Aula 10

10 - Sistema de arquivos no Linux

O arquivo é definido como uma estrutura que organiza um conjunto de informações ou dados utilizados nos sistemas e aplicativos. É uma abstração importante quando se analisa os sistemas computacionais. Através dos arquivos e suas estruturas, torna-se possível a um programador, armazenar uma grande quantidade de dados em um espaço confiável, permitindo seu processamento e manipulação posteriores.

O sistema de arquivos é a parte do software que organiza, gerencia e mantém as estruturas e hierarquias de armazenamento de dados e por isso ele é um ponto importante durante a construção de um sistema operacional. No Linux, existe uma camada de kernel que disponibiliza em um nível elevado, as funções e acessibilidades para que um programador desenvolva um sistema de arquivos sem se preocupar com a interface do nível de usuário. Esta camada de software é o *Virtual File System* (VFS).

No VFS do Linux é possível suportar sistemas de diversos tipos, inclusive sistemas de armazenamento de massa (discos, CDs, etcs), sistemas de rede (NFS, smb, etc) e sistemas virtuais como o *proc*, *ramfs* e *dev*.

Para permitir a capacidade de utilizar todos esses tipos de sistemas de arquivos, o VFS mantém uma lista de chamadas e de objetos que define quais as operações e estruturas um sistema de arquivos Linux deve seguir. Para essas operações, seria possível citar as chamadas *read, write, ioctl, mmap* entre outras. Já para os objetos, os principais são: arquivos, super blocos, i-nodes e entradas de diretórios (dentry).

Com a API do VFS, é possível manipular diversos sistemas de arquivos, incluindo a operação de sistemas de arquivos de terceiros como por exemplo o JFS, FAT32, NTFS, etc.

Atualmente, a maioria dos sistemas de arquivos mais eficientes trabalham com *metadados*, que no caso do Linux, são representados por nós índices ou *i-nodes.* Uma forma eficiente do uso dos metadados compreende a aplicação do sistema de jornais que pode utilizá-los para manter um registro das alterações a serem feitas em disco.

Um alteração substancial é que os sistemas de arquivos do VFS passaram a trabalhar com cache único para endereçamento da memória e do disco. Antes da versão 2.4 do Linux, havia um cache para páginas da memória (page cache) e outro para os índices, ou os arquivos do disco (buffer cache).

10.1 Sistema de arquivos: configuração

Quando se fala em sistemas de arquivos, tem-se a ideia da organização dos arquivos e diretórios de dados. Mas não se pode esquecer que o sistema de arquivos dos Unix e Linux também manipulam arquivos que representam dispositivos, processos ou canais de comunicação entre processos.

Para cada partição criada, o sistema Linux deverá manter uma seção de superbloco para as informações sobre os arquivos e espaços. Como existe uma camada de VFS, estes sistemas de arquivos precisam implementar as operações e objetos requiridos.

A tabela de sistema de arquivos ativos em um sistema Linux está contida no arquivo de configuração */etc/fstab*. Nessa tabela, cada linha define a configuração do sistema de arquivos para cada volume existente.

Essa tabela é chamada no processo de inicialização para ativar as partições, uma por vez. A sintaxe das linhas desse arquivo deve respeitar a seguinte regra:

 # <file system> <mount point> <type> <options> <dump> <pass>

Onde cada um deles pode ser sucintamente definido como:

<file system>: em geral, refere-se ao dispositivo ou local onde o sistema será fisicamente mapeado
<mount point>: local (diretório) em que o sistema de arquivos terá seu ponto inicial. Por exemplo, o sistema raiz tem seu *mount point* como "/".
<type>: caracterização do sistema de arquivos. O kernel precisa suportar o sistema de arquivo a ser carregado. Pode receber opções como: ext3, ext4, fat32, iso9660,nfs, etc.
<options>: opções para o sistema montado. O manual do comando mount pode exibir todas as opções disponíveis. Mas as principais podem ser - *noauto* (não montar automaticamente), *user* (permitir ao usuário montar), ro (read only) e rw (read and write) entre outros.
<dump>: quando ligado (1), determina que o sistema de arquivos precisa ser copiado antes de ser montado. Caso contrário, não.
<pass>: Quando desligado (0), o sistema de arquivos não precisa sofrer teste de consistências (fsck). Quando ligado, recebe um número de 1 ou maior para indicar em qual sequência o sistema deve ser checado. Sempre indicar o sistema raiz como o primeiro, ou seja, com valor 1.
O arquivo fstab é lido em sequência durante a inicialização, por isso também é aconselhável que o sistema raiz esteja a frente de outros sistemas de arquivos derivados dele.
Alternativamente, é possível fazer a montagem manual a qualquer momento de qualquer sistema de arquivos suportado pelo sistema. O comando *mount* se incumbe disso:

#> mount -o <opções> -t <tipo> <dispositivo> <ponto de montagem>

Dessa forma, fazer o carregamento de um *pendrive*, reconhecido como /dev/sdb1, seria viável através do comando.
#> mount -t msdos /dev/sdb1 /mnt/pendrive

Ou ainda, para montar uma pasta *tmp* via *nfs* (network file system) do computador remoto:.
#> mount -t nfs remoto:/tmp /mnt/computador_remoto_temp

Para desfazer qualquer sistema montado, utiliza-se o comando *umount*:
#> umount /mnt/pendrive
#> umount /dev/sdb1

Note que é possível manter linhas no fstab que conduzam um montagem automática de dispositivos que por ventura sejam utilizados com bastante frequência, isto é, CDROM, pendrive, discos flexíveis, entre outros. Isso é possível através da adição da opção *auto* na respectiva entrada do dispositivo em questão. Entretanto, atualmente já existem outros subsistemas que realizam a leitura e o tratamento de eventos de hardware automaticamente, permitindo a montagem automática de dispositivos. É o caso dos *daemons HAL* e *autofs*. Ambos servem para tratarem dispositivos de maneira dinâmica e podem montar e desmontar em nível de usuário os pontos de montagem reconhecidos em tempo de execução.

10.2 Comandos para a manipulação de sistemas de arquivos

Algumas ferramentas importantes e muito utilizadas quanto lidamos com sistemas de arquivos estão listadas a seguir:

- fdisk: Serve para formatar e alterar as características de uma partição.
#> fdisk -l
Visualiza as partiçoes encontradas em seus discos reconhecidos
#> fdisk /dev/sda
Aciona o aplicativo fdisk para manipuar o disco /dev/sda e suas partições

- du e df: vizualiza o volume resultante de um sistema de arquivos ou conjunto de arquivos.
#> df -h
Visualiza todos sistemas de arquivos montados com seus espaços utilizados e livres

#> du -hs /tmp
Visualiza o conteúdo total dentro da pasta /tmp

-mknod: utilizado para criar arquivos de dispositivos.
#> mknod -m0600 /home/meu_disco b 2 0

-**fsck:** utilizado para fazer uma varredura e consistência do sistema de arquivos. Refaz a tabela de arquivos, devendo ser executado sem que a partição em questão esteja montada.
#> fsck /dev/sdc1

-**mkfs**: serve para formatar um sistema de arquivos, criando sua tabela organizacional em uma partição.
#> mkfs -t ext3 /dev/hda2

10.3 Segurança: controle e acesso

Como também, já exaustivamente sabemos, o Linux e todos sistemas operacionais contemporâneos possuem sistemas de arquivos com níveis de segurança e controles de acesso. Cada arquivo tem uma série de atributos que determinam suas propriedades, como tamanho, data de criação, proprietário e até o nível de manipulação que os usuários podem ter sobre tal arquivo.

Uma listagem completa para um arquivo em um sistema Linux seria exibida como segue:

-rw-r—r-- 1 root root 23 Aug 22 2009 20:12 texto1.txt

O primeiro caractere, é uma letra que representa o tipo do arquivo: - (regular), l (link), d (diretório), b (dispositivo de blocos), c (dispositivo de caracteres), etc. O primeiro número mais a esquerda representa a quantidade de subdiretórios que um arquivo diretório possui. A data de alteração e o tamanho do arquivo vem logo anteriormente ao nome completo do arquivo.

As propriedades de permissão deste arquivo indicam que o dono dele é usuário *root*. A terceira coluna nos mostra isso. Já a quarta coluna indica também a palavra *root*. Esta, agora, nos informa que o arquivo é de propriedade do grupo de nome *root*.

Para todos os arquivos, sem exceção, devemos ter um usuário e um grupo proprietário. Para que seja possível alterar esses atributos é necessário utilizar o comando *chown*.

Ex: *#>chown aluno.root texto1.txt* (alterando o dono para aluno e o grupo dono para root de texto1.txt)
#>chown user.guest texto1.txt (alterando o dono para user e o grupo para guest de texto1.txt)

Agora devemos olhar para os atributos de permissão do arquivo. Esses são descritos por três letras (r, w e x). Sendo elas:

- r : determina a permissão de leitura e exibição de um arquivo (read).
- w: determina a permissão de escrita de um arquivo (write).
- x: determina a permissão de execução de um arquivo (execute)

Cada uma das letras acima é acionada ou desligada para um arquivo através de bits relacionados. Ou seja, se o bit para r estiver em 0 significa que o arquivo não terá permissão de leitura habilitada. Se um arquivo tiver todos os três bits ativados, podemos representar por rwx ou 111, o que em base decimal seria equivalente ao número 7.

Entretanto, pode-se notar que existem três blocos de bits controladores de permissão. Como visto no arquivo *texto1.txt*. Este possui rw- r—r--. Isto ocorre porque cada um dos blocos de três bits determina a permissão para um nível de de usuários daquele arquivo.

O primeiro bloco indica as permissões para o proprietário do arquivo, neste exemplo o uid de root.

O segundo bloco determina as permissões para o grupo proprietário deste arquivo, neste caso também o gid de root.

Já o terceiro bloco determina as permissões de todos os outros usuários do sistema, que não fazem parte do grupo proprietário e nem possuem o id do usuário proprietário. Resumidamente, o que temos é:

rwx	*rwx*	*rwx*
usuário	*grupo*	*outros*

Para que seja possível alterar as permissões de um arquivo é necessário utilizar o comando *chmod*. Para este comando é possível alterar as permissões através dos valores em decimal referentes aos três blocos de permissão. Ou seja, definir o decimal correspondente ao primeiro bloco, o decimal correspondente ao segundo bloco e o decimal correspondente ao terceiro.

Ex:
#>*chmod 757 texto1.txt* (com isso, o arquivo ficaria com os atributos rwx r-x rwx)
#>*chmod 646 texto1.txt* (com isso, o arquivo ficaria com os atributos rw- r—rw--)
#>*chmod 000 text1.txt* (com isso, o arquivo ficaria com os atributos --- --- ---)

Outra maneira, menos elegante, é utilizar pontualmente a alteração da permissão que se deseja atribuir.
Ex:
#>*chmod u+x texto1.txt* (adicionar execução ao dono para texto1.txt)
#>*chmod u+rw,g-w texto1.txt* (adicionar leitura e escrita ao dono e retirar escrita do grupo dono).
#>*chmod o=r texto1.txt* (definir que todos possam ler o arquivo, sobrescrevendo a permissão para outros anteriormente definida)

Para finalizar, deve-se lembrar que ainda existe um quarto bloco de 3 bits que e não é mandatório ser informado no momento da alteração das permissões. Esse quarto bloco de bits é conhecido como bloco de bits de atributos especiais. Sendo eles:
- setuid : se este bit estiver ligado o arquivo é executado como se fosse pelo dono do arquivo.
- setgid : se este bit estiver ligado ele é executado pelo grupo dono deste arquivo.
- sticky : se este bit estiver ligado somente o dono do arquivo poderá removê-lo.

Portanto a representação completa da lista de bits de atributos fica assim:

su sg st	*rwx*	*rwx*	*rwx*
especiais	*usuário*	*grupo*	*outros*

E para se alterar esses bits, basta adicionar um decimal à esquerda do número tornando-o agora um número com 4 algarismos.
Ex:
#>*chmod 1777 texto1.txt* (permite rwx para todos e ainda liga o bit sticky).
#>*chmod 2777 texto1.txt* (permite rwx para todos e ainda liga o bit sgid).
#>*chmod 4777 texto1.txt* (permite rwx para todos e ainda liga o bit suid).
#>*chmod 3777 texto1.txt* (permite rwx para todos e ainda liga o bit sguid e o bit sticky).

Com isso, é possível, minimante trabalhar com os acessos e liberações de arquivos. Mas de qualquer forma é importante saber que existem outras ferramentas e técnicas para proteção e controle de acesso aos dados no Linux, como por exemplo um sistema de arquivos com ACLs (access control list) ou com atributos adicionais (*lsattr e chattr*).

Cabe questionar, quando um arquivo é criado por um usuário no sistema, qual atributo de permissão o mesmo recebe? A resposta está no comando *umask,* que é executado durante a leitura do perfil em */etc/profile*

ou do bashrc em */etc/bash.bashrc*. Nesta máscara, por padrão, o sistema nem considera o bit executável, pois este deve ser explicitamente citado, não sendo interessante criar arquivos executáveis o tempo todo.

O padrão, em geral, do umask é mantê-lo com o valor 022, que indica que todo novo arquivo terá seus bits de permissão com os valores 644 (rw r r). Para alterar isso, basta executar *umask* na seção corrente ou alterar o seu valor dentro dos arquivos de *profile* local ou global.

10.4 Exercícios

1)Dado os seguintes arquivos e diretórios faça:

drwxrwxrwt 8 root root 4096 Mar 17 05:15 .

drwxr-xr-x 28 root root 4096 Sep 10 2007 ..

drwxrwxrwt 2 root root 4096 Sep 10 2007 .font-unix

drwxrwxrwt 2 root root 4096 Sep 26 13:28 .ICE-unix

drwx------ 2 robin ngi 4096 Aug 21 2007 keyring-XxjCy4

drwx------ 2 root root 16384 Jun 22 2007 lost+found

-rwxr-xr-x 1 robin ngi 0 Sep 10 2007 mapping-robin

drwx------ 2 robin ngi 4096 Sep 26 13:28 orbit-robin

drwxrwxrwt 2 root ngi 4096 Sep 26 13:28 .X11-unix

a) Altere o dono e grupo dono de orbit-robin para batman e users respectivamente

b) Altere o dono para daemon e o grupo para admin de .X11-unix

c) Altere o dono para root de mapping-robin

d) Altere o grupo dono para users de .ICE-unix

2) Ainda considerando os arquivos anteriores altere as permissões conforme os itens a seguir:

a) para o diretório lost+found usuário possa ler e executar,grupo possa ler e outros possam executar.

b) para o direrório orbit-robin o usuário possa executar,escrever, grupo possa escrever e ler e outros possam ler e executar.

c) para que o .X11-unix seja somente leitura para usuário, grupo e outros.

d) para que .font-unix tenha permissão de leitura escrita e execução para usuário, grupo tenha somente escrita e outros tenham leitura e exeção.

e) para que .font-unix tenha somente permissão de leitura para o grupo.

f) para que usuário denha permissão execução, grupo tenha permissão de execução e outros tenham permissão de execução em .X11-unix.

g) para que somente usuário tenha acesso a leitura e escrita de mapping-robin.

h) para que o diretório lost-found tenha permissão de leitura e escrita para usuário, grupo e outros de forma recursivamente.

i) para que o diretório keying-XxjCy4 tenha permissão de leitura e execução para usuário, grupo tenha leitura e outros tenham leitura e execução recursivamente.

3) Mantenha automática a montagem de um pendrive em seu sistema. Qual arquivo foi alterado? Como ficam as linhas de configuração?

4)Quais são os atributos especiais de execução (suid e sgid). Explique sua finalidade.

5) Explique o que fazem os comandos abaixo:

a) mount -t nfs 192.168.0.1:/documentos /mnt/docs_servidor

b) mount

c) df -h

d) du -hs /home

e) mkfs -t vfat /dev/fd0

6) Através de um sessão de usuário administrador realize e informe quais seriam os comandos responsáveis em:

a) criar um arquivo regular qualquer

b) criar um arquivo de link simbólico qualquer

c) criar um arquivo diretório qualquer

d) criar um arquivo de dispositivo de caracteres

e) criar um arquivo de pipe nomeado

10.5 Bibliografia

Nemeth, E., Snyder, G., Hein, T. R. .Manual Completo do Linux, 2 edição, Makron Books, 2004 São Paulo.