Dæmonio Labs

Manipulando Arquivos Descritores No Shell

POSTADO EM FEVEREIRO 24, 2012 POR DAEMONIO
2

Introdução

Além dos usos de pipes, redirecionamentos e substituições de subshell podemos também ler e escrever em arquivos usando arquivos descritores, do mesmo modo que fazemos em outras linguagens de programação.

Nesse post iremos aprender a manipular arquivos descritores para se realizar I/O em arquivos no shell.

Primeiramente: o que são arquivos descritores?

Para se realizar operações de entrada e saída em arquivos, o sistema operacional fornece uma chamada do sistema que abre tal arquivo e retorna para a aplicação do usuário um número inteiro que representa esse arquivo. Você provavelmente lida com nomes dos arquivos para os identificar, mas os programas internamente lidam com números inteiros e são esses números que recebem o nome de arquivos descritores. Após o arquivo ser aberto, o arquivo descritor dele será responsável pelas operações de entrada e saída.

Para mais detalhes, veja em [1].

Descritores padrões

Para todo processo criado, o shell reserva 3 arquivos descritores padrões que servem para realizar operações básicas como leitura do teclado e saída na tela. Esses descritores são:

ID padronizado	Nome	Descricao
0	stdin	associado a leitura do teclado
1	stdout	associado a saida normal na tela
2	stderr	associado a saida de erro na tela

O ID padronizado representa o número usual associado àquele descritor, mas esse número pode ser mudado, como veremos nesse post.

O comando interno do bash: exec

O bash tem um comando interno (built-in) chamado exec que pode ser usado para operações de arquivos descritores: criar, fechar, copiar. A sintaxe básica desse comando é:

```
$ exec n "redirecionamento" m/arquivo
```

Onde:

n e m	arquivos descritores
"redirecionamento"	<, >, >> e <>
arquivo	nome de um arquivo no sistema

Vamos aproveitar e fazer usos dessas sintaxes:

```
#!/bin/bash
exec 6</etc/passwd
cat <&6
```

O conteúdo do arquivo /etc/passwd aparecerá na tela, de modo semelhante se usássemos o comando cat com o nome do arquivo como parâmetro.

Na linha do exec, associamos o arquivo /etc/passwd ao inteiro 6 e com isso, de agora em

diante, qualquer operação sobre o descritor 6 será feito automaticamente no arquivo /etc/passwd. Como usamos o símbolo '<', somente a operação de leitura poderá ser usada.

Usualmente, quando redirecionamos arquivos pela entrada padrão sem usar pipes, usamos "cat < /etc/passwd", mas nesse caso, lidando com descritores, devemos adicionar um '&' no fim do '<' para o shell entender que o que vem à frente é um arquivo descritor e não um nome de arquivo.

Veja esse outro exemplo:

```
#!/bin/bash

# Associa stdout ao descritor 6
exec 6<&1

# Associa stdout ao arquivo
# /tmp/filesaida
exec 1>/tmp/filesaida

A=0;
while [ $A -lt 10 ]; do echo $A; let A++; done

# Recupera stdout para o
# comando cat
exec 1<&6

cat /tmp/filesaida ; rm -f /tmp/filesaida
```

No arquivo /tmp/filesaida estará a saída do loop while, ou seja, os números de 0 a 9. Dentro do loop foi utilizada uma simples chamada ao comando echo sem usar nenhum redirecionamento para arquivos, mas mesmo assim, a saída foi para um arquivo no disco.

Essa mágica acontece porque associamos a saída padrão (stdout) a um arquivo através do exec. Primeiramente associamos stdout ao descritor de número 6. O número 6 foi só como exemplo, poderíamos escolher qualquer número seguramente na faixa 3-9 [2]. Feito isso, stdout agora pode ser acessado tanto pelo descritor 1 quando pelo 6, mas logo em seguida, fechamos o descritor 1 e o associamos ao arquivo /tmp/filesaida. De agora em diante, todo comando que escrever na tela, ou seja, no descritor 1, automaticamente estará escrevendo no arquivo /tmp/filesaida. No fim do script, recuperamos o descritor 1 em stdout para que o comando cat escreva sua saída na tela.

Por último, veremos um exemplo usando coprocessos [3]:

```
#!/bin/bash

coproc bc

# Salva stdout
exec 6<&1

# Associa stdin e stdout
# a saida e entrada do bc
# respectivamente.
exec 0<&${COPROC[0]}
exec 1>&${COPROC[1]}

# Calcula n^2
for A in {10..20}; do
    echo "$A * $A"
    read RESPOSTA
    echo "$A * $A = $RESPOSTA" >&6
done

# Fecha o descritor 6
exec 6<&-
```

A saída será:

```
10 * 10 = 100
11 * 11 = 121
12 * 12 = 144
13 * 13 = 169
14 * 14 = 196
15 * 15 = 225
16 * 16 = 256
17 * 17 = 289
18 * 18 = 324
19 * 19 = 361
20 * 20 = 400
```

Fechando descritores

Na última linha do script acima está um modo de fechar arquivos descritores. A sintaxe é:

```
 $ exec n<&-
```

Onde n representa o arquivo descritor a ser fechado.

Utilidades de usar arquivos descritores?

Os exemplos apresentados até então (exceto o último) poderiam ser escritos sem a utilização de arquivos descritores, o que gera dúvidas onde eles devem ser utilizados. Bem, arquivos descritores na maioria das vezes não são exclusivos, pois sempre pode-se aparecer uma maneira de escrever o programa sem utilizá-los. Sua utilidade surge como alternativa para facilitar a programação, como por exemplo, quando estamos lidando com muitos arquivos ao mesmo tempo e o uso de arquivos descritores facilita a identificação de cada arquivo isoladamente.

Pesquisando um pouco encontrei alguns outros usos, que serão listados abaixo:

1) Evitar subshells

A dupla "cat arquivo | while" esconde em sua simplicidade um dos erros mais devastos da programação em shell script, que consiste em utilizar variáveis setadas dentro do while após a execução do loop. Como sabemos, as variáveis usadas dentro do while são exclusivas dele e elas não existirão após o término do loop. Isso ocorre porque o pipe cria dois processos sendo cada um com seus próprios dados que não serão compartilhados com o processo pai.

Usando exec e descritores de arquivos podemos evitar subshells, como visto em [5]:

```
#!/bin/bash

# Contem a maior media
MAIOR_MEDIA=0

# $ cat medias.txt
#   45 60 70
#   30 80 39
#   ....
exec 3< medias.txt
while read linha <&3
do
    TEMP=$(echo "$linha" | sed 's/ /+/g; s/.*/(&)\/3/' | bc)
    [ $TEMP -gt $MAIOR_MEDIA ] && MAIOR_MEDIA=$TEMP
done

# Fecha o arquivo medias.txt
exec 3<&-
echo "[+] A maior media e': $MAIOR_MEDIA"
```

2) Leitura de múltiplos arquivos

Usando arquivos descritores, temos a possibilidade de uma leitura, linha por linha, de vários

arquivos. Veja mais detalhes em [2].

3) Aumento de velocidade

De acordo com a referência [4] (muito boa por sinal, recomendo a leitura) alguns scripts podem ser agilizados em até 5x usando o exec para substituir sintaxes conhecidas do shell.

Para testar, vamos executar o script a seguir por duas vezes, uma vez usando a função A1() e em seguida A2().

```
#!/bin/bash
# codigo retirado da referencia [4]
# OBS: Na execucao de A1 os execs
# devem ser retirados.

A1(){
        echo "\c" >> /tmp/tt$$
}

A2(){
        echo "\c"
}

exec 3<&1
exec 1>>/tmp/tt$$

for i in 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 0
do
for j in 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 0
do
for k in 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 0
do
for l in 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 0
do
        A2
done
done
done
done

exec 1<&3

# end of script

rm /tmp/tt$$
```

Os tempos em minha máquina foram:

```
$ time ./script-a1.sh

real    0m4.847s
user    0m3.160s
sys     0m1.661s

$ time ./script-a2.sh

real    0m2.661s
user    0m2.032s
sys     0m0.625s
```

Vemos claramente que, nesse caso em específico, o uso dos execs em relação ao redirecionamento duplo obteve um tempo de execução menor.

Conclusão

O uso de arquivos descritores fornecem alternativas de programação para problemas usuais, por exemplo no tratamento de vários arquivos. Em alguns casos, seus usos podem acelerar a execução dos scripts, trazendo vantagem para aqueles que realizam I/O em grandes quantidades de dados.

Referências

[1] Descritor de arquivo (Acessado em: Fevereiro/2012)
http://pt.wikipedia.org/wiki/Descritor_de_arquivo (http://pt.wikipedia.org/wiki/Descritor_de_arquivo)

[2] 16.1. Using exec from Advanced Bash-Scripting Guide, cap. 16 (Acessado em: Fevereiro/2012)
http://www.linuxtopia.org/online_books/advanced_bash_scripting_guide/x13082.html (http://www.linuxtopia.org/online_books/advanced_bash_scripting_guide/x13082.html)

[3] Entendendo Coprocessos (coproc) Do Bash by Daemonio (Acessado em: Fevereiro/2012)
https://daemoniolabs.wordpress.com/2011/07/19/entendendo-coprocessos-coproc-do-bash/ (https://daemoniolabs.wordpress.com/2011/07/19/entendendo-coprocessos-coproc-do-bash/)

[4] David Butcher: Speeding Up Your UNIX Shell Scripts, by David Butcher (Acessado em: Fevereiro/2012)
http://www.los-gatos.ca.us/davidbu/faster_sh.html (http://www.los-gatos.ca.us/davidbu/faster_sh.html)

[5] Reading Multiple Files with Bash By Mitch Frazier (Acessado em: Fevereiro/2012)
http://www.linuxtopia.org/online_books/advanced_bash_scripting_guide/x13082.html
(http://www.linuxtopia.org/online_books/advanced_bash_scripting_guide/x13082.html)

POSTADO EM COMANDOS, LINUX, PROGRAMAÇÃO, SCRIPTS, SHELL SCRIPT | MARCADO ARQUIVOS DESCRITORES EXEC BASH SHELL SCRIPT, COMO UTILIZAR COMANDO EXEC, EXEC SHELL SCRIPT BASH

2 pensamentos sobre “Manipulando Arquivos Descritores No Shell”

1. Pingback: Sockets No Bash | Daemonio Labs

2. *Daemonio*
   JANEIRO 12, 2013 ÀS 2:07 PM
   Post editado em: 12/01/2013

   Responder