Adicionando Suporte a Atributos Estendidos no Sistema de Arquivos RamFS

Fabiane Cristine Dillenburg, Rossana Baptista Queiroz, Cristiano André da Costa[1]

[1]Universidade do Vale do Rio dos Sinos (UNISINOS)
Av. Unisinos 950 – 93022-000 – São Leopoldo – RS – Brasil

{fabianecd,fellowsheep}@gmail.com, cac@unisinos.br

Abstract. *This paper presents the main characteristics of the extended attributes and describes the implementation of its support on RamFS non-persistent filesystem. It was implemented the* user namespace *in the RamFS filesystem. This work allows applications running on* livecds *to use extended attributes, improving their performance.*

Resumo. *Este artigo apresenta as principais características dos atributos estendidos e descreve a implementação do seu suporte no sistema de arquivos não persistente RamFS. Foi implementado o* namespace user *no sistema de arquivos RamFS. Esta implementação permite que aplicações rodando em* livecds *possam utilizar atributos estendidos, melhorando sua performance.*

1. Introdução

Atributos estendidos (*xattr*s) adicionam informações extras nos *inodes*. Eles implementam a possibilidade de adicionar pares de *nome:valor* a objetos dentro do sistema de arquivos. Assim, podem ser usados para armazenar metadados sobre os arquivos.

Pode-se utilizar atributos estendidos para diversos fins, como caracterizar as informações contidas no arquivo ou incluir funcionalidades de segurança (permissões, por exemplo). Exemplos de aplicações que usam atributos estendidos no Linux são as Listas de Controle de Acesso POSIX (ACLs) [Grünbacher 2006] e o aplicativo de busca Beagle [Beagle 2006b].

No Linux, o suporte a atributos estendidos não está implementado em todos os sistemas de arquivos. Entre os sistemas de arquivos que não possui esse suporte está o RamFS [Ramfs 2007]. O RamFS é um sistema de arquivos que mantém todos os arquivos na memória volátil (RAM). Nesse sistema de arquivos, são permitidos acessos de leitura-escrita e a quantidade de memória RAM utilizada adapta-se ao seu conteúdo. Apesar de não ser um sistema de arquivos persistente, o RamFS é bastante utilizado, por exemplo, nos *livecds*.

De acordo com a sua documentação, o RamFS é considerado um sistema de arquivos bastante didático, sendo assim, ideal para a aprendizagem do funcionamento de um sistema de arquivos. Esta característica, aliada ao fato do RamFS não suportar atributos estendidos, permitiu a definição do seguinte objetivo: adicionar suporte a atributos estendidos no sistema de arquivos RamFS.

O restante do artigo está organizado da seguinte forma: a Seção 2 relaciona os fundamentos dos atributos estendidos. A Seção 3 apresenta detalhes da implementação.

Por fim, a Seção 4 encerra o artigo com considerações finais e perspectivas de trabalhos futuros.

2. Fundamentos

Atributos estendidos são pares de *nome:valor* que podem ser associados a arquivos ou diretórios em um sistema de arquivos. Seu objetivo é adicionar informações extras sobre arquivos ou diretórios nos *inodes*, que usualmente são chamados de metadados. Os metadados podem ser úteis para agilizar vários processos como, por exemplo, tornar desnecessária a leitura dos cabeçalhos de um arquivo para buscar a informação de tipo do mesmo. O nome e valor podem ser descritos por qualquer *string*, limitados por tamanho determinado na implementação do sistema de arquivos.

No Linux, os sistemas de arquivo Ext2, Ext3, ReiserFS, XFS, JFS e NFS suportam atributos estendidos. Sua utilização requer a seleção da `libattr` durante a compilação do *kernel*, sendo ainda necessário habilitar a opção de montagem `user_xattr` para os sistemas de arquivo Ext2 e Ext3. Qualquer arquivo regular pode ter uma lista de atributos estendidos.

Cada atributo é denotado por um nome e dados associados. O nome precisa ser uma cadeia de caracteres terminada em nulo (*null-terminated string*) e precisa ser prefixada por um identificador de *namespace* (que será explicado na Subseção 2.1) e um ponto.

Por ser um recurso relativamente recente no Linux (suportados a partir da versão 2.6 do *kernel*) e ainda não suportado por todos os sistemas de arquivos, poucas aplicações de usuário fazem uso dos atributos estendidos. Podem-se destacar, entretanto, algumas aplicações pioneiras no seu uso, como o Beagle, o FreeDesktop.org e o SELinux, cada uma utilizando-os com uma finalidade específica:

- Beagle [Beagle 2006b]: utiliza atributos estendidos para especificar se o arquivo já foi indexado. Exemplos de atributos estendidos [Beagle 2006a]:
  - `user.Beagle.AttrTime`;
  - `user.Beagle.Fingerprint`;
  - `user.Beagle.MTime`;
  - `user.Beagle.Uid`.
- freedesktop.org [freedesktop.org 2006]: incentiva o uso de atributos estendidos no desenvolvimento de ambientes gráficos para Linux e outros sistemas operacionais *Unix-like*. Exemplos de atributos estendidos:
  - `user.xdg.comment`;
  - `user.xdg.creator`;
  - `user.xdg.origin.url`;
  - `user.xdg.origin.email.subject`;
  - `user.xdg.origin.email.from`;
  - `user.xdg.origin.email.message-id`.
- Security Enhanced Linux (SELinux) [SELinux 2006]: provê uma política de segurança sobre todos os processos e objetos do sistema baseando suas decisões em *labels* (rótulos) contendo uma variedade de informações relevantes à segurança. A lógica da política de tomada de decisões é encapsulada dentro de

um simples componente conhecido como servidor de segurança (*"security server"*) com uma interface geral de segurança. Ele usa atributos estendidos como a base de seu controle de acesso [LinuxQuestions.org 2006].

2.1. *Namespaces*

Usualmente, o nome dos atributos são no formato `namespace.name` (*user.filetype*, por exemplo). A primeira parte refere-se a um *namespace*, para evitar "colisões" entre atributos de mesmo nome (definido pela segunda parte).

Atualmente, no Linux, existem quatro namespaces: (i) *user*, (ii) *trusted*, (iii) *security* e (iv) *system*. O primeiro não tem restrições quanto a nomes e conteúdos. Além disso, esse *namespace* é semelhante ao *trusted*. A diferença entre eles está no fato do segundo ser restrito ao super usuário do sistema. O *namespace system* é usado, geralmente, pelo *kernel* para acessar listas de controle ACLs como, por exemplo, os atributos `system.posix_acl_access` e `system.posix_acl_default`. O *namespace security*, por sua vez, é usado pelo SELinux através do atributo `security.selinux`.

2.2. Uso

Esta subseção apresenta exemplos de como manipular os valores dos atributos estendidos.

- Exemplo de como alterar um atributo estendido:

```
$ setfattr -n user.filetype -v mp3 testfile
```

- Exemplo de como recuperar um atributo estendido:

```
$ getfattr -n user.filetype testfile
```

A saída deve ser algo como:

```
# file: testfile
user.filetype="mp3"
```

- Exemplo de como remover um atributo:

```
$ setfattr -x user.filetype testfile
```

3. Implementação

A implementação do suporte a atributos estendidos para o RamFS foi feita no *kernel* 2.6.18.2 [Linux 2006]. Durante a compilação e os testes das alterações, a arquitetura alvo foi a UML (*User-Mode* Linux) [UML 2006]. A arquitetura UML provê uma máquina virtual que permite executar o *kernel* do Linux e outros processos de maneira segura. Assim, modificações no código-fonte do *kernel* podem ser testadas com segurança nesta arquitetura, antes de serem executadas diretamente na máquina.

Utilizou-se como material para o início da implementação um *patch* para o *kernel* 2.6.8.1 [1]. Esse *patch* também tinha por objetivo adicionar suporte a atributos estendidos no RamFS. No entanto, ele implementava apenas o *namespace security*.

[1] Disponível em `http://www.ussg.iu.edu/hypermail/linux/kernel/0408.2/2329.html`. Acesso em fevereiro de 2007.

Considerando a evolução do *kernel*, que aconteceu entre a versão do *patch* mencionado e a versão utilizada para esta implementação, a primeira tarefa foi a adaptação do *patch* para os novos cabeçalhos de algumas funções, de acordo com a versão 2.6.18.2 do *kernel*. A adaptação dos cabeçalhos foi feita sem grandes dificuldades. Entretanto, surgiram problemas relacionados às funções que eram chamadas para o armazenamento dos atributos nos *inodes*.

No *patch* acima referenciado, o armazenamento dos atributos nos *inodes* era feita através de funções da biblioteca `include/linux/security.h`. Essas funções (`security_inode_getsecurity` e `security_inode_setsecurity`) são protegidas pela macro `CONFIG_SECURITY`. Para utilizá-las, o Linux precisa ser configurado com suporte a segurança. Isso pode ser feito pela opção *"Security options"*, habilitando *"Enable different security models"* no *menuconfig* do *kernel*. Porém, mesmo compilando o Linux com esse suporte, as rotinas *get* e *set* não funcionaram. Não foram feitos testes exaustivos para verificar a origem do problema.

Neste contexto, optou-se por criar métodos próprios `RamFS_xattr_get` e `RamFS_xattr_set`, para as rotinas de *get* e *set* respectivamente, como os outros sistemas de arquivos (por exemplo: Ext3, ReiserFS) fazem. Para tal, foi estudado o funcionamento do suporte a atributos estendidos no Ext3. Observou-se que, devido à persistência dos dados, o Ext3 armazena os atributos estendidos nos blocos dos *inodes*. Como no RamFS é tudo mantido na memória RAM, optou-se por armazenar os atributos nos *inodes* da seguinte maneira:

1. Utilizar o campo `generic_ip` da `struct inode` para armazenar os atributos. Nota-se que a `struct inode` tem esse campo reservado só pra colocar atributos extras que, eventualmente, queira-se carregar junto com cada *inode*. Esse campo está dentro da `union u`. Vale destacar que o campo mencionado é do tipo `void *`.
2. Alocar dinamicamente uma lista (`list_head`) na criação do arquivo. Esse procedimento é realizado na função `RamFS_get_inode`.
3. Vincular essa lista ao `(struct inode *)->u.generic_ip`. Esse procedimento, também, é realizado na função `RamFS_get_inode`.
4. Para cada chamada ao `setxattr`, alocar uma estrutura contendo os dados do atributo estendido (nome e valor) e adicionar à lista. A rotina `setxattr` é responsável por chamar a função de *set* do *handler* do *namespace* do atributo que se deseja setar o valor. Assim, no caso desta implementação, a `setxattr` chama a rotina `RamFS_xattr_user_set` que, por sua vez, chama a rotina `RamFS_xattr_set`. Logo, a adição do novo atributo ao campo `generic_ip` ocorre na função `RamFS_xattr_set`.
5. Para cada chamada do `getxattr`, percorrer a lista e retornar o atributo, se for encontrado. A rotina `getxattr` é responsável por chamar a função de *get* do *handler* do *namespace* do atributo que se deseja recuperar o valor. De forma semelhante ao descrito na etapa anterior. Nesta implementação, a `getxattr` chama a rotina `RamFS_xattr_user_get` que, por sua vez, chama a rotina `RamFS_xattr_get`. Logo, percorre-se a lista associada ao campo `generic_ip`, em busca do atributo desejado, na função `RamFS_xattr_get`.

Dessa maneira, foi implementado o *namespace user*. A idéia inicial de implementar o *namespace security* foi abortada em função das restrições encontradas nas funções

da biblioteca, já mencionada, `include/linux/security.h`.

Para a implementação, em `fs/RamFS`, foram criados os aquivos: `xattr.h`, `xattr.c` e `xattr\_user.c`. Além da inclusão destes arquivos, também foi alterado o arquivo `Kconfig` do sistema de arquivos. Foram, ainda, alterados arquivos já existentes no diretório do RamFS, como o `Makefile`, `file-mmu.c` e o `inode.c` para incluir as informações de atributos estendidos aos *inodes* e, inclusive, relacionar a lista de atributos ao campo `generic_ip` do *inode* na função `RamFS_get_inode`.

A estrutura que armazena os dados do atributo estendido está em `xattr.h`, assim como a `list_head` dos xattrs vinculada ao campo do *inode*:

```
struct RamFS_xattr_info {
  int name_index;
  const char *name;
  const void *value;
  size_t value_len;
};
```

Essa estrutura armazena o índice do *namespace* ao qual pertence, nome do atributo, seu valor e o tamanho do seu valor.

```
struct RamFS_xattr {
        struct list_head list;
        struct RamFS_xattr_info info;
};
```

A estrutura acima, por sua vez, contém a `list_head` que conterá todos os atributos de um arquivo. Além disso, contém um campo que associa os dados do atributo a cada elemento da `list_head`.

As funções próprias de get e set do RamFS estão implementadas no arquivo `xattr.c` e o *namespace user* foi implementado no arquivo `xattr_user.c`. Neste último, destaca-se a criação do *handle* para esse *namespace*, no qual é indicado o prefixo do namespace e as suas funções de *get*, *set* e *list*.

4. Considerações Finais

Atributos estendidos permitem que aplicações criem soluções interessantes utilizando as características dos arquivos armazenados em um sistema de arquivos. O presente artigo apresentou a implementação do suporte de atributos estendidos no RamFS para o *kernel* 2.6.18.2. Foi implementado o *namespace user* e o mesmo funcionou corretamente. Foram feitos alguns testes de adição e remoção de atributos e o resultado obtido foi o esperado.

O suporte implementado baseou-se em parte da implementação do suporte a atributos estendidos no Ext3. No entanto, em função do RamFS não ser um sistema de arquivos persistente utilizou-se uma estrutura genérica de sistemas de arquivos, `struct inode` em `include/linux/fs.h`, para armazenar as informações referentes aos atributos estentidos.

Através da implementação realizada, aplicativos que se beneficiam do suporte de atributos estendidos podem ser usados no RamFS aproveitando todas as funcionalidades

disponíveis. O aplicativo de busca Beagle, por exemplo, poderia ser usado em diretórios temporários como o $home$ dos usuários de *livecds*.

A implementação dos demais *namespaces* no RamFS está entre os trabalhos futuros. Além disso, aplicações que utilizam as funcionalidades disponibilizadas pelos atributos estendidos podem ser exploradas no RamFS.

Agradecimentos

Agradecemos ao nosso amigo e colega Lucas Villa Real pela atenção recebida e pelas valiosas dicas que muito contribuiram para o desenvolvimento deste trabalho.

Referências

Beagle (2006a). Enabling Extended Attributes. Disponível em `http://beagle-project.org/Enabling\_Extended\_Attributes`. Acesso em novembro de 2006.

Beagle (2006b). Main Page. Disponível em `http://beagle-project.org/Main\_Page`. Acesso em novembro de 2006.

freedesktop.org (2006). CommonExtendedAttributes. Disponível em `http://www.freedesktop.org/wiki/CommonExtendedAttributes`. Acesso em novembro de 2006.

Grünbacher, A. (2006). POSIX Access Control Lists on Linux. Disponível em `http://www.suse.de/~agruen/acl/linux-acls/linux-acls-final.pdf`. Acesso em novembro de 2006.

Linux (2006). The Linux Kernel Archives. Disponível em `http://www.kernel.org/`. Acesso em novembro de 2006.

LinuxQuestions.org (2006). The Linux Kernel Archives. Disponível em `http://wiki.linuxquestions.org/wiki/SELinux/`. Acesso em novembro de 2006.

Ramfs (2007). Documentação do RamFS. Disponível em `http://lxr.linux.no/source/Documentation/filesystems/ramfs-rootfs-initramfs.txt`. Acesso em fevereiro de 2007.

SELinux (2006). Security-Enhanced Linux. Disponível em `http://www.nsa.gov/selinux/`. Acesso em novembro de 2006.

UML (2006). The User-mode Linux Kernel Home Page. Disponível em `http://user-mode-linux.sourceforge.net/`. Acesso em novembro de 2006.