GoboHide: Uma Solução Flexível e Escalável para Inodes Ocultos no Kernel Linux

Felipe W Damasio **Lucas Correia Villa Real**
felipewd@terra.com.br **lucasvr@gobolinux.org**

Abstract. *This paper presents a robust and flexible solution to allow proposals of restructure of the legacy directory tree of the GNU/Linux operating system to keep full compatibility with the legacy tree, keeping it invisible to the user. A tool for managing the legacy tree was developed and is being used by the GoboLinux distribution.*

Resumo. *Este artigo apresenta uma solução robusta e flexível para permitir que propostas de reestrutração da árvore de diretórios do sistema operacional GNU/Linux mantenham compatibilidade com a árvore de diretórios tradicional sem que esta seja visível para o usuário. Uma ferramenta para administração da árvore clássica foi desenvolvida e está sendo utilizada pela distribuição GoboLinux.*

1. Introdução

Em virtude do ritmo rápido de desenvolvimento do software livre, o processo de instalação e remoção de programas tornou-se comum e freqüente, diferentemente do que ocorria quando do estabelecimento da hierarquia de diretórios do UNIX, na qual as distribuições do GNU/Linux usualmente se baseiam. Os critérios de organização da árvore de diretórios tradicional não levam em conta estas necessidades; torna-se interessante realizar uma revisão destes critérios e buscar uma alternativa [Muhammad and Detsch, 2002].

A adoção de uma nova árvore de diretórios para o sistema operacional GNU/Linux facilita a manutenção e a atualização de programas, bem como a organização geral do sistema. Porém, existe a necessidade de manter a compatibilidade com a árvore de diretórios tradicional do UNIX, pois muitos programas tem *paths hardcoded* baseados na árvore clássica. É interessante um mecanismo que permita que seja mantida a compatibilidade com a árvore clássica de maneira transparente para qualquer processo, permitindo também ao usuário apenas a visualização da nova árvore de diretórios.

Este artigo apresenta uma solução simples, escalável e de fácil administração para dissimular a existência desta árvore de diretórios tradicional, possibilitando que uma nova árvore de diretórios seja usada sem que seja perdida a compatibilidade com aplicações que utilizam a árvore clássica do GNU/Linux.

O restante deste artigo está organizado da seguinte maneira: Na Seção 2 será vista a abordagem utilizada em outros sistemas operacionais, bem como a abordagem utilizada pelo GoboLinux; na Seção 3 será descrita a implementação do GoboHide no kernel Linux; na Seção 4 será descrita uma ferramenta administrativa feita em espaço de usuário que é utilizada pelo administrador do sistema para gerenciar a árvore *legacy* de diretórios; na Seção 5 o artigo é concluído, deixando alguns pontos abertos que podem ser abordados como trabalhos futuros.

2. Abordagens alternativas

Atualmente, a distribuição GoboLinux[1] utiliza a mesma estratégia do sistema operacional Mac OS X[Apple Computer Inc., 2001], no qual *links* são usados para poder ser criada uma nova árvore de diretórios e manter a compatibilidade com programas com *legacy paths hardcoded*.

[1] ver `http://www.gobolinux.org`

A grande vantagem deste tipo de abordagem está em sua simplicidade, já que o kernel transparentemente re-envia requisições dos *links* simbólicos aos novos diretórios, mantendo total compatibilidade.

No entanto, como a reestruturação da árvore de diretórios visa simplificar a representação desta para usuários domésticos, o uso de *links* simbólicos traz uma grande desvantagem, já que as duas árvores (a árvore proposta e a clássica) são vistas pelo usuário. Para isso, o Mac OS X utiliza uma estratégia incomum para "esconder" a árvore clássica do usuário. Na interface gráfica é exibida uma árvore de diretórios Macintosh, contendo diretórios como */System/Library* e */Network/Remote_Workstation*. Além disso, a interface exibe alguns diretórios em localizações diferentes da real. Por exemplo, o diretório */Mac OS X* é um *link* para o diretório raiz, e alguns diretórios, como */Applications*, aparecem na interface apenas como */Mac OS X/Applications*. No entanto, acessando-se o sistema de arquivos através de um *shell*, tornam-se acessíveis diretórios UNIX como */usr* e */etc*.

Esta abordagem é somente possível em um ambiente proprietário, onde toda a interface primária do sistema é desenvolvida por apenas uma empresa. Além disso, falhas no *design* como a possibilidade de ver a árvore de diretórios através do uso de um *shell* mostram que a abordagem utilizada pelo Mac OS X é inutilizável em um sistema operacional tão heterogêneo como o GNU/Linux. Na Seção 3 é apresentada a solução adotada pelo GoboLinux para esconder a árvore de diretórios *legacy*.

Nota-se também que o ideal é permitir que seja usada qualquer interface com o usuário e qualquer programa de gerenciamento de diretórios de maneira transparente, acessando (e visualizando) apenas a nova árvore de diretórios sem que seja necessária a modificação desses programas para adaptação à nova árvore de diretórios.

3. Implementação no Kernel Linux

A representação de arquivos no kernel Linux é feita com o uso de identificadores denominados *inode numbers*. Estes números são únicos para todos os arquivos em uma dada partição.

Para realizar a leitura de um diretório, é utilizada a chamada de sistema `readdir`. Esta chamada de sistema realiza a leitura de um diretório especificado, copiando seu conteúdo para uma região alocada na memória. Desta forma, para a implementação do GoboHide, basta verificar para cada elemento lido nesta operação se ele é um dos diretórios ou um link simbólico da árvore clássica. Esta informação é obtida a partir do *inode number* referenciado pelas estruturas de entrada de diretório. Caso o *inode* lido seja um dos que deseja-se ocultar, esta entrada simplesmente não é copiada para a região de memória (operação comumente referenciada como *filldir*), e o próximo inode é processado. Como a chamada de sistema `readdir` é implementada em praticamente todos os sistemas de arquivos, ela torna-se um local plausível para realizar a interceptação.

Através da modificação de apenas esta chamada de sistema, resolvemos o problema de visualização da árvore clássica de maneira transparente para qualquer processo, desde que o sistema de arquivos no qual a árvore clássica está localizada faça a checagem das estruturas GoboHide – composta por itens descritivos sobre os inodes sendo ocultados, disponibilizados em uma lista.

Além disso, como apenas a chamada de sistema `readdir` está sendo interceptada, *paths hardcoded* em programas que referenciam a árvore clássica funcionarão normalmente, já que as chamadas de sistema `open` e `close` não são modificadas, e pela utilização de links simbólicos o acesso e redirecionamento à nova árvore de diretório é também feito pelo Kernel (na camada VFS) de maneira transparente.

O problema de ser interceptada a chamada `readdir` para consultas na árvore clássica é manter a consistência da lista e permitir que essa estrutura possua um acesso rápido, já que a chamada de sistema `readdir` é utilizada sempre que um processo deseja ver o conteúdo de um diretório.

Para manter a consistência da lista, foram modificadas as chamadas de sistemas `unlink` e `rmdir` na camada VFS, permitindo uma transparência na gerência dos inodes tanto para qualquer

processo como para qualquer sistema de arquivos. Além disso, na chamada de sistema `unlink` apenas links simbólicos são checados, já que esta chamada é utilizada para a remoção de *qualquer* arquivo. Com isso, é diminuído o *overhead* de checagens na lista GoboHide, pois apenas remoções de diretórios/links simbólicos fazem consultas a ela.

Métodos de leitura da lista de inodes são feitos de maneira concorrente, porém apenas um metódo que deseja modificar a lista pode ser utilizado por vez (através da utilização de *Read-Write Locks*), sendo que são utilizados algoritmos de complexidade **O(1)** para o acesso a lista de inodes, garantindo um tempo constante de acesso à lista de inodes que deverão ser ocultados, ou seja, independente do número de inodes armazenados na lista.

Também foram utilizados *locks* que salvam o contexto de IRQ e desabilitam interrupções no processador local, não permitindo que a estrutura de checagem de *inodes* seja danificada por acessos concorrentes em um sistema SMP (*Symmetric Multi-Processing*) ou mesmo num sistema UP (*Uni-Processing*), bem como não são permitidos que hajam *deadlocks* na aquisição de *write locks*.

Na implementação do GoboHide, foi utilizado o conceito de *embedded ioctls*, no qual em apenas uma `ioctl` (genérica e implementada na camada VFS) é passado um argumento, o qual contém um conjunto de uma ou mais `ioctl` criadas a partir de um *bitwise* OR entre ioctls GoboHide.

Além da adição/remoção/consulta de *inode numbers*, é ainda fornecido ao usuário uma *ioctl* para traduzir os *inode numbers* armazenados na estrutura GoboHide para o *pathname* completo dos diretórios e/ou *links* simbólicos mantidos nesta estrutura.

4. Administração da Árvore *legacy* em espaço de usuário

Como a implementação GoboHide no Kernel Linux cria um *framework* para gerência da árvore clássica do GNU/Linux, foi necessário desenvolver uma aplicação em espaço de usuário para utilizar as operações disponíveis pelo GoboHide-Kernel.

A ferramenta fornece ao administrador do sistema uma interface simples e transparente para ocultar diretórios e/ou *links* simbólicos da árvore *legacy* do GNU/Linux. Ela possui uma CLI (*Command Line Interface*) com as seguintes opções:

```
$ gobohide --help

gobohide: Hide/Unhide a directory

-h, --hide     Hide the directory
-u, --unhide   Unhide the directory
-l, --list     List the hidden directories
    --version  Show the program version
    --help     Show this message
```

A ferramenta basicamente identifica (utilizando a chamada de sistema `stat`) o *inode number* do diretório e/ou link simbólico que o administrador do sistema deseja ocultar e cria a devida GoboHide Embedded ioctl, utilizando a ioctl genérica FIGOBOLINUX (que é a ioctl genérica das estruturas GoboHide) para comunicação com o GoboHide-Kernel.

Tendo acrescentado, por exemplo, o diretório */etc* à lista de diretórios ocultos pelo GoboHide, o administrador do sistema **não** precisa removê-lo (utilizando a opção '-u') da lista de ocultos caso ele seja removido do sistema, pois o GoboHide-kernel faz a detecção automática pela modificação das chamadas de sistemas `unlink` e `rmdir`, e transparentemente na próxima consulta às estatísticas do GoboHide o administrador irá notar que a lista atual possui apenas entradas consistentes (não desperdiçando memória de kernel).

As "estatísticas", que informam ao administrador do sistema quais diretórios estão sendo escondidos pelo GoboHide, são alocadas em kernel, porém em contexto de usuário, ou seja: a memória utilizada para alocar as estruturas necessárias para as estatísticas é obtida utilizando memória virtual do usuário, não fazendo uso da memória de kernel para isso.

5. Conclusões e Trabalhos Futuros

Através da utilização do GoboHide é possível manter compatibilidade de uma maneira simples e consistente entre uma nova árvore de diretórios e a árvore clássica do sistema operacional UNIX, sem que a árvore de diretórios clássica fique visível para o usuário, como é visto na distribuição GoboLinux[Muhammad and Detsch, 2002].

Como a implementação do GoboHide está em kernel, porém com uma interface administrativa em espaço de usuário, existe uma transparência para qualquer processo da árvore clássica, porém programas antigos com *paths hardcoded* baseados na árvore clássica funcionam corretamente, devido à não modificação nas chamadas de sistema `open` e `close`.

O *design* do GoboHide em kernel foi feito de forma a possibilitar, através do uso de *embedded ioctls*, a adição de novas operações ao framework de operações suportadas pela ferramenta administrativa em espaço de usuário, permitindo que outras distribuições baseadas no kernel Linux possam usufruir dessas operações.

Nota-se ainda que como o GoboHide precisa acessar as estruturas dependentes de sistema de arquivos para interceptar o *filldir* na chamada de sistema *readdir*, nem todos os sistemas de arquivos suportados pelo kernel Linux são suportados pelo GoboHide. Uma possível solução para esse problema é a utilização de *Stackable File Systems*[Heidemann, 1995], no qual é acrescentando mais um nível de indireção na camada VFS, no qual seria possível criar um sistema de arquivos "virtual" chamado GoboHide que interceptaria a chamada de sistema `readdir` e passaria para o sistema de arquivos específico apenas os inode numbers que não devem ser ocultados.

Referências

Apple Computer Inc. (2001). *Inside Mac OS X - System Overview*. ISBN: 1400524806.

Heidemann, J. S. (1995). Stackable Design of File Systems. Technical report, University of California, Los Angeles. Ph.D. dissertation.

Muhammad, H. and Detsch, A. (2002). Uma nova proposta para a árvore de diretórios UNIX. Proceedings of the III WSL - Workshop em Software Livre.