Sistemas Operativos

Trabajo Práctico Final

Noviembre 2004

Docente:

Roberto Bevilacqua <robevi(ARROBA)cnea.gov.ar>

Alumnos:

Alejandro Valdez <avaldez(ARROBA)dc.uba.ar>
Hernan Rancati <hrancati(ARROBA)softhome.net>

Homepage:

http://www.midnightsoret.com.ar/personales/alejandrovaldez/minix/

Sistema de cuota de disco

Introducción

El sistema de archivos de Minix provee un esquema de seguridad que permite la protección de los datos entre distintos usuarios, de esta manera el administrador del sistema puede asignar permisos de acceso a distinta información basándose en la identificación de usuario y sus grupos de pertenencia. Este esquema no impone un límite a la cantidad espacio ni a la cantidad de archivos que un usuario puede utilizar en el sistema. Esta carencia podría producir un uso no equitativo de los recursos del sistema entre los distintos usuarios, sin posibilidad de que el administrador pueda limitar o prevenir estas situaciones.

Un sistema de cuota de disco consiste en un mecanismo que permite al administrador del sistema indicar la cantidad de espacio máxima que un usuario puede utilizar. La presencia del sistema de cuota es transparente para los usuarios en tanto no excedan el uso que tienen permitido del espacio en disco, una vez excedido el límite el sistema impedirá el uso de mas espacio hasta que el usuario vuelva a tener un uso del disco inferior al máximo establecido para él.

Propuesta

Minix, al igual que otros sistemas operativos, carece de un sistema de cuota de disco. Se realizó una búsqueda vía Internet en la *web* y *newsgroups* tratando de encontrar algún trabajo en Minix relacionado con este tópico, pero solo se encontraron proyectos que no parecen haber sido terminados. No habiendo trabajos concluidos al respecto, nuestra propuesta es implementar un sistema de cuota para Minix.

Desarrollo

Funcionalidades del sistema de cuota

Nuestra primer tarea es identificar la funcionalidad que se espera del sistema de cuota, para luego poder analizar las distintas maneras de implementar una solución. Sin demasiado nivel de detalle un sistema de cuota debería:

1. Generar una base de datos inicial que refleje el uso del espacio en disco por parte de los usuarios del sistema.
2. Decidir ante una petición de escritura si es posible realizarla sin exceder la cuota asignada al usuario que intenta realizarla.
3. Mantener actualizada una base de datos conteniendo el uso de espacio en disco, de acuerdo a las peticiones de escritura y borrado de los usuarios.
4. Manejar la persistencia la base de datos para poder salvar el estado del sistema al reiniciar el sistema operativo.
5. Proveer herramientas para la administración del sistema de cuota.

Ubicación del sistema de cuota

El modelo de procesos de Minix esta compuesto por cuatro capas de procesos dependiendo de su función dentro del sistema:

1. Administración de procesos y mensajes (kernel)
2. Tareas de I/O (tarea de disco, tty, reloj del sistema, etc.)
3. Procesos servidores (FS y MM)
4. Procesos de usuario (init, ls, cat, etc.)

Considerando esta clasificación, una cuestión a analizar es cual es la ubicación apropiada del *quotad*[1] dentro de este esquema.

Es claro que las primeras dos capas están destinadas a funciones primitivas y sumamente básicas del sistema, por lo que no parecen niveles adecuados para implementar el mecanismo de cuota. La tercer capa esta mas relacionada con la abstracción de filesystem y el servidor de FS es una muestra de ello.

Una alternativa posible es agregar un nuevo servidor en la tercer capa destinado a realizar las tareas de control y mantenimiento de cuota correspondientes a las funciones 2, 3 y 4 de la sección anterior. Dado que los system calls relacionados con la asignación y liberación de recursos de disco están implementados en el servidor de FS, sería necesario implementar un mecanismo de comunicación entre el servidor de FS y el servidor de cuota. Esta comunicación agregaría un overhead no deseado a las operaciones de disco. Por otro lado existiría una discrepancia en el nivel de abstracción de las funcionalidades implementadas por los servidores, claramente un sistema de cuota

[1] Por *quota daemon*, de aquí en adelante se utilizará indistíntamente *quotad* como sinónimo de *sistema de cuota de disco*

es algo mucho mas específico que un servidor de sistema de disco o de manejo de memoria.

Un aspecto favorable de esta solución sería que la funcionalidad del sistema de cuota estaría concentrada en un solo módulo, esto disminuiría el acople con el resto del sistema operativo y facilitaría la distribución del sistema permitiendo tener el servidor de cuota tal vez en otra computadora.

Sin embargo, los motivos expuestos anteriormente nos resultan suficientes para no optar por esta solución.

Otra alternativa es implementar el quotad como un proceso de usuario. Esta solución comparte el problema del overhead en las operaciones de disco debido a la comunicación entre procesos. Además plantea dificultades de implementación ya que no logramos encontrar una manera elegante de lograr el envío de mensajes desde un proceso de nivel servidor hacia un proceso de nivel de usuario; por otra parte aparecen nuevos problemas a contemplar como la posible creación de mas de una instancia de quotad o la terminación de dicho proceso sin aviso al servidor de FS (sea por alguna anomalía o terminado por el superusuario) y varias cuestiones mas que hacen que no consideremos esta solución como adecuada.

La solución adoptada es un compromiso entre las ventajas y desventajas de las soluciones expuestas anteriormente. Para evitar el overhead del envío de mensajes se decidió incorporar la funcionalidad del quotad dentro del servidor de FS y para disminuir el acople del sistema de cuota con el servidor de FS se realizó un diseño del quotad de modo de presentar una interfaz de uso sumamente sencilla e independiente de la implementación original del servidor de FS.

Alcance de la cuota de disco

Para poder imponer el límite en el uso de espacio de disco es preciso considerar con que alcance se impondrá esta restricción.

Podría fijarse el límite a nivel sistema definiendo un máximo total de la cantidad de espacio que cada usuario puede utilizar, independientemente de la ubicación de los archivos. Minix permite ampliar el arbol de directorios agregando nuevos filesystems al sistema, estos últimos pueden estar almacenados en dispositivos removibles y nos plantea un problema de localidad de la cuota: ¿Si se remueve un filesystem, el usuario dejó de utilizar ese espacio en el sistema, o solamente ese espacio no esta accesible en ese momento?. Este tipo de preguntas nos hicieron pensar que tal vez sería mejor definir el límite del uso de disco a un nivel de granularidad mas bajo.

Consideramos imponer el límite a nivel de directorio, pero surge un problema de escalabilidad: El administrador del sistema tendría demasiado trabajo fijando las cuotas para cada directorio que resulte relevante. Esto no parece tener una solución simple y nos llevó a buscar otra solución.

Nos pareció adecuado imponer la limitación de espacio a nivel de filesystem. Está elección permite manejar el problema de remover un filesystem del sistema: dado que tanto el uso actual del filesystem como la limitación de utilización son locales a este, al remover un filesystem del sistema basta con remover del sistema de cuota las restricciones relacionadas a él.

Un aspecto interesante de tener la información relacionada al sistema de cuota incluida dentro del filesystem, es que en caso de agregar un filesystem con información de cuota en otra computadora que utilice el sistema de cuota, se puede continuar brindando el servicio de cuota sin requerir esfuerzo extra por parte del administrador. Otro aspecto positivo es la escalabilidad de la solución, en el sentido que no se espera contar con una gran cantidad de filesystems en un sistema.

Persistencia de datos

Ante la petición de escritura de un usuario, un sistema de cuota necesita conocer los siguientes datos para poder tomar un decisión entre permitir o negar el pedido:

1. Identificación del usuario al que contabilizarle el uso del disco.
2. Cantidad de espacio a asignar.
3. Uso actual del sistema de disco por parte del usuario.
4. Límite impuesto por el administrador del sistema para el uso del disco para el usuario del ítem 1.

Los items 1 y 2 forman parte del pedido de escritura a disco y no es necesario almacenarlos ni recuperarlos por medio de persistencia, pero los items 3 y 4 requieren cierto análisis.

Una manera de conocer el uso actual de espacio en disco por parte de un usuario, es modificar las estructuras de datos que forman el filesystem de Minix agregando campos con información sumarizada sobre el uso de cada usuario. Estos campos deberían ser actualizados desde dentro del servidor de FS y estarían extendiendo la funcionalidad que brinda el sistema de archivos. Esta solución si bien es factible nos pareció un tanto compleja de realizar, además implicaba apartarse del standard de Minix y nos desviaba de nuestro objetivo principal, es por eso que preferimos evaluar otras alternativas.

Otra forma de abordar el problema es incorporando al servidor de FS la funcionalidad de realizar un recorrido de todos los archivos contenidos en el disco, obteniendo la cantidad de espacio utilizada por cada usuario y manteniendo esa información en memoria para que sea accesible desde el sistema de cuota. Esta solución es muy poco eficiente pues deberia realizarse cada vez que se inicia el sistema operativo o se agrega un filesystem al sistema, el tiempo empleado en esa tarea nos resulta inaceptable y por eso descartamos esta solución.

Una mejora sobre esta alternativa es al mismo tiempo que se mantiene en memoria la información sobre la utilización del disco, almacenar esa información en algún archivo distinguido de donde pueda recuperarse estos datos posteriormente. De esta manera se soluciona el problema de tener que realizar un recorrido de los filesystems cada vez que se inicia el sistema operativo. Esta solución sería aceptable si no fuera por un problema de abstracción: ¿Es responsabilidad del kernel de un sistema operativo realizar un recorrido del sistema de archivos para generar información sumarizada de su uso? Un breve análisis mostrará que no es lugar apropiado para realizar este trabajo que podría ser hecho desde un programa de usuario.

De este modo llegamos a la solución implementada que consiste en tener un archivo distinguido (*quota.values*) que contiene la cantidad de espacio utilizado por cada

usuario del sistema, este archivo se genera por medio de un programa de usuario (/bin/*qsck*) que realiza el recorrido de un filesystem para generar esa información. Al momento de iniciar el sistema de cuota, se espera que el archivo quota.values este disponible para ser leido, incorporando su información dentro del sistema de cuota. Las acciónes de apertura, lectura y cerrado del archivo se realizan desde el sistema de cuota utilizando las funciones internas del servidor de FS sin que sea necesario la realización de ningún system call.

Observar que una vez que la información sobre el uso del disco es puesta a disposición del sistema de cuota, es responsabilidad de este mantener dichos datos actualizados en memoria. Del mismo modo es también su responsabilidad almacenar los datos actualizados antes de apagar el sistema operativo, de modo que estén disponibles cuando el sistema vuelva a iniciar.

Un análisis similar puede relizarse acerca de la manera de poner disponible al sistema de cuota la información sobre el límite establecido en el uso de disco para cada usuario. Analogamente al caso anterior utilizamos un archivo distinguido (*quota.conf*) que almacena la información mencionada en cada filesystem.

Robustez

Si se tiene en cuenta que el sistema de cuota mantiene datos en memoria que reflejan el uso del disco y que es parte de sus responsabilidades actualizar la base de datos (quota.values) donde se almacena esta información, surge inmediatamente la necesidad de analizar la robustez del sistema en casos de falla.

Nuestra intención era almacenar el archivo quota.values solo al momento de realizar el shutdown del sistema operativo, esto es suficiente en condiciones normales pero no alcanza para casos en los que no se haga un shutdown, como ser un corte de energía o apagado accidental de la computadora.

Era necesaria una política de actualización de los archivos que realizan la persistencia de la información manejada por el sistema de cuota. Surgieron diversas alternativas para abordar la cuestión.

El sistema de archivos de Minix utiliza un cache para almacenar bloques de disco que fueron accedidos recientemente, para mantener consistente esa cache con los datos grabados en disco existe un system call llamado *sync* que se encarga de grabar los datos en el disco. Con esto en mente podriamos utilizar la misma política que usa el sistema de archivos para mantener consistencia entre el cache y el disco, realizando escrituras al archivo quota.values al momento de realizar un sync del cache de disco.

Sin embargo, la alternativa que implementamos consiste en contar la cantidad de actualizaciones que se realizan a la información del sistema de cuota y al llegar a cierto número realizar la escritura a disco de esa información. Opcionalmente podría forzarse un llamado al system call sync para asegurar que los datos escritos quedaron grabados fisicamente en el disco.

Forzar un llamado al sync era en cierto sentido alterar las características de robustez del mismo sistema de archivos de Minix, y no nos parece adecuado. Así que preferimos limitar la estrategia de sincronización a escribir los archivos de persistencia y dejar que el sistema de archivos realice un sync cuando lo crea conveniente.

Dado que no se fuerza la sincronización del cache de disco, es posible que en una situación donde el equipo no haya realizado el proceso de shutdown, queden datos sin grabar en el archivo quota.values. Para manejar esa situación se utilizó la misma política que utiliza el sistema de archivos ante situaciones similares, y es realizar una verificación durante la fase de inicio del sistema para detectar si hubo un shutdown anómalo del sistema y en ese caso correr el proceso necesario para volver a generar el archivo quota.values, conteniendo el uso actual de los filesystems por parte de los usuarios del sistema.

Comandos de administración

Como se expuso anteriormente el nucleo del sistema de cuota esta incorporado al kernel como parte del servidor del sistema de archivos, mencionaremos ahora las funcionalidades básicas que deben estar a disposición del administrador del sistema para poder utilizar el sistema de cuota.

Una tarea importante es poder realizar un recorrido de los filesystems generando la información sumarizada del uso que hace cada usuario del cada filesystem. Esta funcionalidad esta implementada en el comando */bin/qsck*.

También es necesaria alguna manera de indicar al sistema cual es la cantidad de espacio máxima que un usuario puede utilizar en un filesystem. Esta funcionalidad esta implementada en el comando */bin/setquota*.

Poder averiguar el límite de cuota que tiene asignado un usuario es otra tarea fundamental para el administrador del sistema. Esta funcionalidad esta implementada en el comando */bin/getquota*.

Por último sería deseable que hubiera alguna manera de activar y desactivar el servicio de cuota para un filesystem dado, funcionalidad que esta implementada en el comando */bin/quota*.

Mas adelante daremos detalles sobre la implementación y uso de estos comandos.

Comunicación

Es necesario proveer algún mecanismo para que los comandos relacionados al sistema de cuota puedan comunicarse con la parte del sistema de cuota que esta a nivel del kernel.

Esta comunicación debería permitir que sea posible desde un programa de usuario:

- Incorporar al sistema de cuota los límites asignados a los usuarios para un filesystem dado.
- Incorporar al sistema de cuota el uso actual de espacio para un filesystem dado.
- Activar el sistema de cuota para un filesystem dado.
- Desactivar el sistema de quota para un filesystem dado.
- Consultar el estado de activación del sistema de cuota para un filesystem dado.
- Obtener la cantidad de espacio utilizado por un usuario en un filesystem dado.

Una primer solución consistió en ampliar el conjunto de system calls que maneja el servidor de FS, agregando un system call por cada una de las funcionalidades descriptas anteriormente.

Esta opción permitía solucionar la necesidad de comunicación, pero bastó con observar el propósito de los system calls implementados originalmente en el servidor de FS, para darse cuenta que había una diferencia de abstracción entre las funcionalidades brindadas por el conjunto original de system calls y el resultante de agregar los llamados necesarios para la comunicación con el sistema de cuota.

Esto nos llevó a reducir la cantidad de system calls a uno solo (*quotacmd*), que utiliza un parámetro incluido en el mensaje entregado al system call como selector de la operación a realizar.

Para facilitar el uso del nuevo system call se escribió un conjunto de funciones que posteriormente fueron agrupadas en una librería (*libquota.a*) que permite mediante el uso de funciones de alto nivel realizar las operaciones antes mecionadas desde un programa escrito en lenguaje C.

Sincronización

Como se explicó en la sección *Robustez* es responsabilidad del quotad realizar la actualización del archivo quota.values, que contiene el uso actual del disco por parte de los usuarios. Por otra parte, la generación inicial de ese archivo así como la regeneración en caso que haya alguna discrepancia entre el contenido de esa tabla y el uso real de espacio por parte de los usuarios, es responsabilidad del programa de usuario qsck.

Podría darse el caso que tanto el quotad como el programa qsck escribieran simultaneamente al archivo quota.values, produciendo un archivo corrupto (información inválida, o fuera de formato).

Los escenarios en los que es necesario correr el programa qsck se limitan a aquellos en los que los datos contenidos en el archivo quota.values están desactualizados o son inválidos. En ese caso se puede asumir que los datos con los que esta trabajando el quotad no están actualizados, con este detalle en cuenta se optó por desactivar el sistema de cuota sobre el filesystem donde se esta realizando la generación del archivo quota.values a fin de evitar una potencial escritura concurrente. Los servicios del quotad se activan inmediatamente después de que el programa qsck realiza la escritura del archivo quota.values. La comunicación entre el programa qsck y el quotad se realiza por medio del system call mencionado en la sección anterior.

La desactivación temporal del sistema de cuota, si bien se limita al filesystem donde se está realizando el qsck, puede producir que no se contabilicen algunas escrituras de los usuarios. Un análisis de esta situación se hará en la sección *Presición*.

Otra solución que consideramos consistía en agregar servicios en el system call que permite la comunicación con el quotad, para transferirle la información recientemente obtenida por el programa qsck. Una vez enviada esa información sería responsabilidad del quotad escribir el archivo quota.values cuando lo crea conveniente, de esta manera se evita la potencial escritura concurrente del archivo quota.values.

Esta solución era viable, pero presentaba el problema de continuar agregando funcionalidades al quotad. Una vez avanzado el desarrollo nuestra intención era disminuir esta funcionalidad y transferirla a programas de usuario dentro de lo posible.

Presición

En esta sección pretendemos presentar algunas dificultades al momento de mantener la presición en la contabilidad del uso del disco.

En la sección anterior se presentó una de estas dificultades, consistente en la necesidad de detener los servicios del quotad para un filesystem al que se le está regenerando el archivo quota.values. Es posible que durante la escritura del archivo quota.values el proceso qsck quede bloqueado por I/O y se asigne el CPU a algún otro proceso de usuario P. Si el proceso P escribiera al filesystem que tiene el sistema de quota desactivado, dichas escrituras no serían contabilizadas por el sistema de cuota.

Otro problema relacionado con la contabilidad del uso del espacio en disco surge al escribir el archivo quota.values. Durante el recorrido del filesystem que realiza el programa qsck para contabilizar el uso, se está contabilizando también el tamaño del archivo quota.values existente en el filesystem. Al escribir el nuevo archivo quota.values

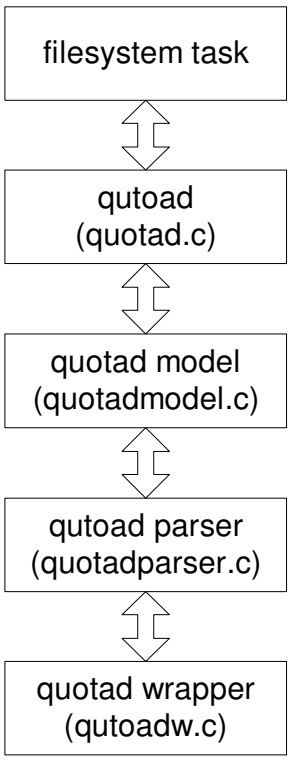
con la información actualizada, se está utilizando espacio que no fue contabilizado durante el recorrido del disco y que no será contabilizado por el quotad (recordar que es necesario desactivarlo durante la escritura del archivo quota.values).

Estas dos situaciones no están resueltas en la implementación que se presenta y quedan como trabajo futuro. De todos modos consideramos que tienen un impacto bajo sobre el funcionamiento global del sistema de cuota.

Implementación

Quotad

Es la porción del sistema de cuota que esta incorporada al servidor de FS dentro del kernel del sistema operativo. El programa fue realizado considerando las abstracciones necesarias de diseño para permitir la portabilidad del sistema de cuota a otros sistemas operativos. Se presenta una visión en bloques del sistema, las flechas indican la comunicación existente entre los distintos bloques:



A continuación se presenta un descripción de las funcionalidades de cada módulo y algunos comentarios al respecto.

Archivo /usr/src/fs/quotadw.c:

Contiene las operaciones de bajo nivel para soportar el sistema de cuota, como ser apertura y cierre de archivos en distintos modos de operación (lectura o escritura), así como la escritura y lectura de datos.

Para facilitar el desarrollo del sistema de cuota se realizaron dos implementaciones de estos servicios. Una implementación está destinada a ser utilizada desde dentro del kernel de Minix, en tanto que la otra está pensada para realizar pruebas utilizando el sistema de cuota como si fuera una aplicación de usuario y simulando los eventos de escritura y lectura con algún programa *stub* para realizar pruebas.

En el caso de ser compilado para ser utilizado desde espacio de usuario, las operaciones encapsulan llamados a las funciones standard de C para manejo de archivos (fopen, fclose, fread y fwrite). Mientras que si se compila para ser utilizado desde dentro de la tarea de FS, los llamados del módulo encapsulan llamados a funciones internas definidas dentro del servidor de FS para las operaciones básicas de manejo de archivos.

Esta doble implementación nos permitió realizar un proceso de debug mas agil y testear el sistema de cuotas fuera del espacio de kernel donde la detección de fallas es bastante mas compleja. Por otro lado nos permitió reutilizar gran parte de la funcionalidad del sistema de cuota al momento de escribir los programas de usuario que permiten la administración del sistema de cuota.

La selección del modo de uso de este módulo se realiza mediante la definición de la constante PLATFORM_MINIX, que indica que se pretende utilizar el módulo desde dentro del kernel de Minix. Por omición el módulo se compila para realizar sus funciones mediante las funciones standard de librería de C.

Archivo /usr/src/fs/quotadparser.c:

Contiene el parser que se utiliza para interpretar los archivos que dan persistencia a la información sobre uso de disco y límites de usuario.

El archivo que contiene los límites de usuarios (quota.conf) debe estar localizado en el directorio raiz de cada filesystem al que se quiera activar el servicio de cuota, se utilizó texto plano para representar los datos con el siguiente formato:

uid_1 <space> $spaceLimit_1$ <linefeed>
uid_2 <space> $spaceLimit_2$ <linefeed>
...
uid_n <space> $spaceLimit_n$ <linefeed>

Donde *uid* y *spaceLimit* representan números formados por caracteres ASCII, sin signos de puntuacion. El número representado por spaceLimit representa el límite máximo de uso de disco expresado un bytes. Se espera que no haya uid repetidos dentro del archivo. Los usuarios que no aparecen listados en este archivo se consideran sin límite de cuota.

El archivo que contiene el uso actual del espacio en disco para cada usuario (quota.values) esta localizado en el directorio raiz de cada filesystem al que se quiera activar el servicio de quota, se utilizó un formato similar al archivo quota.conf:

uid_1 <space> $spaceUsed_1$ <linefeed>
uid_2 <space> $spaceUsed_2$ <linefeed>
...
uid_n <space> $spaceUsed_n$ <linefeed>

Donde *uid* y *spaceUsed* representan números formados por caracteres ASCII, sin signos de puntuación. El número representado por spaceUsed representa el uso actual de disco expresado un bytes. Se espera que no haya uid repetidos dentro del archivo.

Archivo /usr/src/fs/quoadmodel.c:

Es el nucleo del sistema de cuota, maneja las abstracciones relacionadas con los filesystems activos en el sistema, lleva la contabilidad del uso de disco por parte de los usuarios y toma desiciones acerca de la aceptación o no de una petición de escritura por parte de un usuario.

En este módulo se reserva espacio en memoria para el manejo de las tablas del sistema de cuotas, la memoria está reservada por medio de arreglos estáticos dado que no es posible utilizar funciones de manejo de memoria dinámica desde dentro del kernel de Minix.

Dado que los arreglos tienen tamaño fijo existen algunas constantes que limitan la cantidad de filesystems y usuarios con los que el sistema de cuota puede trabajar simultaneamente, estas constantes están definidas en el archivo /usr/src/fs/quotadmodel.h y son:

Constante	Valor
MAX_FS	2
MAX_USERS	10

En caso de exceder estos límites el sistema dará aviso por consola.

Se utilizó el byte como unidad de medida de espacio en disco. Una alternativa podría haber sido considerar como unidad de medida un bloque de disco.

El motivo de esta desición esta basado en que no es culpa del usuario el desperdicio de espacio que puede haber en cada bloque de disco que no esté completo. De todos modos el modelo soporta que se cambie la unidad de medida con relativa sencillez.

Archivo /usr/src/fs/quotad.c

Presenta una interfaz de alto nivel para ser utilizada desde dentro del servidor de FS. Esta interfaz realiza las adaptaciones necesarias para abtraer el funcionamiento del nucleo del sistema de cuota (quoadmodel) del funcionamiento del servidor de FS. A continuación presentamos las funcionalidades brindadas por esta interfaz:

Función	Funcionalidad
quotadIsAddQuotaAllowed(fsId,uid,size)	Interroga al sistema de cuota sobre si es posible que el usuario con *uid* consuma *size* bytes en el filesystem con identificación *fsId*. En caso de estar excedido en el uso de espacio retornará un valor falso.
quotadIsRemoveQuotaAllowed(fsId,uid,size)	Interroga al sistema de cuota sobre si es posible que el usuario *uid* deje de utilizar *size* bytes en el filesystem con identificación *fsId*. Esta operación siempre devuelve un valor verdadero.
quotadAddUsedQuota(fsId,uid,size)	Agrega en el sistema de cuota *size* bytes a cargo del usuario *uid* para el filesystem *fsId*.
quotadRemoveUsedQuota(fsId,uid,size)	Remueve del sistema de cuota *size* bytes de la cantidad de espacio utilizada por el usuario *uid* en el filesystem *fsId.*
quotadStart()	Inicia los servicios del sistema de cuota.
quotadStop()	Detiene los servicios del sistema de cuota.
quotadSync()	Escribe a disco los archivos quota.values para cada filesystem donde el sistema de cuota este activo.
QuotadNotifyMount(fsId, mount_point)	Notifica al sistema de cuota la presencia de un nuevo filesystem en el sistema identificado con *fsId* y montado en el directorio *mount_point*. Se intentan leer los archivos quota.values, quota.conf correspondientes al filesystem y activar el sistema de cuota.
quotadNotifyUnMount(fsId)	Notifica al sistema de cuota la ausencia de un filesystem registrado en el sistema de cuota. Se escribe a disco el archivo quota.values para el filesystem identificado con *fsId*. Se liberan los recursos reservados para el mantener la contabilidad del uso de espacio en el filesystem.

System call

Se agregó una entrada a la lista de system calls de Minix, definiendo la llamada número 77 a la función do_quotacmd(). Esta función esta implementada dentro del archivo /usr/src/fs/open.c y está preparada para recibir mensajes de tipo M1 según los tipos standard de mensajes definidos en Minix.

Esta funcionalidad está disponible solo para el superusuario del sistema, a continuación pueden verse los servicios que brinda esta función:

m.m1_i1 (int)	m.m1_i2 (int)	m.m1_p1 (char*)	Funcionalidad
Q_LOADLMT	-	mount point del FS (ASCIIZ)	Cargar en memoria el contenido del archivo quota.conf para el filesystem montado en el directorio m.m1_p1
Q_LOADUSE	-	mount point del FS (ASCIIZ)	Cargar en memoria el contenido del archivo quota.values para el filesystem montado en el directorio m.m1_p1
Q_QUOTAOFF	-	mount point del FS (ASCIIZ)	Dejar de llevar la contabilidad para el filesystem montado en el directorio m.m1_p1
Q_QUOTAON	-	mount point del FS (ASCIIZ)	Continuar con la contabilidad para el filesystem montado en el directorio m.m1_p1
Q_GETQUOTA	uid	mount point del FS (ASCIIZ)	Retorna en m.m1_p2 (char*) una estructura representando la cantidad de bytes utilizados y el límite máximo para el usuario uid en el filesystem montado en el directorio m.m1_p1
Q_ISFSQON	-	mount point del FS (ASCIIZ)	Retorna un valor de verdad informando si el sistema de cuota esta activado para el filesystem montado en m.m1_p1
Q_CONDUMP	-	-	Imprime por consola las principales tablas del sistema de cuota

Para facilitar el acceso a estas funcionalidades se provee una librería (libquota.a) que puede ser linkeada a cualquier programa en lenguaje C que quiera hacer usos de los servicios de este system call.

Esta librería es utiliza por los programas de usuario escritos para poder administrar el sistema de cuota.

Librería /usr/lib/i386/libquota.a

La librería funciona como interfaz para utilizar las funcionalidades provistas por el system call presentado en la sección anterior. Las funciones provistas por esta librería son:

Nombre	Parametros
quotaon	(char* mountPoint)
quotaoff	(char* mountPoint)
getquota	(int uid, char* mountPoint, struct QuotaEntry*)
loadqlimit	(char* mountPoint)
loadquse	(char* mountPoint)
isquotaon	(char* mountPoint)

Todas ellas retornan un valor entero indicando el éxito o fracaso de la operación. En caso de error se retornan diversos valores para poder distinguir el error.

Aplicaciones de usuario

La implementación de los comandos de usuario resulto relativamente sencilla, puesto que son una combinación de la utilización del módulo quotadparser, quotadmodel y la librería libquota.a para realizar la lectura e interpretación de los archivos quota.conf, quota.values y comunicar al quotad los cambios realizados por el usuario.

Archivo /usr/src/quota/commands/qsck.c:

Este programa es el encargado de realizar el recorrido del arbol de directorios de un filesystem dado por parámetro, con el objetivo de contabilizar la cantidad de espacio utilizada por cada usuario que tenga un archivo en ese filesystem.

La primer implementación de esta aplicación fue un script de shell que realizaba un análisis de la salida generada por una llamado al comando /usr/bin/ls y realizaba el recorrido y contabilidad. Esta solución era extremadamente ineficiente y muy lenta (también muy poco elegante), así que fue necesario buscar otra alternativa de implementación.

Una alternativa evaluada fue escribir un programa C que realizara la lectura de la tabla de inodos del filesystem a analizar, realizando la contabilidad con la información contenida en cada inodo. Esta solución debería ser muy veloz ya que no es necesario recorrer el arbol de directorios del filesystem (evitando accesos a disco), sino que se consideran los archivos en el orden que aparecen en la tabla. Si bien el beneficio en velocidad parece prometedor, el incremento en la dificultad por tener acceso e interpretar la zona de disco conteniendo la tabla de inodos nos hizo buscar otra alternativa mas sencilla.

La solución implementada realiza el recorrido del arbol de directorios utilizando las funciones de librería opendir(), readdir(), closedir() y chdir() lo que brinda un balance razonable entre velocidad y simplicidad del programa.

Los pasos realizados por el programa pueden resumirse de este modo:

1. Realizar el recorrido del arbol de directorios para el filesystem contabilizando el uso del espacio.
2. Detener los servicios del quotad para el filesystem analizado.
3. Escribir el archivo quota.values en el directorio raiz del filesystem.
4. Reanudar los servicios del quotad para el filesystem analizado (quotad lee el archivo quota.values).

La desactivación temporal de los servicios del quotad fue explicada en la sección *Sincronización*.

Una cuestión a atender durante la implementación fue la forma de contabilizar los multiples links que puede tener un archivo. Minix no permite la existencia de links entre distintos filesystems, tampoco permite que distintos links a un archivo tengan distinto propietario. Estos dos factores y el hecho de que existe un campo indicando la cantidad de links al archivo asociado a cada inodo, facilitó el problema de no contar los múltiples links a un archivo como si fueran distinas instancias del archivo.

Un problema distinto aparece al considerar que el filesystem a analizar puede sufrir modificaciones mientras se esta contabilizando su uso. El problema se presenta si mientras se esta realizando la contabilidad de un filesystem, algún usuario escribe un archivo en un directorio que ya fue analizado, antes de que el análisis termine. En esta situación la escritura del usuario no sería contabilizada.

Una posible solución a este problema consiste en relizar modificaciones al servidor de FS para que admita un mensaje que tenga por semantica negar las escrituras a un filesystem hasta recibir otro mensaje que indique lo contrario. De esta manera al comenzar el análisis del filesystem se podrían inhibir las escrituras, y una vez terminado el análisis volver a habilitarse.

Esta solución impide que un usuario realice escrituras mientras se esta realizando el recorrido del arbol de directorios. Sin embargo al momento de escribir el archivo quota.values es necesario que esté habilitada la escritura a disco, en ese momento es posible que se realice alguna escritura de usuario, arruinando nuevamente la presición de la contabilidad realizada. Consideramos esta pérdida de presición aceptable, aunque la dificultad de implementación asociada y las consecuencias prácticas de negar las escrituras a disco nos parecen demasiado costosas y nos llevo a buscar una solución mas simple.

La solución implementada se limita a exigir al usuario que el filesystem se encuentre en modo de solo lectura a fin de evitar escrituras mientras se realiza el recorrido del arbol de directorios. Una vez finalizado el análisis se monta el filesystem analizado en modo escritura, se escribe el archivo quota.values y nuevamente se vuelve a montar el filesystem en modo solo lectura. Para realizar el montado de los filesystem se utilizan las llamadas de librería mount() y umount().

Con estas consideraciones, los pasos a realizar son:

1. Exigir el filesystem en modo lectura.
2. Realizar el recorrido del arbol de directorios para el filesystem contabilizando el uso del espacio.
3. Detener los servicios del quotad para el filesystem analizado.
4. Desmontar el filesystem.
5. Montar el filesystem en modo lectura/escritura
6. Escribir el archivo quota.values en el directorio raiz del filesystem.
7. Desmontar el filesystem.
8. Montar el filesystem en modo lectura.
9. Reanudar los servicios del quotad para el filesystem analizado (quotad lee el archivo quota.values).

Esta solución sufre del mismo problema de falta de presición en la contabilidad que el caso anterior, un usuario podría realizar una escritura a disco mientras se esta escribiendo el archivo quota.values y aun no se reanudó el servicio del quotad.

Otro problema es la posibilidad de que un usuario realice una operación de apertura para escritura en el breve instante en que el filesystem esta montado como lectura/escritura. Si eso sucediera, el llamado del paso 7 para desmontar el filesystem no sería posible y evitaría que qsck termine en forma correcta.

Para paliar esta situación se incluyó código para realizar el manejo de errores y situaciones anómalas, a fin de informar al usuario acerca del posible problema.

Archivo /usr/src/quota/commands/setquota.c:

Este programa es el que permite al administrador del sistema asignar el límite de espacio que un usuario puede utilizar en un filesystem dado por parametro.

La implementación realiza las siguiente tareas, verificando condiciones de error y reportandolas al usuario en caso de haberlas:

1. Inicializar una instancia de quotadmodel.
2. Cargar el achivo quota.conf del filesystem dentro del quotadmodel.
3. Modifcar dentro del quotadmodel el registro correspondiente al usuario.
4. Escribir el archivo quota.conf al filesystem.
5. Indicar al quotad que debe refrescar la información de límite de espacio para los usuarios del filesystem.

El programa hace uso intensivo del módulo quotadmodel y de las funciones contenidas en la librería libquota.a para comunicarse con el quotad.

Archivo /usr/src/quota/commands/getquota.c:

Este programa es el que permite al administrador del sistema conocer el límite de espacio que un usuario puede utilizar y la cantidad que actualmente está utilizando en un filesystem dado por parametro.

La implementación realiza las siguiente tareas, verificando condiciones de error y reportandolas al usuario en caso de haberlas:

1. Verificar que el sistema de quota este activo en el filesystem.
2. Solicitar al quotad la información de cuota asociada al usuario.
3. Imprimir por pantalla la información.

Al igual que en el programa /usr/src/quota/commands/setquota.c este programa hace uso de las funciones de la librería libquota.a para comunicarse con el quotad.

Archivo /usr/src/quota/commands/quota.c:

Este programa permite activar o desactivar los servicios del qutoad para un filesystem. Para esto utiliza funciones de la librería libquota.a para comunicarse con el quotad.

Durante la activación del quotad para el filesystem se verifica la existencia de los archivos quota.conf y quota.values, en caso de no existir la activación no tendrá éxito y se reportará error al usuario.

Durante la desactivación del quotad para el filesystem se escribe el archivo quota.values para dar persistencia a la información contabilizada.

Modificaciones al servidor de FS

La siguiente tabla presenta las modificaciones realizadas al código fuente del servidor de FS para incorporar la funcionalidad del quotad:

Archivo /usr/src/fs/main.c:

Función	Descripción	Cambio realizado
main	Inicializa el servidor de FS y realiza la invocación de los system calls implementados en el servidor de FS.	Inicializar los servicios del quotad. Notificar al qutoad el montado del filesystem raiz.

Archivo /usr/src/fs/misc.c:

Función	Descripción	Cambio realizado
do_exit	Realiza una parte del system call exit() liberando el lugar que ocupaba el proceso en la tabla de procesos.	Detectar la finalización del proceso con ID 1 (proceso init) indicando el shutdown del sistema. Escribir a disco los archivos quotad.values a cada filesystem activo en el quotad.

Archivo /usr/src/fs/mount.c:

Función	Descripción	Cambio realizado
do_mount	Implementa el system call mount() realizando el montado de un filesystem al arbol de directorios.	Notificar al quotad el agregado de un nuevo filesystem. (produce la lectura de los archivos quota.values y quota.conf, también la activación del quotad para el filesystem)
do_umount	Implementa el system call umount() realizando el desmontado de un filesystem previamente montado con el system call mount().	Notificar al quotad la remoción de un filesystem. (produce la escritura al filesystem del archivo quota.values previo al desmontado)

Archivo /usr/src/fs/open.c:

Función	Descripción	Cambio realizado
common_open	Implementa parte del system call open() para la apertura de archivos.	Descontar el espacio utilizado por un archivo que es abierto para ser truncado, al propietario del archivo.
do_quotacmd	Implementa el system call quotacmd() que permite la comunicación de procesos de usuario con el quotad.	Se agregó la función do_quotacmd().

Archivo /usr/src/fs/protect.c:

Función	Descripción	Cambio realizado
do_chown	Implementa el system call chown() para cambiar el usuario propietario de un archivo.	Transferir la utilización del espacio del archivo desde el antiguo dueño del archivo, al nuevo dueño.

Archivo /usr/src/fs/table.c:

Función	Descripción	Cambio realizado
-	Contiene un arreglo con las funciones que implementan los distintos system call.	Agregar una entrada para el system call do_quotacmd().

Archivo /usr/src/fs/read.c:

Función	Descripción	Cambio realizado
read_write	Implementa funcionalidades comunes a los system calls read() y write().	Informar al qutoad la escritura de un usuario a un archivo para realizar la contabilidad.

Instalación del sistema de cuota

El sistema de cuota está contenido en el archivo quota.tar.Z que se entrega en soporte digital junto con este informe. Los fuentes estan preparados para correr en una distribución conteniendo los fuentes originales de la version 2.0.2 de Minix.

Para realizar la instalación de los fuentes modificados ingresar el sistema como el usuario root y realizar los siguientes pasos:

1. Crear el directorio /tmp/quotaDist
2. Copiar el archivo quota.tar.Z al directorio /tmp/quotaDist
3. Correr los siguientes comandos:
    a. cd /tmp/quotaDist
    b. uncompress quota.tar.Z
    c. tar xvf quota.tar
    d. cd quota
    e. make

Una vez instalado el código fuente en los directorios apropiados, correr los siguientes comandos:

1. cd /usr/src/quota
2. make all

Lo cual generará los binarios y librerías necesarias para la operación del sistema de cuota.

Luego utilizar el procedimiento habitual para generar el kernel del sistema.

Operación del sistema de cuota

El sistema de cuota se inicia automaticamente junto con el kernel del sistema operativo. Cada vez que se monta un filesystem el quotad intenta abrir los archivos quotad.conf y quotad.values ubicados en el directorio raiz del filesystem montado.

En caso de poder leer esos archivos se incorpora al filesystem dentro de los filesystems activos en el quotad. En esta situación el quotad realiza la contabilidad y limita la escritura a los usuarios que excedan su cuota. La limitación se manifiesta devolviendo error en las peticiones de escritura, el error devuelto es ENOSPC (Error por falta de espacio en el dispositivo).

En caso de no poder leer esos archivos se imprime un mensaje de aviso por la consola y se incorpora al filesystem dentro de los filesystem inactivos del quotad. Posteriormente es posible intentar iniciar los servicios del quotad para el filesystem mediante un comando de usuario.

A continuación se presentan los comandos disponibles para el administrador del sistema:

Comando	/bin/qsck
Sintaxis	qsck mount_point [-f]
Propósito	Genera el archivo quota.values para el filesystem montado en el directorio **mount_point** en modo solo lectura.
Parámetros	El parametro **–f** permite forzar la generacion del archivo quota.values aun cuando el filesystem este montado en modo lectura/escritura.

Comando	/bin/setquota
Sintaxis	setquota uid quota mount_point
Propósito	Asigna al usuario con **uid** la cantidad de bytes **quota** como límite de uso del filesystem montado en el directorio **mount_point**.

Comando	/bin/getquota
Sintaxis	getquota uid mount_point
Propósito	Imprime por pantalla la cantidad de espacio utilizado y la cuota asignada al usuario **uid** en el filesystem montado en el directorio **mount_point**.
Notas	Es necesario que el sistema de quota este activado para el filesystem montado en el directorio **mount_point**. Caso contrario el comando fallará.

Comando	/bin/quota
Sintaxis	quota [on | off] mount_point
Propósito	Permite activar o desactivar los servicios del quotad para el filesystem montado en el directorio **mount_point**.
Notas	Para que la activación tenga éxito es necesaria la existencia de los archivos quota.values y quota.conf en el directorio raiz del filesystem montado en el directorio **mount_point**. Caso contrario el comando fallará.

Ejemplo de configuración

Los pasos descriptos a continuación prenten facilitar la configuración inicial del sistema de cuota, y requieren el sistema de cuota instalado e incorporado al kernel de Minix. El ejemplo se realiza con el filesystem /usr pero puede aplicarse a cualquier otro.

Generar la base de datos con el uso del espacio del filesystem:

```
# qsck /usr –f
```

Asignar cuota a los usuarios del sistema, al usuario con uid 8 se asigna una cuota de 100000 bytes y al usuario con uid 9 se asignan 200000:

```
# setquota 8 100000 /usr
# setquota 9 200000 /usr
```

Activar el sistema de cuota para el filesystem /usr:

```
# quota on /usr
```

Con la configuración realizada, la proxima vez que inicie el sistema operativo el sistema de cuota se activará automaticamente para el filesystem /usr.