Capítulo 12 - Processos

12.1. Objetivo:

Entender como é feito o gerenciamento de processos no Linux.

12.2. Introdução

Antes de qualquer coisa é necessário ter claro o conceito correto de processo. Processo é uma tarefa em execução controlada pelo kernel ocupando uma área na memória (RAM), no qual so Linux usa o mecanismo de identificá-los por números. Ou seja, tudo que estiver rodando no sistema será um processo. É o processo que utiliza os recursos de um computador, como memória e processamento. Um simples comando é um processo, um browser aberto é um processo. O shell que você está usando é um processo, seu terminal é um processo.

Quem controla os processos no sistema é o Kernel, aliás essa é uma de suas tarefas. É responsabilidade do Kernel gerenciar os processos tentando otimizar a performance da CPU – Unidade Central de Processamento.

O processo init , que é pai de todos os outros processos.

O GNU/Linux usa um mecanismo de identificá-los por números, ou seja, cada processo tem um número diferente de identificação que chamamos de PID. Dessa forma pode-se manipulá-los e controlá-los. Controlar é poder matar , reiniciar. Na prática usa-se mais é para matar um processo.

Formas de visualização de processos:

 # ls /proc

Nesse diretório, você verá vários arquivos com informações gerais que o kernel nos fornece, por exemplo os arquivos: cpuinfo, ioports, meminfo, interrupts, mounts, swaps e muitos outros. Vale a pena ver o conteúdo desses arquivos para aprender mais sobre eles.

No /proc, você pode entrar em um número (diretório) que identifica um processo e listar o conteúdo dele, exemplo:

```
# cd /proc/3512
# cat cmdline
/usr/lib/iceweasel/firefox-bin-afirefox
```

Foi escolhido um número aqui aleatoriamente no sistema. Na seu sistema este número pode ser de outro processo e até mesmo não existir, portanto escolha um no seu /proc e explore. Dois arquivos interessantes de explorar nesses diretórios dos processos são:

```
cmdline
status
```

Para saber mais sobre o /proc:

```
# man proc
```

Para listar todos os processos:

```
# ps aux | more
```

Explicando o comando:

```
#ps aux
a = todos os processos
u = de todos os usuários
x = e até processos sem controle pelo terminal
```

Ou seja, todos os processos mesmo!

Controlado ou não pelos terminais significa que são processos internos, que não foram gerados por um usuário que esteja usando um determinado terminal.

É melhor usar o comando ps do que entrar no /proc e entrar em diretório por diretório.

O PID nada mais é do que Identificação do Processo. É através do PID que é posśivel manipular um processo. Ou seja, matar caso ele esteja atrapalhando o sistema! O PID seria a segunda coluna do resultado do comando ps. Vamos analisar as colunas do comando ps.

```
# ps aux | more
```

USER - Usuário responsável pelo processo.
PID - Número que identifica o processo, não se repete!
% CPU - Quanto de CPU ele usa!
% MEM - Quanto de MEM ele usa!
VSZ - Tamanho virtual do processo;
RSS - Indica a quantidade de memória usada (em KB);
TTY - Terminal que gerou o processo.
? - sem terminal
STAT - Estado do processo, sendo representado por uma letra.
R - executável;
D - em espera no disco;
S - Suspenso;
T - interrompido;
Z - Zumbi.
COMMAND - Nome do processo, ou seja, o comando em si.

Com a opção f em ps, é possível ver em forma de árvore, caso o processo tem processos filhos:

```
# ps faux | more
```

Uma alternativa ao ps faux é:

```
# ps aux | more
# pstree
```

Para monitorar a execução dos processos vendo uma tabela que se atualiza de tempos em tempos:

```
# top
```

Esse comando gera uma tabela dinâmica com informações sobre os processos, é possível matar um processo daqui mesmo. Para maiores informações sobre o comando top, digite o comando “h” (help) quando estiver nele, ou:

```
# man top
```

Para matar um processo, você tem duas formas:
Ou pelo PID (número do processo) (2ª coluna)
Ou pelo COMMAND (nome do processo) (última coluna)

Lembrando que você precisa ser o dono do processo ou o usuário root para destruí-lo. Então podemos visualizá-lo dessa maneira:

```
# ps aux | awk '{print $2,$11}'
```

Onde estará mostrando apenas o PID e o nome dos processos.

Aí, é só pegar o PID dado pelo comando ps e matar o processo:

 # kill -9 PID
-9 - estou forçando para que o mesmo seja morto.

Com essa opção -9 , você mata o processo de uma forma destrutiva, ou seja, ele é forçado a ser morto, então dependendo do serviço ele não irá fazer as tarefas necessárias antes do processo morrer.

O -15 apenas joga um sinal de término. Mas se o processo estiver travado, ou executando alguma coisa, ele não vai terminar e vai apenas ignorar o seu pedido.

O 9 e 15 que usamos aqui são conhecidos como sinais. Um sinal é um meio para interagir (comunicar) com o processo com a finalidade de o sistema interferir em seu funcionamento.

Sintaxe do comando kill:

 # kill SINAL PID

Se o sinal for omitido é usado 15 como padrão.

Ou ainda, pegar o nome e usar o comando:

 # killall firefox

Onde: killall - comando para matar um processo pelo nome.

Na prática no comando kill apenas usamos o -9 e o -15 para matar um processo que esteja travado ou atrapalhando o bom funcionamento do sistema. Conhecendo os demais sinais:

 # kill -l

Para descobrir para que serve cada sinal desse:

```
# man 7 signal
```

Alguns sinais mais usados:

STOP - esse sinal tem a função de interromper a execução de um processo e só reativá-lo após o recebimento do sinal CONT;
CONT - esse sinal tem a função de instruir a execução de um processo após este ter sido interrompido;
TERM - esse sinal tem a função de terminar completamente o processo, ou seja, este deixa de existir após a finalização;
KILL - esse sinal tem a função de "matar" um processo e é usado em momentos de críticos.
HUP - pode ser usado para que um serviço leia novamente seu arquivo de configuração.

Na maioria das distribuições, o agendador de tarefas "cron" sempre vem “startado” quando a máquina é iniciada. Mas neste caso, se não está sendo usado seu processo pode ser morto.

Para localizar esse processo, usa-se um comando muito legal:

```
# pgrep firefox
```

Esse comando irá retornar qual o número PID que está o cron.

```
# pidof firefox
```

Então com o número em mãos, pode matar esse processo:

 # kill -15 PID
Ou
 # kill -s SIGTERM 3222
Ou ainda:
 # pkill -15 firefox

12.3. Background e foreground

Um comando em background é útil quando o mesmo irá demorar um certo tempo e não se quer que o mesmo ocupe o shell, ou seja, quando é necessário continuar trabalhando no shell antes do comando terminar. Por isso que se fala segundo plano, pois o primeiro plano (shell) continua sendo usado por outros processos e esse processo será executado em segundo (atrás do shell).

Então, suponha que está acontecendo uma cópia de alguns arquivos de um diretório para outro e esses arquivos são grandes. Para continuar usando o shell, coloque o comando em segundo plano:

$ cp -R dir dir2 & (Apenas exemplo)

Onde o comando para colocar em background é apenas o & no final do comando em si.

Para ver o processo que está rodando em background:

#jobs

Onde o comando jobs irá mostrar o processo que está em background.

Para colocar esse processo novamente para primeiro plano pegue o número que o jobs passou e digite:

```
#fg 1
```

Onde fg (foreground) traz o processo para primeiro plano. Porque este foi o número do job relacionado ao processo que se quer trazer para foreground.

Exemplificando na prática:

```
#updatedb &
[1] 14082
```

Que é o comando que atualiza a base de dados do comando locate. Coloque o processo em segundo plano. E é possível continuar trabalhando no terminal.

```
#jobs
[1]+ Running updatedb &
```

Mostra o processo que está em segundo plano e o status dele (Rodando)

```
# fg 1
updatedb
```

Com o número que o comando jobs retornou, é só colocar o processo em primeiro plano. Assim o processo ocupa o shell novamente.

Para terminar a execução de um programa em background poderá matá-lo pelo comando kill, pois ele aparece no ps aux. Ou trazer para primeiro plano e dar um CTRL+C, que é um Sinal de Interrupção.

Trazendo o processo novamente para primeiro plano, ele ocupa meu shell, então deve-se jogá-lo novamente para segundo plano:

Lembrando que não pode ir em outro terminal fazer isso!

Então, para fazer isso é necessário parar o processo no shell que está travado com o mesmo :

```
CTRL+Z
```

Essa seqüência de tecla PÁRA (STOP) o processo, mas o mesmo fica em segundo plano parado. Então, é necessário executar o comando jobs novamente:

```
#jobs
```

Assim poderá ver que ele está lá, só que agora parado. Copie o número dele que o jobs te retornou e fala para ele continuar em segundo plano agora:

```
# bg N°
```

Exemplo prático:

Abro um arquivo no VI

```
$ vi teste.txt
```

Suponha que você quer usar o terminal, então pare o Vi por um tempo. Para enviar o VI para segundo plano:

```
CTRL+Z
[1]+ Stopped vi teste.txt
```

Pronto, está parado em segundo plano!

```
# jobs
[1]+ Stopped vi teste.txt
```

Para trazer o vi novamente para o shell:

#fg 1

12.4. Prioridades de processos

Quando um processo no GNU/Linux é iniciado, o mesmo tem uma prioridade padrão. Prioridade é o nível que o sistema irá se preocupar com esse processo. Ou seja, colocar ele na frente de todos os processos da minha lista, ou até colocar ele como último da minha lista.

Todos os processos tem uma prioridade padrão que seria 10, ou seja, estando todos os processos com a mesma prioridade, o sistema irá gerenciar quem vai primeiro ou não. Ao configurar uma prioridade, o sistema (kernel) tratará determinado processo com mais atenção (prioridade alta) ou simplesmente nem dar tanta bola para o mesmo (prioridade baixa). Os comandos usados são:

nice e renice

O nice é usado para quando quero iniciar um processo com uma determinada prioridade. Já o renice é para mudar a prioridade de um processo que já esteja rodando. Mas os dois definem prioridades tanto baixa quanto alta. Esses comandos costumam cair na LPI!

Lembrando que:

-20 ----------10---------- 19
alta----------padrão---------- + baixa

A regra é simples e inversamente proporcional: quanto menor o número, maior sua prioridade. Ou seja, se o seu processo deve ter prioridade alta, coloque um valor baixo, como por exemplo -10.

Por motivos óbvios de segurança, usuários normais não podem alterar a prioridade de seu processo para um valor abaixo de 0 (zero).

Usando o comando top para verificar a prioridade, você deve analisar a coluna NI. O campo "NI" refere-se ao número setado pelo "nice". Observe que vários programas do sistema utilizam números diferentes de prioridade. Observe também que, caso não use o comando "nice" (o que acontece na maioria dos casos), a prioridade assumida por padrão é a "0"; vários programas da listagem do "top" provavelmente devem ter essa prioridade. Exemplo:

Vou iniciar um processo com prioridade alta:

```
# cd /
# nice -n -19 find /boot -name menu.lst
```

O comando acima faz com que ele procure o arquivo menu.lst com uma prioridade muito alta. Para ver a prioridade, vá em outro terminal e digite:

```
# ps -lax
```

Onde a opção -l do comando ps é para listar em formato longo, ou seja, todas as informações. Agora para mudar a prioridade de um processo que está em execução,é preciso do PID dele.

```
# ps aux
```

Altere a prioridade de um determinado processo:

```
#renice 19 -p PID
```

Essa questão de prioridades, usa-se mais em servidores como banco de dados, que precisam de total atenção, já que as gravações no banco são em tempo real. De qualquer forma isso fica transparente para o usuário do sistema.

Os comandos "nice" e "renice" mostram-se muito úteis em diversas situações, principalmente naquelas em que o desempenho e a rapidez são essenciais. No entanto, seu uso deve ser ponderado, uma vez que ao mudar a prioridade de um processo para "-20" ou algum valor próximo a isso o sistema torna-se extremamente lento.

12.5. Comandos complementares de processos

12.5.1. Comando fuser

É interessante comentar também sobre os seguintes comandos: fuser e nohup. O comando fuser mostra qual processo faz uso de um determinado arquivo ou diretório. Sua sintaxe é:

 # fuser opção caminho arquivo/diretório

-k - finaliza o processo que utiliza o arquivo/diretório;
-v - o resultado é mostrado em um padrão de exibição semelhante ao comando ps.

Usa-se muito o fuser para fazer check list para verificar se um determinado serviço está no ar. Veja isso mais adiante na prática.

Suponha que alguém esteja usando o cd-rom, mas você quer desmontá-lo, mas não consegue e nem sabe quem está usando.

Lembre-se que para demonstrar um dispositivo, o mesmo não pode estar em uso.

Então, use o comando abaixo para verificar quem está usando o cd-rom:

```
# fuser -v /media/cdrom
USER PID ACCESS COMMAND
/media/cdrom: leo 5125 ..c.. bash
```

Através desse comando é possível ver que quem está usando o diretório onde está montado o cd-rom é o usuário leo e o PID desse processo é 5125. Se você quiser ver somente quem é o usuário:

```
fuser -u /media/cdrom
/media/cdrom: 5125c(leo)
```

Para matar o processo com o fuser:

```
# fuser -k /media/cdrom
```

Veja algo agora:

```
# fuser -v /media/cdrom
```

Outro exemplo é verificar o PID de um serviço de rede, como por exemplo o SSH:

```
# fuser -v 22/tcp
USER PID ACCESS COMMAND
22/tcp: root 3025 F.... sshd
```

Esse exemplo é um passo da check list que fazemos quando vamos configurar um serviço, veremos isso com detalhe mais adiante. No caso aqui o processo do ssh está rodando e seu PID é 3025.

12.5.2. Comando nohup

O comando nohup permite que o processo fique ativo mesmo quando o usuário faz logout. Em sistemas baseados em Unix (Unix-Like) é normal que eles interrompam processos caso seu dono não esteja mais ativo, por isso, o nohup pode ser muito útil para manter o processo ativo mesmo se o dono fizer logout. Sua sintaxe é:

```
# nohup comando
```