Sistema de arquivos FUSE

Caique de Paula Figueiredo Coelho[1], **Lucas Gonzalez de Queiroz**[1]

[1]Faculdade de Computação – Universidade Federal de Mato Grosso Sul (UFMS)
Caixa Postal 549 – 79.070-900 – Campo Grande – MS – Brazil

caiquedpfc@gmail.com, lucasgq71@gmail.com

Abstract. *This paper presents the general concept of FUSE file system . Where are presented its Concepts , History , Use , Advantages and disadvantages for user , among other issues to be addressed*

Resumo. *Este trabalho apresenta o conceito geral do sistema de arquivos FUSE. Onde são apresentados seus conceitos, história, utilidade, vantagens e desvantagens para o usuário, dentre outros aspectos que serão abordados.*

1. Introdução

Antes de falarmos sobre o sistema de arquivos FUSE temos que falar sobre o espaço de usuário e sua usabilidade, vantagens, desvantagens e perigos. Ter um sistema de arquivos no espaço do usuário, ou seja, na área não privilegiada, onde os aplicativos de usuário são executados, pode ser um grande feito, uma vez que não afetam o kernel diretamente. Agora Vamos dar uma olhada em como e por que os sistemas de arquivos podem residir em userspace.

Há uma delimitação entre o que é chamado ”espaço de kernel”e ”espaço de usuário”. Espaço de kernel é de certa fomra “tudo o que está acontecendo dentro do código do kernel” ou no ”espaço”do código do kernel e dos seus recursos, tipicamente como um usuário privilegiado. Por outro lado o ”espaço de usuário” é “tudo o que está acontecendo fora do kernel”, onde qualquer usuário pode criar e executar um aplicativo e sistema de uso de recursos. Mas o delineamento entre esse dois espaços tem certas implicações.

Por exemplo, vamos supor que um bom programador tem uma ideia para algo que beneficiaria um número de pessoas. Conseguir que ”algo”esteja no kernel, se isso a ideia do bom programador, é um processo muito longo e difícil, e com razão. Se o código está no kernel, o grupo de desenvolvedores do kernel tem que se preocupar com a segurança do código, facilidade de manutenção, utilidade geral, as interações com outras partes do código, e até mesmo o estilo de codificação (com um estilo de codificação consistente faz com que seja mais fácil de ler partes do código para todos). Além disso, eles têm que pensar sobre se o código responde a estas preocupações e pertence ao kernel, porque a adição de código fará com que o kernel fique maior, isto é ”inchaço do kernel”, com uma série de efeitos colaterais indesejáveis..

Uma solução que até mesmo o próprio Linus sugeriu é mover o código para o espaço do usuário. Fazendo isso temos uma série de vantagens para o código, incluindo o fato de que um erro não é suscetível de derrubar todo o sistema como pode acontecer com um bug no kernel. Linus disse: ”Do ponto de vista técnico, eu acredito que o kernel será mais do mesmo, e que todas as coisas realmente interessantes estarão acontecendo no

espaço do usuário."Isso foi dito a mais de 9 anos atrás, no entanto, ele estava certo. O kernel é projetado para ser estável e fornecer acesso adequado aos recursos para aplicações do usuário e não ser o local do último sistema de arquivos que se teve uma ideia.

Os sistemas de arquivos no espaço do usuário tem algumas vantagens distintas. O primeiro é que eles não residem no kernel, por isso é bastante fácil de distribuir o seu código, ou seja, o código não tem que passar pelo processo extremamente rigoroso e às vezes agravante de testes, avaliação, justificação, alterações de código estilísticas, teste, avaliação, justificativa, etc. Esperemos que o desenvolvedor(s) do sistema de arquivos do espaço do usuário tenha prestado atenção à segurança, a facilidade de manutenção, as interações com outros aplicativos do usuário, etc., de modo que o código seja razoavelmente seguro para testes. Estar no espaço do usuário também significa que o código do sistema de arquivos pode ser testado mais cedo, uma vez que um bug geralmente só trava a aplicação e não causa um kernel panic. Os desenvolvedores não tem que esperar por algo completo e testado antes de liberar a um público mais amplo de teste e podem obter algum feedback mais imediato quanto à utilidade do sistema de arquivos.

A segunda vantagem é que você pode rapidamente atualizar o sistema de arquivos (sem espera por um par de revisões do kernel). O sistema de arquivos está em espaço de usuário para que você possa atualizar como faria com qualquer outro aplicativo de espaço do usuário, usando as ferramentas de pacote para a sua distribuição (por exemplo, yum ou apt-get).

Uma terceira vantagem é que, se o sistema de arquivo falhar por alguma razão, ele não necessariamente derruba o sistema operacional inteiro. Se o sistema de arquivo no kernel trava, há uma possibilidade de que o kernel também possa lançar uma exceção e acidente. Os desenvolvedores do kernel e do sistema de arquivos têm tido um enorme esforço para evitar que isso aconteça, mas o fato de um sistema de arquivos residir no espaço do kernel aumenta a possibilidade de um problema causar um kernel panic comparado a um aplicativo de espaço do usuário que normalmente cai sem causar um kernel panic. Se um sistema de arquivos do espaço do usuário falhar, você pode simplesmente matar todos os processos associados e, em seguida, remontar o sistema de arquivos.

O código de usuário não precisa ser executado com privilégio de raiz, se ele não precisa acessar dados ou dispositivos protegidos, ele pode implementar um sistema de arquivos virtual muito mais simples do que um driver de dispositivo tradicional. Devido a isso um grande número de sistemas de arquivos estão agora disponíveis para rodar em cima de FUSE. FUSE foi originalmente desenvolvido para suportar AVFS (A Virtual File System).

Assim o espaço de usuário tem algumas boas vantagens para os desenvolvedores. E o FUSE vem como ferramenta/mecanismo para escrever sistemas de arquivos no espaço do usuário.

Sistema de arquivos no Userspace (FUSE) é um subsistema que roda no espaço do kernel (ou seja, é controlado diretamente por ele e ocupa a mesma porção de memória). O FUSE é uma interface de software para sistemas operacionais de computadores Unix-like que permite que usuários não-privilegiados criarem os seus próprios sistemas de arquivos com programas que eles mesmos possam controlar, através de um módulo carregavel do kernel e uma API a ser utilizada pelo programa em modo usuário. Isto é conseguido

através da execução de código do sistema de arquivos no espaço do usuário enquanto o módulo FUSE fornece apenas uma ”ponte”para as interfaces do kernel reais.

FUSE está disponível para Linux, FreeBSD, OpenBSD, NetBSD (como puffs), OpenSolaris, Minix 3, Android e OS X. E adere a licença GPL sendo gratuito para download e um projeto open-source.

2. História do FUSE

Escrito na linguagem C, com o projeto iniciado em 2001. FUSE foi originalmente parte do AVFS (A Virtual Filesystem), uma implementacao de sistema de arquivos altamente influenciada pelo conceito do GNU Hurd. FUSE foi originalmente lancada sob os termos do GNU General Public License and the GNU Lesser General Public License, depois reimplementada como parte do FreeBSD base system e lancado sob os termos da BSD license. An ISC-licensed re-implementation by Sylvestre Gallon was released in March 2013, and incorporated into OpenBSD in June 2013. FUSE foi inserido no Linux kernel version 2.6.14.

3. Utilidade do FUSE

O FUSE (Filesystem in Userspace) é uma interface para programas de espaço de usuário para exportar um sistema de arquivos para o kernel Linux. O projeto FUSE consiste em dois componentes: o módulo do kernel fuse (mantido nos repositórios do kernel regular) e a biblioteca libfuse userspace (mantido neste repositório), libfuse fornece a implementação de referência para a comunicação com o módulo do kernel FUSE.

Um sistema de arquivos FUSE é tipicamente implementado como um aplicativo independente que liga com a biblioteca libfuse que por vez fornece funções para montar o sistema de arquivos, desmontá-lo, ler os pedidos do kernel e enviar respostas de volta. A biblioteca libfuse oferece duas APIs: uma de ”alto nível”, API síncrona, e uma de ”baixo nível”API assíncrona. Em ambos os casos, os pedidos recebidos a partir do kernel são passados para o programa principal usando retornos de chamada. Ao usar a API de alto nível, os retornos podem trabalhar com nomes de arquivos e caminhos, o processamento de uma solicitação termina quando a função de retorno de chamada retorna. Ao usar a API de baixo nível, os retornos de chamada devem trabalhar com inodes e as respostas devem ser enviadas explicitamente usando um conjunto separado de funções API.

Uma das dificuldades em escrever código no espaço do usuário está em interagir com o kernel. Em particular, um sistema de arquivos provavelmente vai ter que interagir com o kernel VFS (Virtual File System) em algum ponto para acessar o hardware. Então como é que um sistema de arquivos do espaço do usuário acessar o VFS.

Durante muito tempo este foi basicamente impossível. Qualquer código de sistema de arquivos teve que viver no espaço do kernel (isto é parte do kernel ou um módulo) e que era o fim da história. Mas o que você faria se, por exemplo, você quer criptografar/descriptografar os dados em um sistema de arquivos automaticamente? Ou se você deseja compactar/descompactar os dados em um sistema de arquivos automaticamente dando aos usuários a capacidade de ver os dados dentro de um arquivo tar sem untarring os dados? Ou talvez você quer apresentar os dados em uma tabela SQL como um diretório com arquivos associados? Todas estas coisas são grandes ideias, mas era im-

provável de fazê-lo no kernel. Idealmente, esses sistemas de arquivos devem viver no espaço do usuário, mas eles ainda precisam interagir com o kernel.

Diversos desenvolvedores decidiram que queriam desenvolver sistemas de arquivos ou sistemas de arquivos virtuais, como os já mencionados, no espaço do usuário, mas precisavam de um pouco de ”ajuda”do kernel para chegar lá. Então, eles criaram algo chamado FUSE (Sistema de Arquivo no espaço do usuário). O conceito é criar um módulo simples no kernel que interage com o kernel, em particular o VFS, em nome dos aplicativos de usuário não privilegiado e tem uma API que pode ser acessada a partir do espaço do usuário. Desta forma, seria possível que um usuário qualquer rode ou implemente o seu próprio sistema de arquivos, sem necessitar permissões de administrador ou quaisquer modificações no kernel do sistema. O FUSE faz a ligação entre o programa no modo usuário e a interface de sistema de arquivos do kernel. Para tal, ele e composto por um módulo carregável do kernel, responsável por repassar as requisições pela interface do sistema de arquivos, e uma API, que é utilizada pelo programa no modo usuário. O FUSE é constituído por um módulo localizado no kernel do SO (fuse.ko), uma biblioteca no espaço do usuário (libfuse) e um utilitário de montagem (fusermount).

O Módulo do kernel é o responsável pela conexão ao sistema de arquivos com o VFS, localizado no núcleo do SO. A libfuse é uma biblioteca no espaço do usuário responsável pela comunicação com o módulo do kernel. Enquanto o fusermount, por sua vez, é necessário devido ao fato de que a montagem de sistemas de arquivos, em ambientes Linux, é uma operação restrita aos usuários com privilégios de administração, logo este módulo é o responsável por assegurar que usuários, mesmo sem privilégios, poderão montar sistemas baseados em FUSE. Figura 1 abaixo é o exemplo clássico de como isso funciona.

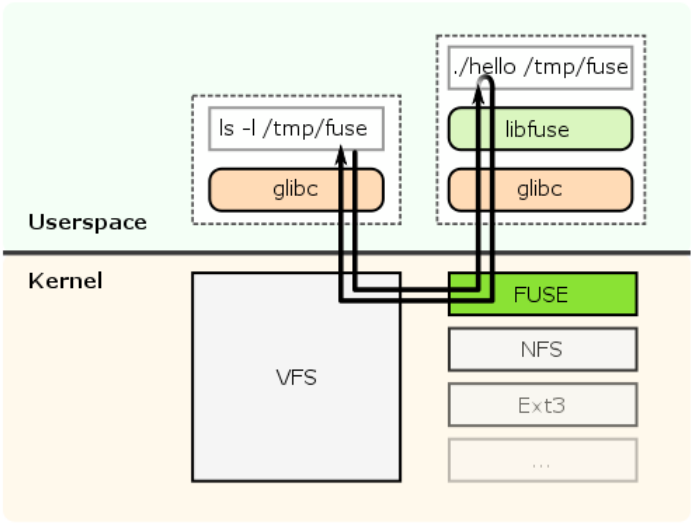

Figura 1. Um diagrama de fluxograma ilustrando como FUSE funciona (Wikipedia)

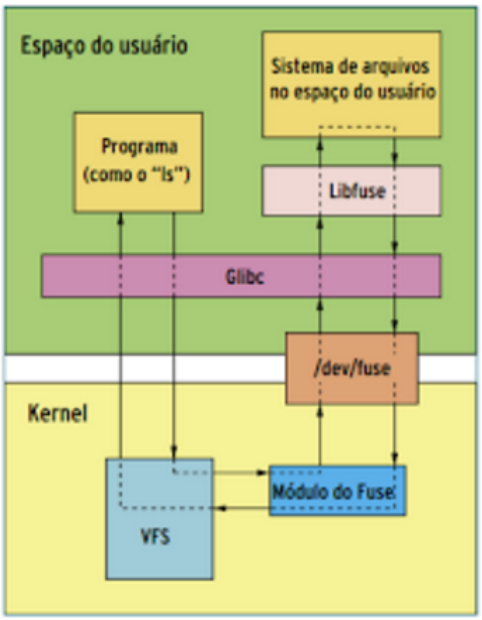

Figura 2. Um diagrama de fluxograma ilustrando como FUSE funciona de maneira geral

4. Exemplos de uso do FUSE

There are many examples of file systems that use FUSE. Sometimes FUSE is used for prototyping or testing file systems or it is used as the file system itself. It is beyond the scope of this article to list all of them or even a good chunk of them, but some that you might recognize (or might not) include:

Keybase filesystem (KBFS): A distributed filesystem with end-to-end encryption and a global namespace that uses FUSE to create cryptographically secure file mounts.

Wuala: A multi-platform, Java-based fully OS integrated distributed file system. Using FUSE, MacFUSE and Callback File System respectively for file system integration, in addition to a Java-based app accessible from any Java-enabled web browser.

WebDrive: A commercial filesystem implementing WebDAV, SFTP, FTP, FTPS and Amazon S3.

Transmit: A commercial FTP client that also adds the ability to mount WebDAV, SFTP, FTP and Amazon S3 servers as disks in Finder, via MacFUSE.

ExpanDrive: A commercial filesystem implementing SFTP/FTP/S3/Swift using FUSE.

GlusterFS: Clustered Distributed Filesystem having ability to scale up to several petabytes.

SSHFS: Provides access to a remote filesystem through SSH FTPFS.

GmailFS: Filesystem which stores data as mail in Gmail.

GVfs: The virtual filesystem for the GNOME desktop.

5. Vantagens e Desvantagens do FUSE

5.1. Vantagens

- Aplicações com capacidade de suportar dados em diversas formas de arquivos comprimidos, locais ou remotos, ou em dispositivos que não estão diretamente acessíveis por meios normais, sem modificar as aplicações;
- Muitas linguagens de programação;
- Suporte para Linux, OS / X, FreeBSD, NetBSD, OpenSolaris e Hurd;
- Fornece um método seguro para usuários não privilegiados que criam e montam suas próprias implementações de sistemas de arquivos;
- Portabilidade entre diversos sistemas operacionais;
- Utilização por usuários sem privilégios de administrador;
- Abstração das operações de baixo nível.

5.2. Desvantagens

- Não é suportado pelo Red Hat Enterprise Linux;
- Um número menor de sistemas de arquivos baseados em FUSE estão disponíveis ainda para MacOSX;
- Perda de desempenho: trocas de contexto que implica em aplicações mais lentas do que apenas no Kernel;
- Conteúdos do FUSE não são "cacheáveis" em geral.

6. Conclusão

O sistema de arquivos FUSE, apesar de ter sido criado há um bom tempo, permanece sendo importante para diversos sistemas e também para diversas empresas, desde das pequenas as grandes, assim, sua longevidade será alta pois também consegue incorporar diversos outros sistemas de arquivos.

Referências

Cornell, B., Dinda, P. A., and Bustamante, F. E. (2004). Wayback: A user-level versioning file system for linux. In *Proceedings of Usenix Annual Technical Conference, FREENIX Track*, pages 19–28.

Henk, C. and Szeredi, M. (2012). Fuse: Filesystem in userspace. *Online at http://sourceforge. net/projects/fuse*, 92.

Patil, S. and Gibson, G. A. (2011). Scale and concurrency of giga+: File system directories with millions of files. In *FAST*, volume 11, pages 13–13.

Szeredi, M. Fuse: Filesystem in userspace. 2005. *URL http://fuse. sourceforge. net*.

Szeredi, M. et al. (2010). Fuse: Filesystem in userspace. *Accessed on*.