contiene todos los dispositivos del primer bus USB.
[RHM4]

/proc/driver/

Este directorio contiene información para drivers específicos que el kernel está utilizando. Un fichero habitual aquí es rtc, que proporciona salida desde el driver para el Real Time Clock (RTC) del sistema, dispositivo que guarda el tiempo mientras que el sistema está apagado.
[RHM4]

/proc/ide/

Este directorio contiene información sobre los dispositivos IDE del sistema. Cada canal IDE está representado como un directorio separado, como /proc/ide/ide0 y /proc/ide/ide1. Además, un fichero drivers también está disponible.

Muchos chipsets proporcionan un fichero de información en este directorio que aporta datos adicionales referentes a las unidades conectadas a través de varios canales.

Al navegar en un directorio para un canal IDE, como ide0 para el primer canal, obtendrá más información. El fichero channel proporciona el número de canal, mientras que model le facilita el tipo de bus para el canal (como pci).
[RHM4]

Directorios de dispositivo

Cada directorio del canal IDE es un directorio de dispositivo. El nombre del directorio del dispositivo corresponde a la letra del dispositivo en el directorio /dev.

Cada dispositivo, como un disco duro o un CD-ROM, tendrá en ese canal su propio directorio en el que están incluidas su propia recopilación de información y estadísticas. Los contenidos de esos directorios varían de acuerdo con el tipo de dispositivo conectado. Algunos de los ficheros más útiles y habituales en diferentes dispositivos incluyen:

· cache: La caché del dispositivo.
· capacity: La capacidad del dispositivo, en bloques de 512 bytes.
· driver: El driver y la versión usados para controlar el dispositivo.
· geometry: La geometría física y lógica del dispositivo.
· media: El tipo de dispositivo, como por ejemplo un disk.
· model: El nombre del modelo del dispositivo.
· settings: Recopilación de parámetros actuales del dispositivo. Este fichero habitualmente contiene bastante información técnica útil.

[RHM4]

/proc/irq/

Este directorio se usa para configurar la afinidad de una IRQ con una CPU, lo que le permite conectar una IRQ particular a una sola CPU. De manera alternativa, puede evitar que una CPU manipule cualquier IRQ.

Cada IRQ tiene su propio directorio, permitiendo que cada IRQ sea configurada de forma diferente al otro. El fichero /proc/irq/prof_cpu_mask es una máscara de bits que contiene los valores predeterminados para el fichero smp_affinity en el directorio de IRQ. Los valores en smp_affinity especifican qué CPUs manipulan esa IRQ en particular.

[RHM4]

/proc/net/

El directorio proporciona una visión de diversos parámetros y estadísticas de red. Cada uno de los ficheros cubre una cantidad específica de información relacionada con la red en el sistema:

· arp: Contiene la tabla del kernel ARP. Este fichero es particularmente útil para conectar la dirección del hardware a una dirección IP de un sistema.
· atm: Directorio que contiene ficheros con diversas configuraciones y estadísticas Asynchronous Transfer Mode (ATM) y las tarjetas ASDL.
· dev: Lista los diferentes dispositivos de red configurados en en sistema, completa con transmisión y recibe estadísticas. Este fichero le indica el número de paquetes de entrada y de salida, el número de errores, el número de paquetes eliminados, etc.
· dev_mcast: Visualiza los diversos grupos Layer2 con destinatario múltiple a los que cada dispositivo escucha.
· igmp: Lista las direcciones IP con destinatarios múltiples a las que el sistema se ha incorporado.

· ip_fwchains: Revela cualquier cadena del firewall actual.
· ip_fwnames: Lista todos los nombres de cadenas del firewall.
· ip_masquerade: Proporciona una tabla de información de masquerading.
· ip_mr_cache: Lista de la caché de routing de múltiple destinatario.
· ip_mr_vif: Lista de las interfaces virtuales de múltiple destinatario.
· netstat: Contiene una amplia colección de estadísticas de red, incluyendo la temporización TCP, los cookies enviados y recibidos y mucho más.
· psched: Lista de parámetros de planificación global del paquete.
· raw: Lista de estadísticas de dispositivo raw.
· route: Visualiza la tabla de ruta del kernel.
· rt_cache: Contiene la caché de ruta actual.
· snmp: Lista de los datos del protocolo Simple Network Management Protocol (SNMP) para varios protocolos de red en uso.
· sockstat: Proporciona estadísticas de socket.
· tcp: Contiene información detallada del socket TCP.
· tr_rif: Tabla de routing RIF del token ring.
· udp: Contiene información detallada del socket UDP.
· unix: Lista sockets de dominio UNIX.
· wireless: Lista datos de la interfaz de radio.
[RHM4]

/proc/scsi/

Del mismo modo el directorio /proc/ide solo existe si un controlador IDE está conectado al sistema, el directorio /proc/scsi sólo está disponible si tiene un adaptador de host SCSI. El fichero primario aquí es /proc/scsi/scsi, que contiene una lista de cada dispositivo SCSI reorganizado.
Cada driver SCSI usado por el sistema tiene su propio directorio en /proc/scsi que contiene ficheros específicos para cada controlador SCSI que utiliza aquel driver. Los ficheros en cada uno de los directorios contienen típicamente extensión de dirección de entrada y de salida, IRQ y estadísticas par el controlador especial SCSI que utiliza el driver. Mientras que cada controlador remite un tipo y cantidad de información diferente, la salida de datos de la mayoría de estos ficheros resulta relativamente de fácil comprensión para un humano.
[RHM4]

/proc/sys/

Este directorio es especial y diferente de los otros directorios en /proc, ya que no sólo proporciona mucha información sobre el sistema sino que también le permite hacer cambios en la configuración de un kernel en ejecución.

El directorio /proc/sys contiene directorios diferentes que controlan diferentes aspectos de la ejecución de un kernel.
[RHM4]

/proc/sys/dev/

Este directorio proporciona parámetros para dispositivos particulares en el sistema. La mayoría de sistemas tienen al menos dos directorios cdrom y raid, sin embargo, los kernels personalizados pueden tener otros, como por ejemplo parport, que proporciona la habilidad de compartir un puerto paralelo entre drivers de dispositivo múltiple.

El directorio cdrom contiene un fichero llamado info, que revela algunos parámetros importantes de CD-ROM.

Este fichero se puede escanear con la finalidad de descubrir las cualidades de un CD-ROM desconocido, por lo menos para el kernel. Si tiene a su disposición múltiples CD-ROMs en un sistema, cada dispositivo tendrá su propia columna de información.

Se pueden utilizar diversos ficheros en /proc/sys/dev/cdrom, como autoclose y checkmedia, para controlar el CD-ROM del sistema.

Si se se compila el soporte RAID en el kernel, tendrá a su disposición un directorio /proc/sys/dev/raid con, al menos, dos ficheros dentro del mismo: speed_limit_min y speed_limit_max. Estas configuraciones se dan para aminorar o acelerar la velocidad en que se utiliza el dispositivo RAID para tareas intensivas de entrada y salida en particular, tales como la resincronización de los discos.
[RHM4]

/proc/sys/fs/

Este directorio contiene un compendio de opciones y de información referente a varios aspectos del sistema de ficheros.

Los ficheros importantes en /proc/sys/fs incluyen:
· dentry-state: Proporciona el estado de la caché del directorio.
· dquot-nr: Muestra el número máximo de entradas de cuota de disco cacheado.
· file-max: Permite cambiar el número máximo de manipulación de ficheros que asignará el kernel. Si alza el valor de este fichero resolverá los errores causados por la falta de manipulación de ficheros disponibles.
· file-nr: Visualiza el número de manipulación de ficheros asignados, manipulación de ficheros usados, así como el número máximo de manipulación de ficheros, en este orden.
· overflowgid y overflowuid: Define el ID de grupo establecido y el ID de usuario, respectivamente, para el uso con el sistema de ficheros que tan sólo soporta los IDs de grupo y usuario de 16 bits.
· super-max: Controla el número máximo de superbloques disponible.
· super-nr: Visualiza el número actual de superbloques en uso.
[RHM4]

/proc/sys/kernel/

Este directorio contiene una variedad de ficheros de configuración diferentes que afectan directamente a la operación del kernel. Algunos de los ficheros más importantes incluyen:

· acct: Controla la suspensión del proceso de contabilización basado en el porcentaje de espacio libre disponible en el sistema de ficheros que contienen el log.
· cap-bound: Controla las configuraciones de las opciones de capacidad de bounding que proporcionan una lista de capacidades que cualquier proceso del sistema puede realizar. Si una capacidad no está listada aquí, ningún proceso, por muy privilegiado que sea éste, puede realizarlo. La idea inicial es hacer que el sistema sea más seguro asegurando que no acontezcan ciertas cosas, por lo menos llegados a un cierto nivel del proceso de arranque.
· ctrl-alt-del: Controla que [Ctrl]-[Alt]-[Delete] reinicien el ordenador mediante el uso de init o fuercen un rearranque inmediato.
· domainname: Le permite configurar el nombre de dominio del sistema, como domain.com.
· hostname: Le permite configurar el nombre del host del sistema, como host.domain.com.
· hotplug: Configura la utilidad para que ésta sea usada cuando se detecta un cambio en la configuración del sistema. Principalmente se usa con USB y Cardbus PCI.
· modprobe: Fija la ubicación del programa a usar para cargar los módulos del kernel cuando éstos sean necesarios. El valor por defecto de /sbin/modprobe significa que kmod lo solicitará para cargar el módulo cuando un thread del kernel solicite kmod.
· msgmax: Fija el tamaño máximo de cualquier mensaje enviado desde un proceso a otro, que está fijado en 8192 bytes por defecto. Debería tener cuidado al alzar este valor, ya que los mensajes en cola entre procesos están almacenados en una memoria de kernel sin memoria de intercambio y cualquier aumento en msgmax incrementará los requisitos de RAM para el sistema.
· msgmnb: Establece el número máximo de bytes en una única cola de mensajes. Por defecto, 16384.
· msgmni: Establece el número máximo de identificadores de la cola de mensajes. Por defecto, 16.
· osrelease: Lista el número de versión del kernel de Linux. Este fichero tan sólo puede ser alterado al cambiar la fuente del kernel y recompilarla.
· ostype: Visualiza el tipo de sistema operativo. Por defecto, este fichero está configurado para Linux y este valor tan sólo puede ser cambiado al cambiar la fuente del kernel y recompilarla.
· overflowgid y overflowuid: Define el ID de grupo establecido y el ID de usuario, respectivamente, para el uso con llamadas del sistema a arquitecturas que tan sólo soportan IDs de grupo y usuario de 16 bits.
· panic: Define el número de segundos que el kernel pospone el arranque del sistema cuando se experimenta una emergencia en el kernel. Por defecto, el valor está establecido en 0, lo que deshabilita el rearranque automático tras una emergencia.
· printk: Este fichero controla una variedad de configuraciones relacionadas con la impresión o los mensajes de error de registro. Cada mensaje de error remitido por el kernel tiene un loglevel asociado a éste que define la importancia del mensaje. Los valores de loglevel aparecen en el orden siguiente:
o 0: Emergencia del Kernel. No se puede utilizar el sistema.
o 1: Alerta del kernel. Se debe actuar inmediatamente.
o 2: La condición del kernel se considera crítica.
o 3: Condición de error general del kernel.
o 4: Condición de aviso general del kernel.
o 5: Nota del kernel de una condición normal pero significativa.
o 6: Mensaje informativo del kernel.

o 7: Mensajes de depuración del kernel.

En el fichero printk aparecen cuatro valores. Cada uno de estos valores define una regla diferente para tratar con los mensajes de error. El primer valor, llamado console loglevel, define la prioridad más baja de mensajes que se imprimirán en la consola. El segundo valor establece el loglevel por defecto para mensajes adjuntos a éstos sin un loglevel explícito. El tercer valor establece la configuración del loglevel lo más bajo posible para el loglevel de la consola. El último valor establece el valor por defecto para el loglevel de la consola.

· rtsig-max: Configura el número máximo de señales en tiempo real POSIX que el sistema podría haber puesto en cola en cualquier otro momento. El valor por defecto es 1024.

· rtsig-nr: El número actual de señales POIX en tiempo real que el kernel ha puesto en cola.

· sem: Este fichero establece las configuraciones del semáforo interior del kernel.

· shmall: Establece la cantidad total de memoria que se puede utilizar de una sola vez en el sistema, en bytes. Por defecto, este valor es 2097152.

· shmmax: Establece el mayor tamaño de segmento de memoria compartida que permite el kernel, en bytes. Por defecto, este valor es 33554432. No obstante, el kernel soporta valores con mucho más margen.

· shmmni: Establece el número máximo de segmentos de memoria compartida para todo el sistema. Por defecto, este valor es 4096.

· threads-max: Establece el número máximo de threads que puede usar el kernel, con un valor por defecto de 2048.

· version: Visualiza la fecha y la hora en los que el kernel fue completado por última vez. El primer campo en este fichero, está relacionado con el número de veces que se ha construido un kernel desde la base de la fuente.

[RHM4]

/proc/sys/net

Este directorio contiene diversos directorios que tratan cuestiones acerca de red, incluyendo protocolos variados y centros de énfasis. La disponibilidad de diferentes directorios es posible gracias a algunas configuraciones realizadas durante la compilación del kernel, tales como appletalk, ethernet, ipv4, ipx y ipv6. Dentro de estos directorios, podrá ajustar los diversos valores de red para esa configuración en un sistema en funcionamiento.

Debido a la amplia variedad de posibles opciones de red disponibles con Linux y la gran cantidad de espacio requerido para comentarlas, tan sólo se comentarán los directorios /proc/sys/net más habituales.

El **directorio core** contiene una variedad de configuraciones que controlan la interacción entre el kernel y las capas de red. Los ficheros más importantes son:

· message_burst: Décimas de segundos requeridos para escribir un mensaje nuevo de aviso. Se usa para prevenir ataques Denial of Service (DoS) y la configuración por defecto es 50.

· message_cost: También se utiliza para prevenir ataques de DoS poniendo un coste a cada mensaje de aviso. Cuanto más alto es el valor de este fichero (por defecto 5), más probable es que el aviso del mensaje sea ignorado.

· netdev_max_backlog: Establece el número máximo de paquetes permitido para hacer cola cuando una interfaz en particular recibe paquetes a una velocidad superior a la que el kernel puede procesarlos. El valor por defecto para este fichero es 300.
· optmem_max: Configura el tamaño máximo de buffer secundario por socket.
· rmem_default: Establece el tamaño por defecto del buffer del socket de recepción en bytes.
· rmem_max: Establece el tamaño máximo del buffer de recepción en bytes.
· wmem_default: Establece el tamaño por defecto del buffer del socket de enviar en bytes.
· wmem_max: Establece el tamaño máximo de la buffer del socket de enviar en bytes.

Teniendo en cuenta lo extendido que está el uso de las redes IP con Linux, la consulta de los ficheros más importantes en **ipv4** revela configuraciones de red de gran potencia. Muchas de estas configuraciones, usadas conjuntamente con otras, resultan muy útiles para la prevención de ataques en su sistema.
Aquí tiene algunos de los ficheros más importantes en el directorio ipv4:
· icmp_destunreach_rate, icmp_echoreply_rate, icmp_paramprob_rate y icmp_timeexeed_rate: Establece el ratio máximo de paquetes ICMP a enviar, en centésimas de un segundo en sistemas Intel, para hosts bajo diversas condiciones.
· icmp_echo_ignore_all and icmp_echo_ignore_broadcasts: Permite que el kernel ignore paquetes ICMP ECHO desde cada host o tan sólo aquéllos que se originen desde direcciones broadcast y de destinatario múltiple, respectivamente.
· ip_default_ttl: Establece el Time To Live (TTL) predeterminado, que limita el número de saltos que un paquete puede efectuar antes de alcanzar su destino. Si incrementa este valor, la ejecución del sistema puede disminuir.
· ip_forward: Permite interfaces en el sistema para reenviar paquetes a otro. Por defecto, este fichero está fijado en 0 para deshabilitar el reenvío; sin embargo, si fija este fichero en 1 habilitará el reenvío.
· ip_local_port_range: Especifica la extensión de puertos a usar por TCP o UDP cuando es necesario un puerto local. El primer número es el puerto más bajo que puede utilizar, y el segundo especifica el puerto más alto. Cualquier sistema que se crea que necesitará más puertos que los predeterminados 1024 hasta 4999 debería usar la extensión 32768 hasta 61000S en este fichero.
· tcp_syn_retries: Proporciona un límite en el número de veces que se intenta acceder a una conexión.
· tcp_retries1: Establece el número de retransmisiones permitidas que intentan responder una conexión de entrada. 3 por defecto.
· tcp_retries2: Establece el número de retransmisiones permitidas de paquetes TCP. 15 por defecto.

Existe un número de otros directorios dentro del directorio /proc/sys/net/ipv4 que cubren temas más específicos. El directorio conf permite que cada una de las interfaces del sistema sea configurada de diferentes formas, incluyendo el uso de configuraciones por defecto para dispositivos sin configurar (en el subdirectorio default) y configuraciones que sobreescriben todas las configuraciones especiales (en el subdirectorio all).

Para controlar las conexiones entre “vecinos” directos, es decir, cualquier otro sistema directamente conectado a su sistema, el directorio neigh acepta configuraciones especiales para cada interfaz. Del mismo modo, también facilita la implantación de reglas estrictas para sistemas que están distanciados por varios saltos.
[RHM4]

/proc/sys/vm/

Este directorio facilita la configuración del subsistema de memoria virtual (VM) del kernel. El kernel hace un uso extensivo e inteligente de la memoria virtual, conocida comunmente como espacio swap.
Los siguientes ficheros se encuentran habitualmente en el directorio /proc/sys/vm:
· bdflush: Establece varios valores relacionados con el demonio del kernel bdflush.
· buffermem: Le permite controlar la cantidad porcentual de la memoria total del sistema a usar para la memoria del buffer. Los primeros y últimos valores establecen el porcentaje mínimo y máximo de memoria que ha de usarse como memoria de buffer, respectivamente. El valor del medio establece el porcentaje de memoria del sistema dedicado a la memoria de buffer donde el subsistema de gestión de memoria iniciará borrando la caché del buffer más que otros tipos de memoria para compensar la falta en general de memoria libre.
· freepages: Visualiza varios valores relacionados con páginas libres de memoria del sistema. El primer valor muestra el número mínimo de páginas libres permitido antes de que el kernel asuma el control de la asignación de la memoria adicional. El segundo valor da el número de páginas libres antes de que el kernel empiece hacer swap para preservar la ejecución. El tercer valor es el número de páginas libres que el sistema mantiene siempre disponibles.
· kswapd: Establece varios valores referentes al demonio de swap-out del kernel, kswapd. Este fichero tiene tres valores: El primer valor establece el número máximo de páginas que kswapd intentará liberar en una sola vez. El segundo valor establece el número mínimo de veces que kswapd intenta dejar libre una página. El tercer valor establece el número de páginas kswapd escribir en un solo intento. Una correcta sintonización de este valor final puede mejorar la ejecución de un sistema usando mucho espacio swap al hacer que el kernel escriba páginas en grandes cantidades, minimizando el número de búsquedas de disco.
· max_map_count: Configura el número máximo de áreas de mapa de memoria que puede tener un proceso.
· overcommit_memory: Contiene un valor, que cuando no es el predeterminado 0, permite al kernel saltarse un chequeo estándar para ver si existe suficiente memoria antes de asignarlo.
· pagecache: Controla la cantidad de memoria usada por la caché de la página. Los valores en pagecache son porcentajes y funcionan de manera parecida a buffermem para reforzar los mínimos y los máximos de memoria caché de página disponible.
· page-cluster: Establece el número de páginas leídas en un solo intento. El valor por defecto de 4 establecido en 16 páginas, es apropiado para la mayoría de los sistemas.
· pagetable_cache: Controla el número de tablas de página que están cacheadas en las bases para un procesador. Los primeros y segundos valores están relacionados con el número mínimo y máximo de tablas de página a establecer aparte, respectivamente.
[RHM4]

/proc/tty/

Este directorio contiene información sobre los dispositivos basados en caracteres, conocidos originalmente como dispositivos teletype.

En Linux existen tres tipos diferentes de dispositivos tty: dispositivos en serie, terminales virtuales y terminales pseudo.

· El fichero drivers es una lista de dispositivos tty actualmente en uso.

· El fichero /proc/tty/driver/serial lista las estadísticas en uso y el estado de cada una de las líneas de serie tty.

[RHM4]

Anexo 2.3: Implementación de la aplicación de acceso a los datos de rendimiento y archivos de */proc*

La aplicación, cuyo código aparece a continuación, ha sido escrita en lenguaje C, usando la aplicación para creación de interfaces gráficas Glade, y basándose en la tecnología de interfaces gráficas Gtk+ con librerías Glib.

```
///////CALLBACKS.C

#ifdef HAVE_CONFIG_H
#  include <config.h>
#endif

#include <pthread.h>

#include <gtk/gtk.h>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <fcntl.h>
#include <dirent.h>
#include <sys/stat.h>
#include <string.h>
#include "callbacks.h"
#include "interface.h"
#include "support.h"
////////////////////////////////////////////////////////////////////////////
char* contadores[BUFSIZ];       //almacena todas las entradas de /proc
char* contadores2[BUFSIZ];      //que se seleccionan para listar al
                                //pulsar "on_anadirEntrada_clicked".
                                //"contadores2" es auxiliar para copias

char* datos[BUFSIZ];      //almacena todos los datos de rendimiento y
char* datos2[BUFSIZ];     //archivos de /proc que se han seleccionado
char* datos3[BUFSIZ];     //para realizarles las mediciones al pulsar
                          //"on_anadir"clicked". "datos2" y "datos3" son
                          //auxiliares para copias.

int fd_destino;   //identifica el archivo de destino en el que se
                  //escriben las mediciones. lo pasa el usuario a través
                  //de la interfaz gráfica.

int mide = 0;      //indica durante qué intervalo de tiempo se realizan
                   //mediciones. se pone a uno al pulsar en "iniciar" y
                   //vuelve a cero al pulsar "parar".

int dormir;     //indica el tiempo que transcurre entre una toma de
```

```
                    //datos y la siguiente.
////////////////////////////////////////////////////////////////////////////
//definición de un nuevo tipo de datos struct "dato". representa a un
//dato de rendimiento, y sus atributos son:
//"ruta" almacena la ruta completa del archivo que contiene al dato.
//"nombre" es el nombre que el desarrollador da al dato.
//"tipo" representa el tipo de dato de rendimiento que es (1 ó 2 en
//la implementación presentada).
//"linea" indica la línea, dentro del archivo, en la que se encuentra
//almacenado el dato.
//"inicio" y "fin" indican, para el tipo 1, entre qué dos caracteres
//se encuentra exactamente el dato de rendimiento dentro de la línea.
//"cual" y "separador" indican, para el tipo 2, en qué token y en base
//a qué separador, se encuentra exactamente el dato de rendimiento.
typedef struct d{
        char* ruta;
        char* nombre;
        int tipo;
        int linea;
        int inicio;
        int fin;
        char* separador;
        int cual;
}dato;


//constructora del tipo de dato 1. rellena los atributos ruta, nombre,
//tipo(1), linea, inicio y fin.
dato nuevoDato1(char* ruta,char* nombre,int tipo,int linea,int inicio,int fin){
        dato d;
        d.ruta = ruta;
        d.nombre = nombre;
        d.tipo = tipo;
        d.linea = linea;
        d.inicio = inicio;
        d.fin = fin;
        return d;
}


//constructora del tipo de dato 2. rellena los atributos ruta, nombre,
//tipo(2), linea, separador y cual.
dato nuevoDato2(char* ruta,char* nombre,int tipo,int linea,char* separador,int cual){
        dato d;
        d.ruta = ruta;
        d.nombre = nombre;
        d.tipo = tipo;
        d.linea = linea;
        d.separador = separador;
        d.cual = cual;
```

```
	return d;
}

//accesor que devuelve el atributo "linea" del tipo de datos "dato".
int getLinea(dato d){
	return d.linea;
}


//accesor que devuelve el atributo "inicio" del tipo de datos "dato".
//válido para el tipo de datos 1.
int getInicio(dato d){
	return d.inicio;
}


//accesor que devuelve el atributo "fin" del tipo de datos "dato".
//válido para el tipo de datos 1.
int getFin(dato d){
	return d.fin;
}


//accesor que devuelve el atributo "cual" del tipo de datos "dato".
//válido para el tipo de datos 2.
int getCual(dato d){
	return d.cual;
}


//accesor que devuelve el atributo "separador" del tipo de datos "dato".
//válido para el tipo de datos 2.
char* getSeparador(dato d){
	return d.separador;
}


//accesor que devuelve el atributo "tipo" del tipo de datos "dato".
//en la implementación presentada, el valor debvuelto será 1 ó 2.
int getTipo(dato d){
	return d.tipo;
}


//accesor que devuelve el atributo "nombre" del tipo de datos "dato".
char* getNombre(dato d){
	return d.nombre;
}
```

```
//accesor que devuelve el atributo "ruta" del tipo de datos "dato".
char* getRuta(dato d){
	return d.ruta;
}


int numDatos = 0;			//"numDatos" indica el tamaño de
dato almacenDatos[BUFSIZ];	//"almacenDatos", que almacena todos
				//los datos de rendimiento estudiados
				//hasta el momento.


//método que añade un nuevo dato de rendimiento estudiado a la
//variable "almacenDatos" que los almacena a todos.
void anadeDato(dato almacenDatos[BUFSIZ],dato d){
	almacenDatos[numDatos] = d;
	numDatos++;
}


//método que inicializa la variable "almacenDatos" con todos los datos
//estudiados hasta el momento. cuando un desarrollador encuentre un
//nuevo dato, deberá definirlo aquí con la constructora correspondiente
//a su tipo, y añadirlo con "anadeDato". "inicializar" será llamado
//cada vez que se ejecute la aplicación.
void inicializar(){
dato procesos_en_ejecucion = nuevoDato2("/proc/loadavg","procesos en ejecucion /
procesos totales",2,1," ",4);
anadeDato(almacenDatos,procesos_en_ejecucion);
dato uso_CPU_un_min = nuevoDato2("/proc/loadavg","uso de CPU en ultimo
min",2,1," ",1);
anadeDato(almacenDatos,uso_CPU_un_min);
dato uso_CPU_cinco_min = nuevoDato2("/proc/loadavg","uso de CPU en ultimos 5
min",2,1," ",2);
anadeDato(almacenDatos,uso_CPU_cinco_min);
dato uso_CPU_diez_min = nuevoDato2("/proc/loadavg","uso de CPU en ultimos 10
min",2,1," ",3);
anadeDato(almacenDatos,uso_CPU_diez_min);
dato ultimo_PID_usado = nuevoDato2("/proc/loadavg","ultimo PID usado",2,1," ",5);
anadeDato(almacenDatos,ultimo_PID_usado);
dato jiffies_modo_usuario = nuevoDato2("/proc/stat","jiffies en modo usuario",2,1,"
",2);
anadeDato(almacenDatos,jiffies_modo_usuario);
dato jiffies_modo_usuario_prioridad_baja = nuevoDato2("/proc/stat","jiffies en modo
usuario con prioridad baja",2,1," ",4);
anadeDato(almacenDatos,jiffies_modo_usuario_prioridad_baja);
dato jiffies_modo_sistema = nuevoDato2("/proc/stat","jiffies en modo sistema",2,1,"
",5);
```

```
anadeDato(almacenDatos,jiffies_modo_sistema);
dato jiffies_modo_inactivo = nuevoDato2("/proc/stat","jiffies en modo inactivo",2,1,"
",6);
anadeDato(almacenDatos,jiffies_modo_inactivo);
dato interrupciones = nuevoDato2("/proc/stat","numero de interrupciones",2,3," ",2);
anadeDato(almacenDatos,interrupciones);
dato segundos_encendido = nuevoDato2("/proc/uptime","segundos encendido",2,1,"
",1);
anadeDato(almacenDatos,segundos_encendido);
dato segundos_inactivo = nuevoDato2("/proc/uptime","segundos inactivo",2,1," ",2);
anadeDato(almacenDatos,segundos_inactivo);
dato entradas_cache_inutilizadas = nuevoDato2("/proc/sys/fs/dentry-state","entradas
cache inutilizadas",2,1," ",2);
anadeDato(almacenDatos,entradas_cache_inutilizadas);
dato paginas_requeridas = nuevoDato2("/proc/sys/fs/dentry-state","paginas requeridas
por el sistema",2,1," ",4);
anadeDato(almacenDatos,paginas_requeridas);
dato kB_mem_RAM_total = nuevoDato2("/proc/meminfo","kB de memoria
RAM",2,1," ",2);
anadeDato(almacenDatos,kB_mem_RAM_total);
dato kB_mem_RAM_libre = nuevoDato2("/proc/meminfo","kB de memoria RAM
libre",2,2," ",2);
anadeDato(almacenDatos,kB_mem_RAM_libre);
dato kB_mem_swap_total = nuevoDato2("/proc/meminfo","kB de memoria
swap",2,12," ",2);
anadeDato(almacenDatos,kB_mem_swap_total);
dato kB_mem_swap_libre = nuevoDato2("/proc/meminfo","kB de memoria swap
libre",2,13," ",2);
anadeDato(almacenDatos,kB_mem_swap_libre);
dato kB_RAM_para_buffers = nuevoDato2("/proc/meminfo","kB de memoria RAM
para buffers",2,3," ",2);
anadeDato(almacenDatos,kB_RAM_para_buffers);
dato kB_RAM_para_cache = nuevoDato2("/proc/meminfo","kB de memoria RAM para
cache",2,4," ",2);
anadeDato(almacenDatos,kB_RAM_para_cache);
dato buffer_cache_activa = nuevoDato2("/proc/meminfo","kB de memoria cache y
buffers activa",2,6," ",2);
anadeDato(almacenDatos,buffer_cache_activa);
}


////////////////////////////////////////////////////////////////////////////
//método que recorre un directorio (/proc en nuestro caso), y lista
//todos los archivos presentes en aquellos subdirectorios que coincidan
//con alguna de las entradas almacenadas en "contadores". rellena el
//comboBox "cp" con las rutas de todos estos archivos.
void recorrer_arbol(char *dir,GtkWidget * cp,int a) {
        DIR *dirp;
           struct dirent *dp;
           struct stat buf;
```

```
        char fichero[256];
        int ok;
        int es;
        char* uno;

        // Abre el directorio dir, y obtiene un descriptor dirp
        // de tipo DIR
        if ((dirp = opendir(dir)) == NULL){}
else{

        // Recorre recursivamente los directorios por debajo del actual
        // Comienza leyendo una entrada del directorio
        while ((dp = readdir(dirp)) != NULL) {
                //prescinde de las entradas "." y ".." que producirían
                //un bucle sin fin si se consideraran
                if (!strcmp(".", dp->d_name)) continue;
                if (!strcmp("..", dp->d_name)) continue;
                int i=0;
           //copia de seguridad de "contadores", que se modificará
           //al llamar a "strtok".
                while (contadores[i]!='\0'){
                        contadores2[i]=contadores[i];
                        i++;
                }
                es = 0;
           //compara la entrada actual de /proc con las entradas
           //almacenadas en "contadores".
                uno=strtok(contadores2,"|");
                while(uno != NULL){
                        if (!strcmp(uno,dp->d_name)) {
                                es = 1;
                        }
                        uno = strtok(NULL,"|");
                }
                while (contadores2[i]!='\0'){
                        contadores2[i]="|";
                        i++;
                }
                if ((a==1)||(es==1)){
                        es = 0;
                        // Construye el nombre del fichero a partir
                        // del directorio dir
                sprintf(fichero, "%s/%s", dir, dp->d_name);
                // Averigua si la entrada corresponde a un
                //sub-directorio
                        ok = stat(fichero, &buf);
                        if ((ok != 1) && ((buf.st_mode & S_IFMT)==S_IFDIR))  {
                                recorrer_arbol(fichero,cp,1);
                        }
                else {
```

```
                        gtk_combo_box_append_text (cp, fichero);
                        }
                }
                else {}
        }
    }
}

//////////////////////////////////////////////////////////////////
//oyente de la interfaz gráfica para el botón "añadir". cada vez que
//se pulse, el archivo o dato de rendimiento que se encuentre
//seleccionado en el comboBox, se añadirá a la ventana de archivos y
//datos seleccionados, así como a la variable "datos", en
//la que iremos almacenando todos los archivos y datos sobre los que
//posteriormente haremos las capturas.
void
on_anadir_clicked                   (GtkButton       *button,
                                    gpointer         user_data)
{
        //puntero al comboBox "comboProc" de la interfaz gráfica.
        GtkWidget * cp = lookup_widget(GTK_WIDGET(button), "comboProc");
        //puntero al gtk_text_entry "datos" de la interfaz gráfica.
        GtkWidget * ventana = lookup_widget(GTK_WIDGET(button), "datos");
        //variable que almacena el texto presente alctualmente en el
        //comboBox "comboProc".
        gchar* texto = gtk_combo_box_get_active_text(cp);
        gtk_entry_append_text(ventana,texto);
        strcat(datos,texto);
        strcat(datos,"|");
        strcat(datos2,texto);
        strcat(datos2,"|");
        strcat(datos3,texto);
        strcat(datos3,"|");
}


//oyente de la interfaz gráfica para el botón "eliminar". al pulsarlo,
//tanto la ventana de los archivos y datos de rendimiento seleccionados
//como el array donde los estamos almacenando("datos"), quedan borrados.
void
on_eliminar_clicked                 (GtkButton       *button,
                                    gpointer         user_data)
{
        //puntero al gtk_text_entry "datos" de la interfaz gráfica.
        GtkWidget * ventana = lookup_widget(GTK_WIDGET(button), "datos");
        gtk_entry_set_text(ventana,"");
        int i=0;
        while (datos3[i]!='\0'){
                datos[i]=' ';
                datos2[i]=' ';
```

```
                datos3[i]=' ';
                i++;
                }
}


//oyente de la interfaz gráfica para el botón "iniciar", que provoca
//que comiencen las tomas de datos sobre los archivos y datos de
//rendimiento. abre el fichero destino de estas mediciones, pasado
//por el usuario a través de la interfaz gráfica, calcula el valor de
//"dormir" a partir de los datos que el usuario haya introducido por
//la interfaz gráfica, y llama al método "iniciarMedidas".
void
on_iniciar_clicked              (GtkButton       *button,
                                gpointer         user_data)
{
        mide = 1;
        char DecimalSeparator = '.';
        //puntero al gtk_text_entry "destino" de la interfaz gráfica.
        GtkWidget * dest = lookup_widget(GTK_WIDGET(button), "destino");
        //texto presente en el gtk_text_entry "destino".
        gchar* fichero = gtk_entry_get_text(dest);
        //identificador del fichero destino de las medidas.
        fd_destino = open(fichero, O_WRONLY|O_TRUNC|O_CREAT, 0666);
        gdouble numero;
        //puntero al gtk_spin_button "cuantos" de la interfaz gráfica.
        GtkWidget * cuantos = lookup_widget(GTK_WIDGET(button), "cuantos");
        numero = gtk_spin_button_get_value(cuantos);
        //puntero al comboBox "tiempo" de la interfaz gráfica.
        GtkWidget * tiempo = lookup_widget(GTK_WIDGET(button), "tiempo");
        //texto presente actualmente en el comboBox "tiempo".
        gchar* unidades = gtk_combo_box_get_active_text(tiempo);
        //calcula el valor de "dormir" a partir del valor presente en
        //el comboBox "tiempo" y en el gtk_spin_button "cuantos".
        if (strcmp(unidades,"segundos")==0){
                        dormir = numero;
                }
                else if (strcmp(unidades,"minutos")==0){
                                dormir = numero;
                                dormir = dormir * 60;
                        }

        iniciarMedidas (fd_destino,fichero);
}
```

```
//método que llama inicialmente a "iniciaARFF", depués a
//"contadorAnadido" por cada elemento almacenado en "datos", y después
//a "acabaARFF". Posteriormente, y mientras la variable "mide" indique
//que hay que medir, llama a "cuenta" y a "sleep" durante el intervalo
//indicado por "dormir".
void iniciarMedidas (int fd_destino,gchar* fichero){
	iniciaARFF (fd_destino,fichero);
	int i = 0;
	int j = 0;
	char* contador;
	//copia de seguridad de "datos", que se modifica con "strtok".
	i=0;
	while (datos2[i]!='\0'){
		datos3[i]=datos2[i];
		i++;
	}
	//llamada a "contadorAnadido" por cada elemento en "datos"
	contador=strtok(datos2,"|");
	while(contador != NULL){
		contadorAnadido(fd_destino,contador);
		contador = strtok(NULL, "|");
	}
	acabaARFF(fd_destino);
	i=0;
	while (datos3[i]!='\0'){
		datos2[i]=datos3[i];
		i++;
	}
	while(mide==1){
		cuenta(fd_destino);
		sleep(dormir);
	}
}


//método que, para cada elemento presente en "datos", lo compara con los
//nombres de los datos de rendimiento almacenados en "almacenDatos".
//si coincide con alguno de ellos, llama a la escritora correspondiente
//al tipo de dicho dato. y si no coincide con ningún nombre de dato
//presente en "almacenDatos", llama a "escribeTodo", que escribirá
//en el archivo de destino el contenido completo del archivo de /proc.
void introduceDatos(int fd_destino){
	int fd_origen;
	int i;
	int j;
	int h = 0;
	char* contador;
	char* contadores[BUFSIZ];
	char buffer[BUFSIZ];
```

```
	//copia de seguridad de "datos", que se modifica con "strtok".
	i=0;
	while (datos2[i]!='\0'){
		datos3[i]=datos2[i];
		i++;
	}
	contador=strtok(datos2,"|");
	while(contador != NULL){
		contadores[h]=contador;h++;
		contador = strtok(NULL,"|");
	}
	int o=0;
	//para cada dato de rendimiento o archivo presente en "datos"
	while(contadores[o]!='\0'){
		int k = 0;
	      //para cada elemento de "almacenDatos"...
		while (k < numDatos){
		  //si el elemento de "datos" coincide con el
		  //nombre de algún dato de rendimiento que haya
		  //en "almacenDatos"...
			if (strcmp(getNombre(almacenDatos[k]),contadores[o])==0){
			 //se llama a la escritora
			 //correspondiente en función del tipo
			 //del dato de rendimiento.
				switch(getTipo(almacenDatos[k])){
				case 1:

	escribe1(getRuta(almacenDatos[k]),fd_destino,getLinea(almacenDatos[k]),getIn
icio(almacenDatos[k]),getFin(almacenDatos[k]));
				break;
				case 2:

	escribe2(getRuta(almacenDatos[k]),fd_destino,getLinea(almacenDatos[k]),getSe
parador(almacenDatos[k]),getCual(almacenDatos[k]));
				break;
				}
				break;
			}
			else{k++;}
			}
		  //si el elemento de "datos" no coincide con el
		  //nombre de ningún dato de rendimiento de
		  //"almacenDatos", se llama a "escribeTodo".
			if(k==numDatos){
				escribeTodo(contadores[o],fd_destino);
			}
			o++;
			}
	write(fd_destino,"\n",1);
	j=0;
```

```
	while (datos3[j]!='\0'){
		datos2[j]=datos3[j];
		j++;
	}
}


//método que escribe el contenido completo del archivo "contador" en
//el fichero de destino identificado por "fd_destino", siempre
//respetando el formato de archivos ARFF.
void escribeTodo(char* contador,int fd_destino){
	int fd_origen;
	char buffer[BUFSIZ];
	//abre el fichero de /proc del que se toman las medidas.
	fd_origen = open(contador, O_RDONLY);
	int i=0;
	while(i<BUFSIZ){
		buffer[i]='|';
		i++;
	}
	//lee el fichero de /proc a medir
	read(fd_origen,buffer,sizeof(buffer));
	int j=0;
	while(buffer[j]!='|'){
		j++;
	}
	write(fd_destino,", ",2);
	//escribe las medidas en el fichero de destino.
	write(fd_destino,buffer,j);
}


//método que escribe los caracteres presentes entre "inicio" y "fin" de
//la línea "linea" del archivo "contador", en el fichero de destino
//"fd_destino", siempre respetando el formato de archivos ARFF.
//es la escritora correspondiente al tipo de dato de rendimiento 1.
void escribe1(char* contador,int fd_destino,int linea,int inicio,int fin){
	int fd_origenDos;
	char bufferDos[BUFSIZ];
	//abre el fichero de /proc en el que se encuentra el dato.
	fd_origenDos = open(contador, O_RDONLY);
	int i=0;
	int k=0;
	int n=1;
	int p=0;
	char dataDos[BUFSIZ];
	char basura[BUFSIZ];
	while(i<BUFSIZ){bufferDos[i]='|';i=i+1;}
	read(fd_origenDos,bufferDos,sizeof(bufferDos));
	//línea del archivo en el que se encuentra exactamente el dato.
```

```
	char unaLineaDos[BUFSIZ];
	int j = 0;
	int m = 0;
	while(j < linea){
		while(bufferDos[k]!='\n'){
			unaLineaDos[m]=bufferDos[k];
			k++;
			m++;
		}
		j++;
		k++;
		m=0;
	}
	int l=0;
	while(l<inicio){
		basura[p]=unaLineaDos[l];
		p++;
		l++;
	}
	int r=0;
	//guardamos los caracteres entre "inicio" y "fin" de la linea.
	while(l<=fin){
		dataDos[r]=unaLineaDos[l];
		r++;l++;
	}
	r--;
	close(fd_origenDos);
	write(fd_destino,", ",2);
	//se escriben los datos en el fichero destino de las medidas.
	write(fd_destino,dataDos,r);
	write(fd_destino,"\n",1);

}


//método que escribe el token número "cual", en base al separador
//"separador", de la línea "linea" del archivo "contador", en el
//fichero de destino "fd_destino", en base al formato de archivos ARFF.
//es la escritora correspondiente al tipo de dato de rendimiento 2.
void escribe2(char* contador,int fd_destino,int linea,char* separador,int cual){
	int fd_origen;
	char buffer[BUFSIZ];
	//abre el fichero de /proc en el que se encuentra el dato.
	fd_origen = open(contador, O_RDONLY);
	int i=0;
	int k=0;
	int l=0;
	int n=1;
	char* t;
	char data[BUFSIZ];
```

```
        while(i<BUFSIZ){
                buffer[i]='|';
                i++;
        }
        read(fd_origen,buffer,sizeof(buffer));
        //línea del archivo en el que se encuentra exactamente el dato.
        char unaLinea[BUFSIZ];
        char unaLinea2[BUFSIZ];
        int j = 0;
        int m = 0;
        while(j < linea){
                while(buffer[k]!='\n'){
                        unaLinea[m]=buffer[k];
                        k++;
                        m++;
                }
                j++;
                k++;
                m=0;
        }
        while (unaLinea[l]!='\0'){
                unaLinea2[l]=unaLinea[l];
                l++;
        }
        //guardamos el token número "cual" en base al separador
        //"separador" de la línea del archivo anteriormente calculada.
        t=strtok(unaLinea2,separador);
        while (n < cual){
                t=strtok(NULL,separador);
                n++;
        }
        close(fd_origen);
        write(fd_destino,", ",2);
        //escribimos las medidas en el fichero de destino.
        write(fd_destino,t,strlen(t));
        write(fd_destino,"\n",1);

}


//método al que se llama cada vez que se quiere realizar una toma de
//datos (desde el método "iniciarMedidas"). invoca a los métodos
//"ponHora" e "introduceDatos".
void cuenta (int fd_destino)
{
        ponHora (fd_destino);
        introduceDatos (fd_destino);
}
```

```
//método que escribe la hora actual en el archivo de destino
//"fd_destino", en base al formato ARFF. usa el método de C "time".
void ponHora (int fd_destino)
{
	//toma la fecha y hora actuales con "time".
	time_t tAct = time(NULL);
	char* tiempoo;
	//convierte la hora tomada anteriormente al formato estándar.
	tiempoo = asctime(localtime(&tAct));
	//escribe fecha y hora en el fichero destino de las medidas.
	write(fd_destino,tiempoo,24);
	write(fd_destino,"\n",1);
	write(fd_destino," ",1);
}


//método que escribe los primeros caracteres de un archivo en formato
//ARFF. escribirá la ruta del propio fichero de destino en el que
//se está escribiendo.
void iniciaARFF (int fd_destino,gchar* fichero)
{
	int i = 0;
	while (fichero[i]!='\0'){
		i++;
	}
	//escribe la cabecera de un archivo con formato ARFF.
	write(fd_destino, "@RELATION ", 10);
	write(fd_destino, fichero, i);
	write(fd_destino, "\n", 1);
	write(fd_destino, "\n", 1);
	write(fd_destino, "@ATTRIBUTE time DATE", 20);
	write(fd_destino, "\n", 1);
}


//método que escribe los últimos caracteres de la cabecera de un
//archivo en formato ARFF.
void acabaARFF (int fd_destino)
{
	//acaba de escribir la cabecera de un archivo con formato ARFF.
	write(fd_destino, "\n", 1);
	write(fd_destino, "@DATA", 5);
	write(fd_destino, "\n", 1);
	write(fd_destino, "\n", 1);
}
```

```
//método que escribe en formato ARFF, en el fichero de destino
//"fd_destino", un contador de los que se va a tomar medidas.
//se le llamará desde "iniciarMedidas" para cada contador almacenado.
void contadorAnadido (int fd_destino,char contador[])
{
        int i = 0;
        while (contador[i]!='\0'){
                i++;
        }
        //escribe el nombre de cada elemento del que se tomarán medidas.
        write(fd_destino,"@ATTRIBUTE ",11);
        write(fd_destino,contador,i);
        write(fd_destino," NUMERIC",8);
        write(fd_destino, "\n", 1);
}


//oyente de la interfaz gráfica para el botón "parar", que detiene
//la toma de medidas. pone la variable "mide" de nuevo a cero, cierra
//el fichero destino de las medidas, y sale de la interfaz gráfica.
void
on_parar_clicked                (GtkButton      *button,
                                gpointer        user_data)
{
        int a;
        //cierra el fichero destino de las medidas.
        a = close(fd_destino);
        //detiene la toma de medidas.
        mide = 0;
        //sale de la interfaz gráfica.
        gtk_main_quit();
}


//oyente de la interfaz gráfica para el botón "inicio". llama al
//método "inicializar", que rellena "almacenDatos" con todos los
//datos de rendimiento estudiados hasta el momento. rellena el comboBox
//con todos los archivos de /proc presentes en las entradas previamente
//indicadas (llamando a "recorrer_arbol"), y con todos los datos de
//rendimiento que haya en "almacenDatos". también rellena el
//comoboBox de las unidades de tiempo.
void
on_inicio_clicked               (GtkButton      *button,
                                gpointer        user_data)
{
        //llama al método "inicializar", que rellena "almacenDatos" con
        //todos los datos de rendimiento estudiados hasta el momento.
        inicializar();
        //puntero al comboBox "comboProc" de la interfaz gráfica.
```

```
	GtkWidget * cp = lookup_widget(GTK_WIDGET(button), "comboProc");
	//llama al método "recorrer_arbol", que rellena el comboBox con todos
	//los archivos presentes en las entradas del directorio /proc que se
	//hayan indicado.
	recorrer_arbol("/proc",cp,0);
	//añade al comboBox todos los datos de rendimiento estudiados hasta el
	//moomento, y que están presentes en "almacenDatos".
	int k = 0;
		while (k < numDatos){
			gtk_combo_box_append_text (cp,getNombre(almacenDatos[k]));
			k++;
		}
	//puntero al comboBox "tiempo" de la interfaz gráfica.
	GtkWidget * tiempo = lookup_widget(GTK_WIDGET(button), "tiempo");
	//rellena el comboBox "tiempo" con las unidades segundos y minutos.
	gtk_combo_box_append_text (tiempo, "segundos");
	gtk_combo_box_append_text (tiempo, "minutos");
}


//oyente de la interfaz gráfica para el botón "guardar", que guarda, en
//un archivo previamente indicado por el usuario a través de la
//interfaz gráfica, la lista con todos los datos de rendimiento y
//archivos que hay que medir, y que están almacenados en "datos".
void
on_guardar_clicked		(GtkButton	*button,
				gpointer	user_data)
{
	int i;
	int fd_donde;
	char* contador;
	//puntero al gtk_text_entry de la interaz gráfica.
	GtkWidget * donde = lookup_widget(GTK_WIDGET(button),
"cargarGuardar");
	//texto presente en el gtk_text_entry "cargarGuardar".
	gchar* fichero = gtk_entry_get_text(donde);
	//abre el fichero indicado por el texto anterior.
	fd_donde = open(fichero, O_WRONLY|O_TRUNC|O_CREAT, 0666);
	//copia de seguridad para "datos", que se modificará con "strtok".
	i=0;
	while (datos2[i]!='\0'){
		datos3[i]=datos2[i];
		i++;
	}
	//guarda en el fichero indicado anteriormente todos los datos de rendimiento
	//y archivos que estuvieran almacenados en "datos" para medirlos.
	contador=strtok(datos2,"|");
	while(contador != NULL){
		int i = 0;
		while (contador[i]!='\0'){
```

```
                        i++;
                }
                write(fd_donde,contador,i);
                write(fd_donde,"|",1);
                contador = strtok(NULL, "|");
        }
        i=0;
        while (datos3[i]!='\0'){
                datos2[i]=datos3[i];
                i++;
        }
}


//oyente de la interfaz gráfica para el botón "cargar", que carga en
//"datos" y en el campo de texto de la interfaz gráfica habilitado
//para ello, la lista de todos los datos de rendimiento y archivos
//a medir que está almacenada en el archivo pasado previamente
//por el usuario a través de la interfaz gráfica.
void
on_cargar_clicked                   (GtkButton       *button,
                                    gpointer         user_data)
{
        int fd_donde;
        char* contador;
        char* buffer[BUFSIZ];
        //puntero al gtk_text_entry de la interaz gráfica.
        GtkWidget * que = lookup_widget(GTK_WIDGET(button), "datos");
        //puntero al gtk_text_entry de la interaz gráfica.
        GtkWidget * donde = lookup_widget(GTK_WIDGET(button),
"cargarGuardar");
        //texto presente en el gtk_text_entry "cargarGuardar".
        gchar* fichero = gtk_entry_get_text(donde);
        //abre el fichero indicado por el texto anterior.
        fd_donde = open(fichero, O_RDONLY);
        //lee los datos de rendimiento y archivos presentes en dicho fichero
        read(fd_donde,buffer,sizeof(buffer));
        //y los va metiendo en la variable "datos" y en el gtk_text_entry "datos".
        contador=strtok(buffer,"|");
        while(contador != NULL){
                strcat(datos,contador);
                strcat(datos,"|");
                strcat(datos2,contador);
                strcat(datos2,"|");
                strcat(datos3,contador);
                strcat(datos3,"|");
                gtk_entry_append_text(que,contador);
                contador = strtok(NULL, "|");
        }
}
```

```
//oyente de la interfaz gráfica para el botón "anadirEntrada",
//que añade una nueva entrada a "contadores", que almacena el nombre
//de todas las entradas de /proc cuyos arhivos habrá de listar el
//método "recorrer_arbol".
void
on_anadirEntrada_clicked               (GtkButton       *button,
                                        gpointer         user_data)
{
        //puntero al gtk_text_entry "posibleCont" de la interfaz gráfica.
        GtkWidget * pos = lookup_widget(GTK_WIDGET(button), "posibleCont");
        //texto presente en dicha entrada de texto.
        gchar* posible = gtk_entry_get_text(pos);
        //añade a la lista de entradas a listar por "recorrer_arbol" (contadores)
        //un nuevo elemento.
        strcat(contadores,posible);
        strcat(contadores,"|");
        gtk_entry_set_text(pos,"");
}


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///////INTERFACE.C

/*
 * DO NOT EDIT THIS FILE - it is generated by Glade.
 */
#ifdef HAVE_CONFIG_H
#  include <config.h>
#endif
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>

#include <gdk/gdkkeysyms.h>
#include <gtk/gtk.h>

#include "callbacks.h"
#include "interface.h"
#include "support.h"

#define GLADE_HOOKUP_OBJECT(component,widget,name) \
  g_object_set_data_full (G_OBJECT (component), name, \
    gtk_widget_ref (widget), (GDestroyNotify) gtk_widget_unref)

#define GLADE_HOOKUP_OBJECT_NO_REF(component,widget,name) \
  g_object_set_data (G_OBJECT (component), name, widget)
```

```
GtkWidget*
create_datosLinux (void)
{
  GtkWidget *datosLinux;
  GtkWidget *fixed1;
  GtkObject *cuantos_adj;
  GtkWidget *cuantos;
  GtkWidget *destino;
  GtkWidget *iniciar;
  GtkWidget *parar;
  GtkWidget *datos;
  GtkWidget *cargarGuardar;
  GtkWidget *posibleCont;
  GtkWidget *guardar;
  GtkWidget *cargar;
  GtkWidget *eliminar;
  GtkWidget *inicio;
  GtkWidget *anadir;
  GtkWidget *anadirEntrada;
  GtkWidget *comboProc;
  GtkWidget *tiempo;

  datosLinux = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title (GTK_WINDOW (datosLinux), _("window1"));

  fixed1 = gtk_fixed_new ();
  gtk_widget_show (fixed1);
  gtk_container_add (GTK_CONTAINER (datosLinux), fixed1);

  cuantos_adj = gtk_adjustment_new (1, 0, 100, 1, 10, 10);
  cuantos = gtk_spin_button_new (GTK_ADJUSTMENT (cuantos_adj), 1, 0);
  gtk_widget_show (cuantos);
  gtk_fixed_put (GTK_FIXED (fixed1), cuantos, 24, 464);
  gtk_widget_set_size_request (cuantos, 64, 32);

  destino = gtk_entry_new ();
  gtk_widget_show (destino);
  gtk_fixed_put (GTK_FIXED (fixed1), destino, 24, 504);
  gtk_widget_set_size_request (destino, 720, 40);

  iniciar = gtk_button_new_with_mnemonic (_("iniciar medicion"));
  gtk_widget_show (iniciar);
  gtk_fixed_put (GTK_FIXED (fixed1), iniciar, 200, 560);
  gtk_widget_set_size_request (iniciar, 144, 40);

  parar = gtk_button_new_with_mnemonic (_("parar medicion"));
  gtk_widget_show (parar);
  gtk_fixed_put (GTK_FIXED (fixed1), parar, 376, 560);
  gtk_widget_set_size_request (parar, 144, 40);
```

```
datos = gtk_entry_new ();
gtk_widget_show (datos);
gtk_fixed_put (GTK_FIXED (fixed1), datos, 32, 160);
gtk_widget_set_size_request (datos, 712, 56);

cargarGuardar = gtk_entry_new ();
gtk_widget_show (cargarGuardar);
gtk_fixed_put (GTK_FIXED (fixed1), cargarGuardar, 32, 296);
gtk_widget_set_size_request (cargarGuardar, 712, 56);

posibleCont = gtk_entry_new ();
gtk_widget_show (posibleCont);
gtk_fixed_put (GTK_FIXED (fixed1), posibleCont, 24, 16);
gtk_widget_set_size_request (posibleCont, 704, 40);

guardar = gtk_button_new_with_mnemonic (_("guardar"));
gtk_widget_show (guardar);
gtk_fixed_put (GTK_FIXED (fixed1), guardar, 200, 360);
gtk_widget_set_size_request (guardar, 88, 40);

cargar = gtk_button_new_with_mnemonic (_("cargar"));
gtk_widget_show (cargar);
gtk_fixed_put (GTK_FIXED (fixed1), cargar, 416, 360);
gtk_widget_set_size_request (cargar, 96, 40);

eliminar = gtk_button_new_with_mnemonic (_("eliminar"));
gtk_widget_show (eliminar);
gtk_fixed_put (GTK_FIXED (fixed1), eliminar, 416, 224);
gtk_widget_set_size_request (eliminar, 88, 40);

inicio = gtk_button_new_with_mnemonic (_("inicio"));
gtk_widget_show (inicio);
gtk_fixed_put (GTK_FIXED (fixed1), inicio, 32, 104);
gtk_widget_set_size_request (inicio, 96, 32);

anadir = gtk_button_new_with_mnemonic (_("aÃ±adir"));
gtk_widget_show (anadir);
gtk_fixed_put (GTK_FIXED (fixed1), anadir, 200, 224);
gtk_widget_set_size_request (anadir, 88, 40);

anadirEntrada = gtk_button_new_with_mnemonic (_("aÃ±andir entrada"));
gtk_widget_show (anadirEntrada);
gtk_fixed_put (GTK_FIXED (fixed1), anadirEntrada, 272, 64);
gtk_widget_set_size_request (anadirEntrada, 168, 32);

comboProc = gtk_combo_box_new_text ();
gtk_widget_show (comboProc);
gtk_fixed_put (GTK_FIXED (fixed1), comboProc, 136, 104);
gtk_widget_set_size_request (comboProc, 608, 32);
```

```
tiempo = gtk_combo_box_new_text ();
gtk_widget_show (tiempo);
gtk_fixed_put (GTK_FIXED (fixed1), tiempo, 104, 464);
gtk_widget_set_size_request (tiempo, 176, 32);

g_signal_connect ((gpointer) iniciar, "clicked",
                  G_CALLBACK (on_iniciar_clicked),
                  NULL);
g_signal_connect ((gpointer) parar, "clicked",
                  G_CALLBACK (on_parar_clicked),
                  NULL);
g_signal_connect ((gpointer) guardar, "clicked",
                  G_CALLBACK (on_guardar_clicked),
                  NULL);
g_signal_connect ((gpointer) cargar, "clicked",
                  G_CALLBACK (on_cargar_clicked),
                  NULL);
g_signal_connect ((gpointer) eliminar, "clicked",
                  G_CALLBACK (on_eliminar_clicked),
                  NULL);
g_signal_connect ((gpointer) inicio, "clicked",
                  G_CALLBACK (on_inicio_clicked),
                  NULL);
g_signal_connect ((gpointer) anadir, "clicked",
                  G_CALLBACK (on_anadir_clicked),
                  NULL);
g_signal_connect ((gpointer) anadirEntrada, "clicked",
                  G_CALLBACK (on_anadirEntrada_clicked),
                  NULL);

/* Store pointers to all widgets, for use by lookup_widget(). */
GLADE_HOOKUP_OBJECT_NO_REF (datosLinux, datosLinux, "datosLinux");
GLADE_HOOKUP_OBJECT (datosLinux, fixed1, "fixed1");
GLADE_HOOKUP_OBJECT (datosLinux, cuantos, "cuantos");
GLADE_HOOKUP_OBJECT (datosLinux, destino, "destino");
GLADE_HOOKUP_OBJECT (datosLinux, iniciar, "iniciar");
GLADE_HOOKUP_OBJECT (datosLinux, parar, "parar");
GLADE_HOOKUP_OBJECT (datosLinux, datos, "datos");
GLADE_HOOKUP_OBJECT (datosLinux, cargarGuardar, "cargarGuardar");
GLADE_HOOKUP_OBJECT (datosLinux, posibleCont, "posibleCont");
GLADE_HOOKUP_OBJECT (datosLinux, guardar, "guardar");
GLADE_HOOKUP_OBJECT (datosLinux, cargar, "cargar");
GLADE_HOOKUP_OBJECT (datosLinux, eliminar, "eliminar");
GLADE_HOOKUP_OBJECT (datosLinux, inicio, "inicio");
GLADE_HOOKUP_OBJECT (datosLinux, anadir, "anadir");
GLADE_HOOKUP_OBJECT (datosLinux, anadirEntrada, "anadirEntrada");
GLADE_HOOKUP_OBJECT (datosLinux, comboProc, "comboProc");
GLADE_HOOKUP_OBJECT (datosLinux, tiempo, "tiempo");

return datosLinux;}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////SUPPORT.C

/*
 * DO NOT EDIT THIS FILE - it is generated by Glade.
 */

#ifdef HAVE_CONFIG_H
#  include <config.h>
#endif

#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>

#include <gtk/gtk.h>

#include "support.h"

GtkWidget*
lookup_widget                   (GtkWidget       *widget,
                                 const gchar     *widget_name)
{
  GtkWidget *parent, *found_widget;

  for (;;)
    {
      if (GTK_IS_MENU (widget))
        parent = gtk_menu_get_attach_widget (GTK_MENU (widget));
      else
        parent = widget->parent;
      if (!parent)
        parent = (GtkWidget*) g_object_get_data (G_OBJECT (widget),
"GladeParentKey");
      if (parent == NULL)
        break;
      widget = parent;
    }

  found_widget = (GtkWidget*) g_object_get_data (G_OBJECT (widget),
                                                 widget_name);
  if (!found_widget)
    g_warning ("Widget not found: %s", widget_name);
  return found_widget;
}

static GList *pixmaps_directories = NULL;
```

```
/* Use this function to set the directory containing installed pixmaps. */
void
add_pixmap_directory                    (const gchar     *directory)
{
  pixmaps_directories = g_list_prepend (pixmaps_directories,
                                        g_strdup (directory));
}

/* This is an internally used function to find pixmap files. */
static gchar*
find_pixmap_file                        (const gchar     *filename)
{
  GList *elem;

  /* We step through each of the pixmaps directory to find it. */
  elem = pixmaps_directories;
  while (elem)
    {
      gchar *pathname = g_strdup_printf ("%s%s%s", (gchar*)elem->data,
                                         G_DIR_SEPARATOR_S, filename);
      if (g_file_test (pathname, G_FILE_TEST_EXISTS))
        return pathname;
      g_free (pathname);
      elem = elem->next;
    }
  return NULL;
}

/* This is an internally used function to create pixmaps. */
GtkWidget*
create_pixmap                           (GtkWidget       *widget,
                                         const gchar     *filename)
{
  gchar *pathname = NULL;
  GtkWidget *pixmap;

  if (!filename || !filename[0])
      return gtk_image_new ();

  pathname = find_pixmap_file (filename);

  if (!pathname)
    {
      g_warning (_("Couldn't find pixmap file: %s"), filename);
      return gtk_image_new ();
    }

  pixmap = gtk_image_new_from_file (pathname);
  g_free (pathname);
  return pixmap;
```

```
}

/* This is an internally used function to create pixmaps. */
GdkPixbuf*
create_pixbuf                   (const gchar     *filename)
{
  gchar *pathname = NULL;
  GdkPixbuf *pixbuf;
  GError *error = NULL;

  if (!filename || !filename[0])
      return NULL;

  pathname = find_pixmap_file (filename);

  if (!pathname)
    {
      g_warning (_("Couldn't find pixmap file: %s"), filename);
      return NULL;
    }

  pixbuf = gdk_pixbuf_new_from_file (pathname, &error);
  if (!pixbuf)
    {
      fprintf (stderr, "Failed to load pixbuf file: %s: %s\n",
               pathname, error->message);
      g_error_free (error);
    }
  g_free (pathname);
  return pixbuf;
}

/* This is used to set ATK action descriptions. */
void
glade_set_atk_action_description       (AtkAction       *action,
                                        const gchar     *action_name,
                                        const gchar     *description)
{
  gint n_actions, i;

  n_actions = atk_action_get_n_actions (action);
  for (i = 0; i < n_actions; i++)
    {
      if (!strcmp (atk_action_get_name (action, i), action_name))
        atk_action_set_description (action, i, description);
    }
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////MAIN.C


/*
 * Initial main.c file generated by Glade. Edit as required.
 * Glade will not overwrite this file.
 */

#ifdef HAVE_CONFIG_H
#  include <config.h>
#endif

#include <gtk/gtk.h>

#include "interface.h"
#include "support.h"

int
main (int argc, char *argv[])
{
  GtkWidget *datosLinux;

#ifdef ENABLE_NLS
  bindtextdomain (GETTEXT_PACKAGE, PACKAGE_LOCALE_DIR);
  bind_textdomain_codeset (GETTEXT_PACKAGE, "UTF-8");
  textdomain (GETTEXT_PACKAGE);
#endif

  gtk_set_locale ();
  gtk_init (&argc, &argv);

  add_pixmap_directory (PACKAGE_DATA_DIR "/" PACKAGE "/pixmaps");

  /*
   * The following code was added by Glade to create one of each component
   * (except popup menus), just so that you see something after building
   * the project. Delete any components that you don't want shown initially.
   */
  datosLinux = create_datosLinux ();
  gtk_widget_show (datosLinux);

  gtk_main ();
  return 0;
}
```

Anexo 3.1:
Implementación de la propuesta de organización, versión 1

Se muestra sólo la especificación de las clases.

El código está disponible en el CD que acompaña a la memoria.

Para favorecer la legibilidad no se incluyen los constructores y destructores ni los métodos accesores /mutadores.

CDatoRendimiento
char* nombre
int compara (CDatoRendimiento*) static CDatoRendimiento** Intersección (CDatoRendimiento**, CDatoRendimiento**, unsigned int, unsigned int, unsigned int&)

CMultiOrg
COrgIndividual** bosque unsigned int capacidad unsigned int nArboles CControlMonitor* monitor
void AnadeDatoRendimiento (CDatoRendimiento*) void EliminaDatoRendimiento (CDatoRendimiento*) void EliminaDatoRendimiento (unsigned int) void EliminaDatoRendimiento (char*) void AnadeOrganizacion (COrgIndividual*) CDatoRendimiento** Busca (char**, unsigned int&) void CargaDatosRendimiento (char*) void GuardaDatosRendimiento (char*) int Consulta (char*, unsigned int&) unsigned int DameNumParametros () char** DameParametros (unsigned int&)

CMultiOrgWin: implementa CMultiOrg
void AbreArchivo (char*) void CierraArchivo () void consultaArchivo () void preparaArchivo ()

CControlMonitor
CDatoRendimiento** cjtoConsulta unsigned int nDatos unsigned int capacidadNDatos
void anade (CDatoRendimiento*) void elimina (CDatoRendimiento*) void elimina (unsigned int)

void elimina (char*) void CargaDatos (char*) void GuardaDatos (char*) int consulta (char*, unsigned int&)

ControlMonitorWin: implementa CControlMonitor
ofstream archivo HCOUNTER** hCounter HQUERY hQuery
void AbreArchivo (char*) void anade(CDatoRendimiento*) void CabeceraARFF () void CargaDatos (char*) void CierraArchivo () int consulta (char*, unsigned int&) void consultaARFF () void DameDatosConsultados (void*) void elimina (CDatoRendimiento*) void elimina (int) void elimina (char*) void GuardarDatos (char*)

COrgIndividual
char* nombre char** campos int nCampos
CDatoRendimiento** Busca (char**, unsigned int&)

COrgActWin: implementa COrgIndividual
CDatoRendimiento** Busca (char**, unsigned int&) void CalculaInstanciaContadorValido (AnsiString, TStringList*, TStringList*, char*, char*) void CalculaObjetosValidos (TStringList*, char*) void CompletaRutasValidas (TStringList*, TStringList*, char*, char*) void devuelveInstanciasContadores (AnsiString, TStringList*, TStringList*) void devuelveObjetos (TStringList*) void extiendeInstancias (AnsiString, TStringList*, AniString) AnsiString hazPathContador (AnsiString, AnsiString, AnsiString)

Anexo 3.2:
Implementación de la propuesta de organización, versión 2

Se muestra sólo la especificación de las clases.

El código está disponible en el CD que acompaña a la memoria.

Para favorecer la legibilidad no se incluyen los constructores y destructores ni los métodos accesores /mutadores.

CDatoRendimiento
char* nombre
int compara (CDatoRendimiento*) static CDatoRendimiento** Intersección (CDatoRendimiento**, CDatoRendimiento**, unsigned int, unsigned int, unsigned int&)

CMultiOrg
CSistCon* SisConsulta CSisOrg* SisOrganización
void AnadeDatoRendimiento (CDatoRendimiento*) void EliminaDatoRendimiento (CDatoRendimiento*) void EliminaDatoRendimiento (unsigned int) void EliminaDatoRendimiento (char*) void AnadeOrganizacion (COrgIndividual*) CDatoRendimiento** Busca (char**, unsigned int&) void CargaDatosRendimiento (char*) void GuardaDatosRendimiento (char*) int Consulta (char*, unsigned int&) unsigned int DameNumParametros () char** DameParametros (unsigned int&)

CMultiOrgWin: implementa CMultiOrg
(Sólo redefine los constructores)

CSistCon
CControlMonitor* monitor
void AnadeDatoRendimiento (CDatoRendimiento*) void EliminaDatoRendimiento (CDatoRendimiento*) void EliminaDatoRendimiento (unsigned int) void EliminaDatoRendimiento (char*) void AnadeOrganizacion (COrgIndividual*) CDatoRendimiento** Busca (char**, unsigned int&) void CargaDatosRendimiento (char*) void GuardaDatosRendimiento (char*) int Consulta (char*, unsigned int&)

CSistConWin: implementa CSistCon
(Sólo redefine los constructores)

CControlMonitor
CDatoRendimiento** cjtoConsulta unsigned int nDatos unsigned int capacidadNDatos
void anade (CDatoRendimiento*) void elimina (CDatoRendimiento*) void elimina (unsigned int) void elimina (char*) void CargaDatos (char*) void GuardaDatos (char*) int consulta (char*, unsigned int&)

CControlMonitorWin: implementa CControlMonitor
HCOUNTER** hCounter HQUERY hQuery
void anade (CDatoRendimiento*) void CargaDatos (char*) int consulta (void*&, unsigned int&) unsigned int DameDatosConsultados (void*) void elimina (CDatoRendimiento*) void elimina (int) void elimina (char*) void GuardaDatos (char*)

CSisOrg
COrgIndividual** bosque unsigned int capacidad unsigned int nArboles
void AnadeOrganizacion (COrgIndividual*) CDatoRendimiento** Busca (char**, unsigned int&)

COrgIndividual
char* nombre char** campos int nCampos
CDatoRendimiento** Busca (char**, unsigned int&)

COrgActWin: implementa COrgIndividual

CDatoRendimiento** Busca (char**, unsigned int&) void CalculaInstanciaContadorValido (AnsiString, TStringList*, TStringList*, char*, char*) void CalculaObjetosValidos (TStringList*, char*) void CompletaRutasValidas (TStringList*, TStringList*, char*, char*) void devuelveInstanciasContadores (AnsiString, TStringList*, TStringList*) void devuelveObjetos (TStringList*) void extiendeInstancias (AnsiString, TStringList*, AniString) AnsiString hazPathContador (AnsiString, AnsiString, AnsiString)

Anexo 4.1
Implementación desarrollada para evaluaciones y pruebas de rendimiento

0. Introducción

El código desarrollado para el recubrimiento de la organización actual de los contadores de rendimiento se ha extendido para permitir la monitorización y las pruebas de carga de elementos funcionales dentro de la misma aplicación.

El código de la estructura de recubrimiento del que se parte está en el anexo 3.2.

El código de esta implementación está en el CD que acompaña a la memoria, en este anexo sólo se describen las clases.

Para aumentar la legibilidad, no aparecen constructores, destructores ni los métodos accesores y mutadores de los atributos de las clases.

CDatoRendimiento
char* nombre
int compara (CDatoRendimiento*) static CDatoRendimiento** Intersección (CDatoRendimiento**, CDatoRendimiento**, unsigned int, unsigned int, unsigned int&)

CMultiOrg
CSistCon* SisConsulta CSisOrg* SisOrganización CSisCarga* SisCarga
int activa (int, char**, int) int activa (char*, char**, int) void anadeCarga (CUniCarga*) void AnadeDatoRendimiento (CDatoRendimiento*) void AsignaCargador (CSisCarga*) void EliminaDatoRendimiento (CDatoRendimiento*) void EliminaDatoRendimiento (unsigned int) void EliminaDatoRendimiento (char*) void AnadeOrganizacion (COrgIndividual*) CDatoRendimiento** Busca (char**, unsigned int&) int buscaCarga (char*) void CargaDatosRendimiento (char*) void GuardaDatosRendimiento (char*) int Consulta (void*, unsigned int&) unsigned int DameNumParametros () char** DameParametros (unsigned int&) unsigned int DameDatosConsultados (void*) int DameEstadoCargador (char**, short*, int) int DameEstadoCargador (char**, short*, char*) int desactiva (int) int desactiva (char*)

bool estaActivo (int) bool estaActivo (char*)

CMultiOrgWin: implementa CMultiOrg
(Sólo redefine los constructores)

CSistCon
CControlMonitor* monitor
void AnadeDatoRendimiento (CDatoRendimiento*) void EliminaDatoRendimiento (CDatoRendimiento*) void EliminaDatoRendimiento (unsigned int) void EliminaDatoRendimiento (char*) void AnadeOrganizacion (COrgIndividual*) CDatoRendimiento** Busca (char**, unsigned int&) void CargaDatosRendimiento (char*) void GuardaDatosRendimiento (char*) int Consulta (char*, unsigned int&)

CSistConWin: implementa CSistCon
(Sólo redefine los constructores)

CControlMonitor
CDatoRendimiento** cjtoConsulta unsigned int nDatos unsigned int capacidadNDatos
void anade (CDatoRendimiento*) void elimina (CDatoRendimiento*) void elimina (unsigned int) void elimina (char*) void CargaDatos (char*) void GuardaDatos (char*) int consulta (char*, unsigned int&)

CControlMonitorWin: implementa CControlMonitor
HCOUNTER** hCounter HQUERY hQuery
void anade (CDatoRendimiento*) void CargaDatos (char*) int consulta (void*&, unsigned int&) unsigned int DameDatosConsultados (void*) void elimina (CDatoRendimiento*) void elimina (int)

void elimina (char*) void GuardaDatos (char*)

CSisOrg
COrgIndividual** bosque unsigned int capacidad unsigned int nArboles
void AnadeOrganizacion (COrgIndividual*) CDatoRendimiento** Busca (char**, unsigned int&)

COrgIndividual
char* nombre char** campos int nCampos
CDatoRendimiento** Busca (char**, unsigned int&)

COrgActWin: implementa COrgIndividual
CDatoRendimiento** Busca (char**, unsigned int&) void CalculaInstanciaContadorValido (AnsiString, TStringList*, TStringList*, char*, char*) void CalculaObjetosValidos (TStringList*, char*) void CompletaRutasValidas (TStringList*, TStringList*, char*, char*) void devuelveInstanciasContadores (AnsiString, TStringList*, TStringList*) void devuelveObjetos (TStringList*) void extiendeInstancias (AnsiString, TStringList*, AniString) AnsiString hazPathContador (AnsiString, AnsiString, AnsiString)

CSisCarga
int capacidad int tamano CUniCarga** unidades
int activa (int, char**, int) int activa (char*, char**, int) int busca (char*) bool dameActivo (int) bool dameActivo (char*) short DameEstado (char**&, short*&, int) short DameEstado (char**&, short*&, char*) int desactiva (int) int desactiva (char*)

CUniCarga
bool activo

short elemEstado char** estado short* grado char* nombre
int Carga (char**, int) int Desactivar ()

CWhetCarg
Implementa un método de carga del procesador basado en el benchmar Whetstone.

CWhetCarg: implementa CUniCarga
WetThread* hilo
int Carga (char**, int) int Desactivar ()

WetThread: extiende TThread
bool* activo double E1[4] char** estado short* grado int I double M int J int K int L int N1 int N2 int N3 int N4 int N5 int N6 int N7 int N8 int N9 int N10 int N11 double T double T1 double T2 double X double X1 double X2 double X3 double X4 double Y double Z
void Execute () void Inicializa (bool*, char**, short*)

```
void MODULE1()
void MODULE2()
void MODULE4()
void MODULE6()
void MODULE7()
void MODULE8()
void MODULE10()
void MODULE11()
void P3 (double, double, double&)
void PA (double E[])
```

Bibliografía

[BCB1] FRANCISCO CHARTE Programación con C++ Builder 5 , Ediciones Anaya Multimedia, 2000

[CEC 1] En http://www.cecalc.ula.ve/documentacion/tutoriales/beowulf/node22.html, obtenido con Internet Explorer, en Diciembre de 2005.

[CLX1] En http://fferrer.dsic.upv.es/cursos/Linux/basico/ch08.html, obtenido con Internet Explorer, en Noviembre de 2005.

[DMTF1] En www.dmtf.org, obtenido con Internet Explorer, en Octubre de 2005.

[MMCH1] Documentación de Microsoft Management Console

[MSDN1] En http://msdn.microsoft.com/library/, obtenido con Internet Explorer, en Octubre de 2005.

[MSS1] En http://www.microsoft.com/spain/technet/fases/benchmarks/default.mspx, obtenido con Internet Explorer, en Octubre de 2005.

[PDHREF1] Microsoft Windows Performance Data Helper Reference

[RHM1] En http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/ref-guide/ch-proc.html, obtenido con Internet Explorer, en Octubre de 2005.

[RHM2] En http://www2.tiendalinux.com/docs/manuales/redhat/rhl-rg-es-7.1/s1-filesystem-proc.html, obtenido con Internet Explorer, en Octubre de 2005.

[RHM3] En http://www.centos.org/docs/2/rhl-rg-en-7.2/ch-proc.html, obtenido con Internet Explorer, en Noviembre de 2005.

[RHM4] Red Hat Linux 8 Reference Guide

[SWT1] En http://www.serverwatch.com/tutorials/article.php/2177171, obtenido con Internet Explorer, en Marzo de 2006.

[SWT2] En http://www.serverwatch.com/tutorials/article.php/1476631, obtenido con Internet Explorer, en Marzo de 2005.

[WIK 1] En http://es.wikipedia.org/wiki/Benchmark, obtenido con Internet Explorer, en Noviembre de 2005.

[WIN32-1] Win32 Programmer's Reference