Ponha ordem em seu sistema usando a...

Linguagem de Sinais

Um sistema Unix executa até 30 processos simultâneos. Eles freqüentemente precisam se comunicar, e os sinais fornecem o melhor método conhecido para isso. POR MARC ANDRÉ SELIG

Um processo é iniciado, completa seu serviço, coloca tudo em ordem e termina. Se o trabalho é muito complexo, o processo pode levar um certo tempo – semanas, ou até mesmo meses, para completá-lo. Tome por exemplo um servidor Web, que geralmente residirá na memória até que uma versão atualizada do sistema seja lançada; o mesmo também se aplica a muitos outros "daemons". Isto não quer dizer que o ambiente para este tipo de software será sempre o mesmo. Após modificar um arquivo de configuração, o administrador precisa fazer que o daemon note que modificações foram efetuadas, o que envolve tipicamente alguma forma de comunicação entre processos.

Sinais

Sinais são uma das técnicas mais bem conhecidas para comunicação entre processos (IPC – inter-process communication). Cada vez que você digita o comando *kill* para encerrar um processo desnecessário, você está na verdade enviando um sinal a este processo (veja Figura 1). A Figura 2 mostra uma lista dos sinais definidos para *kill*. Esta lista, também contida no cabeçalho do arquivo */usr/include/bits/sgnum.h* vale para todos os programas.

Se você tentar encerrar um processo sem fornecer qualquer parâmetro adicional, o comando emitirá o sinal 15 (*TERM*, terminar). Entretanto, alguns processos simplesmente ignoram este sinal, ou já podem ter se encerrado. Neste caso, o sinal *TERM* não é usado. Para obter o resultado desejado, você precisa digitar *kill -9*, dessa forma emitindo um sinal *KILL*. A menos que você tenha escolhido um zombie ("zumbi", veja o quadro), ou um processo que está aguardando por uma operação de I/O, e que portanto reside no espaço do kernel, o processo é obrigado a se encerrar.

Encerrando Programas

Alguns outros sinais são muito importantes – ainda que não pareça tão óbvio à primeira vista. A maioria dos leitores já pressionou *[Ctrl]+[Z]* para interromper um programa, como um editor de textos. O shell reage mostrando o seguinte:

```
[2] + Stopped  vi ➥
myfile
```

Você pode reiniciar o editor digitando o comando *fg* (*foreground*, primeiro plano). O comando *jobs* lhe diz o estado de todos os processos controlados atualmente pelo shell:

```
[1] - Running  emacs &
[2] + Stopped  vi ➥
myfile
```

[Ctrl]+[Z] envia o sinal 19, STOP. O comando *fg* continua o processo interrompido, emitindo o sinal 18, *CONT* (continua). O sinal 18 também é usado no comando *bg*, que envia um processo para background (segundo plano), do mesmo jeito que adicionar um & à frente de

Figura 1: O usuário *mas* executa os comandos *ps* e *grep* para procurar pelos processos do Opera, e então executa o comando *kill PID*, para dizer a esses processos eles que devem terminar.

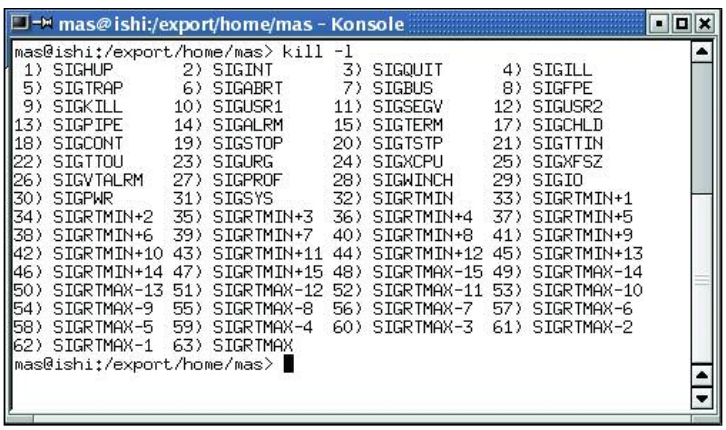

Figura 2: O Linux reconhece 64 sinais diferentes, os quais possuem um nome e um número. O freqüentemente usado *kill -9*, também pode ser expresso como: *kill -SIGKILL* ou *kill -KILL*.

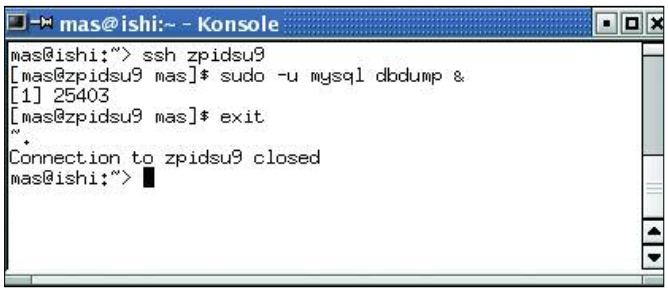

Figura 3a: O SSH aguarda pacientemente que os processos-filho terminem antes de fechar a conexão. Digitar ~ força o fechamento.

Figura 3b: Processos demorados em background precisam ser iniciados com o comando *nohup*, se quiser que eles sobrevivam ao comando ~.

um comando. Se você quiser parar temporariamente um processo em background (porque ele está consumindo muitos recursos, por exemplo), e continuá-lo mais tarde, pode usar os comandos *kill -19* e *kill -18*.

O exemplo acima demonstra uma notação simplificada para os números de processos, oferecida pelo bash como um atalho. Normalmente cada processo no Linux tem um PID (*Process ID* - identificador de processo), entre 1 e 32767, onde 1 é reservado para o init, o primeiro de todos os processos. Os primeiros 100 PIDs são usados pelos processos do próprio kernel. Os demais identificadores são usados pelos programas executados pelo usuário.

Digitar repetidamente *kill -18 21967*, sem errar, é cansativo. O Bash oferece duas possibilidades de simplificação. A primeira é que ele sabe os PIDs dos processos atuais e anteriores. Eles são marcados com sinais de mais e menos, respectivamente, na lista de processos. Se você utilizar o comando *fg*, para manipular um processo, mas não especificar um PID, o comando é aplicado ao processo atual. A segunda coisa que o Bash faz é fornecer atalhos para os nomes de processos, [1] e [2], neste caso. Utilize o símbolo de porcentagem antes do número para acessar estes atalhos. No exemplo anterior, o comando *fg %1* envia o Emacs para primeiro plano.

Além de *[Ctrl]+[Z]* (suspender), alguns shells também executam uma suspensão "atrasada" de um processo, *[Ctrl]+[Y]*. Enquanto o comando para suspender imediatamente um processo envia um sinal *STOP*, o *[Ctrl]+[Y]* espera até que o processo termine de ler

dados do terminal. Isto permite que um processo seja manipulado imediatamente após aceitar uma entrada.

Desconexão

Um outro sinal remonta à época dos terminais seriais. O sinal número 1 é chamado de *HUP* ou *hangup* (desconexão). Se você utiliza um modem para acessar um sistema Unix e a conexão é interrompida, o seu shell recebe um sinal HUP e pode se recompor. Por exemplo, um editor pode criar um backup e então encerrar sua execução.

Conexões via modem são raras atualmente, mas o sinal HUP mantém sua utilidade. Encerrar uma conexão SSH é um dos usos, e também uma "pega-dinha" muito comum.

Imagine o cenário da Figura 3a. O usuário *mas* acessa via SSH uma outra máquina para executar algumas tarefas de gerenciamento remoto e inicia um processo demorado. Como ele não quer esperar pelos resultados, adiciona o caractere & ao final da linha de comando para enviar automaticamente o processo para background (segundo plano) antes de encerrar a sessão. Infelizmente, isso parece não funcionar, e o ssh aparenta estar "travado". Para o usuário, parece não haver alternativa a não ser pressionar ~, o caractere de escape do SSH, para encerrar a conexão.

Zombies

Um processo que gera um outro é conhecido como um processo pai, e o relacionamento entre os dois é referido como pai/filho. O processo que chama é o pai, e o chamado é o filho. Se o processo filho termina antes do seu pai, o pai envia um sinal *CLD* ou *CHLD* (child). É esperado que o processo pai confirme a conclusão dos seus filhos. Se o pai falha neste procedimento, o filho é identificado na tabela de processos como um zombie (zumbi) [1]. Afinal, como o kernel vai saber se e quando o pai vai verificar o estado de saída do filho? A entrada do processo na tabela tem que ser mantida, para cobrir esta possibilidade.

Até lá, a maioria dos recursos de sistema já terão sido liberados. O processo filho original não existe mais, e o comando *kill -9* não terá efeito sobre ele. O processo "zumbi" só desaparece quando o processo pai termina ou quando procura saber o estado de saída do processo filho.

O que realmente aconteceu é que o SSH não travou, simplesmente notou uma tarefa em segundo plano. O SSH estava esperando que este processo terminasse para então encerrar a conexão. Quando a conexão é interrompida, um sinal HUP é enviado ao processo, encerrando-o.

A Figura 3b, mostra o jeito certo de fazer isto. O comando *nohup* protege um processo em segundo plano de um sinal *HUP*. O SSH nota isto e continua a esperar, mas desta vez a conexão pode ser encerrada sem afetar o processo em segundo plano. O utilitário *dbdump* continuará em execução, e apresentará seus resultados na próxima vez que o usuário *mas* se conectar ao sistema. Esta sintaxe do *dbdump* é mais elegante:

```
nohup sudo -u mysql dbdump ➥
</dev/null &
```

Isto significa que o processo não está conectado ao terminal e o SSH terminará após o usuário se desconectar do sistema, sem que ele tenha que digitar ~. para encerrar a conexão.

Modificando Configurações

Um uso muito comum do sinal HUP é para avisar a um daemon de que foram feitas modificações em seu arquivo de configuração.

O daemon syslog [2], é um exemplo típico. Normalmente ele registra apenas alertas e mensagens de erro, para poupar espaço no disco rígido e dores-de-cabeça do administrador. Mas após instalar um novo pacote de software, é necessário mais do que isso.

Se o software não funciona como esperado, o administrador pode querer o registro de mensagens adicionais e informações de debug para ajudar na solução do problema. Para isso, é necessário adicionar uma linha ao arquivo de configuração do syslog, */etc/syslog.conf*, para dizer ao daemon que estas informações devem ser registradas:

```
*.* -/var/log/everything
```

Você precisa enviar um sinal HUP para que o syslog reconheça a mudança. *ps -ef | grep syslog* mostra o PID, e o comando *kill -1 PID*, se encarrega de encerrar o daemon. Se você não tem tempo para localizar o PID, algumas distribuições Linux incluem o comando *killall*, que lhe permite enviar um sinal qualquer a todos os processos com o mesmo nome que você informar:

```
killall -1 syslogd
```

Este comando aceita uma série de parâmetros que podem ser consultados com o comando *man killall*

Sinais definidos pelo usuário

Alguns outros sinais, como 10 (USR1), e 12 (USR2), estão disponíveis para tarefas definidas pelo usuário. Por exemplo: se o arquivo de configuração do Apache for modificado, você pode forçar o daemon a reler o arquivo enviando um sinal *HUP*. Entretanto, isto significa que quaisquer processos filhos do Apache serão abandonados. Se, ao invés disso, você enviar um sinal USR1 ao processo principal do Apache, ele irá aguardar até que todos os trabalhos tenham terminado antes de interromper e reiniciar os processos-filhos.

O uso de sinais é amplamente difundido e parte essencial do cotidiano de todo administrador de sistemas. Mas restrições com o número de sinais limitado e o fato de que, por segurança, apenas o super-usuário pode enviar sinais aos processos que não lhe pertençam justificam a existência de mecanismos alternativos, que serão examinados em um futuro artigo na Linux Magazine. ■

SOBRE O AUTOR

Marc André Selig trabalha como cientista assistente na Universidade de Trier, na Alemanha, e como médico no hospital Schramberg. Quando encontra algum tempo livre, seu interesse atual é a criação de sistemas de base de dados baseados na Web em várias plataformas UNIX.

INFORMAÇÕES

[1] Artigo na Wikipedia sobre processos zumbi: *http://en.wikipedia.org/wiki/Zombie_process*

[2] Marc André Selig, "The System Logger", Linux Magazine, Edição 40, Março de 2004, página 64.