Tópicos Especiais em Fundamentos da Computação
Shell Scripting

Processos

Andrei Rimsa Álvares
andrei@cefetmg.br

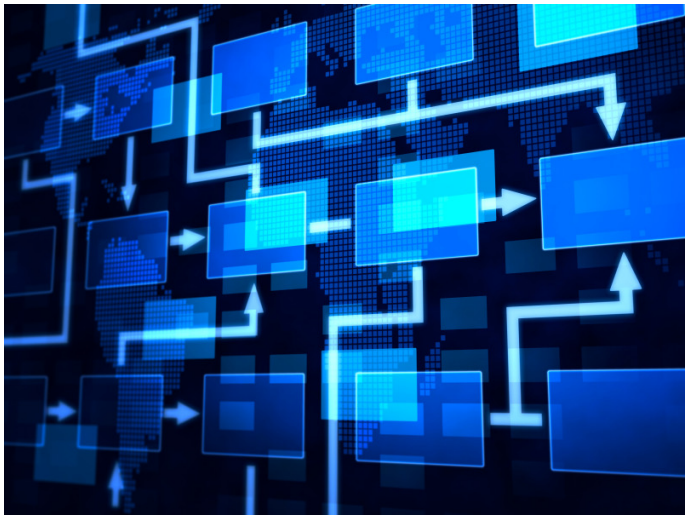
Sumário

- Introdução
- Como funcionam processos
- Controlando processos
- Sinais
- Mais comandos

INTRODUÇÃO

Introdução

- Sistemas operacionais modernos são **multitarefas** (*multitasking*)
  - Eles criam a ilusão de fazer mais de uma coisa ao mesmo tempo ao trocar rapidamente de um programa em execução para outro



- O *kernel* do Linux gerencia isso através do uso de **processos**
  - O kernel organiza os diferentes programas esperando sua vez para usar a CPU

Diagnóstico

- Às vezes o computador pode ficar lento ou uma aplicação pode parar de responder

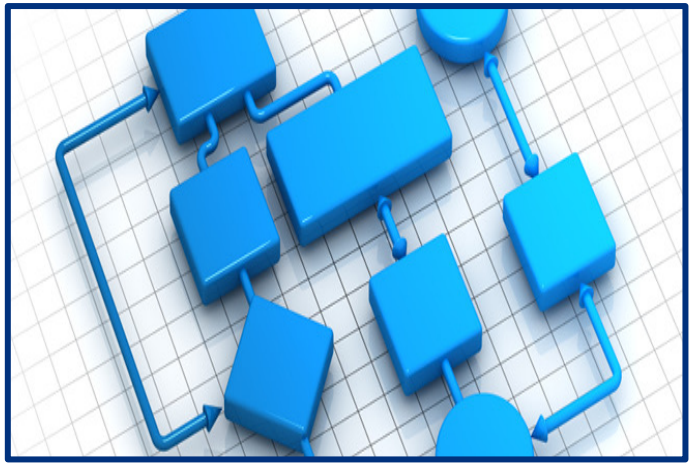
- Serão vistas algumas ferramentas disponíveis na linha de comando que permitem
  - Examinar o que programas estão fazendo
  - Terminar um processo que não esteja comportando corretamente

Comandos

- Alguns dos programas que serão vistos

  - `ps`: reportar a situação instantânea dos processos atuais
  - `top`: mostrar tarefas
  - `jobs`: mostrar trabalhos ativos
  - `bg`: colocar um trabalho no plano de fundo (*background*)
  - `fg`: colocar um trabalho no primeiro plano (*foreground*)
  - `kill`: enviar um sinal para um processo
  - `killall`: matar um processo pelo nome

COMO FUNCIONAM PROCESSOS



Shell Scripting

Processos

- Quando o sistema é inicializado, o *kernel* começa algumas de suas próprias atividades como processos e lança um programa especial chamado `init`

- `init`, por sua vez, executa uma série de *shell scripts* localizados em `/etc/init.d` chamados de *scripts* de inicialização que disparam os serviços disponíveis no sistema
  - Muitos desses serviços são implementados como *daemons*♣: programas que rodam em plano de fundo sem nenhuma interface de usuário

- Mesmo que você não esteja *logado*, o sistema pode estar ocupado executando algumas tarefas

♣: https://en.wikipedia.org/wiki/Daemon_(computing)

Processos

- O fato que um programa pode lançar (executar) outros programas é expresso no esquema de processos como
  - Um **processo pai** (*parent process*) que produz um **processo filho** (*child process*)

- O *kernel* mantém informações sobre cada processo para ajudar a manter as coisas organizadas
  - Número chamado de identificador do processo (PID: *process ID*)
    - PIDs são atribuídos em ordem crescente (`init`: PID `1`)
  - Áreas de memórias alocadas para o processo
  - Estado de prontidão para resumir sua execução
  - Donos, IDs de usuário, usuários efetivos (assim como arquivos)

Exibindo Processos

- A ferramenta mais comumente usada para ver processos é `ps` (embora existam outras)
  - Ela possui muitas opções, mas em sua forma mais simples é dado conforme a seguir

```
$ ps
  PID TTY          TIME CMD
 5198 pts/1    00:00:00 bash
10129 pts/1    00:00:00 ps
```

Por padrão, `ps` não exibe muita coisa, apenas os processos associados à sessão do terminal

Como exibir todos os processos do sistema?

Exibindo Mais Processos

- Adicionar a opção `x` (sem começar com traço) faz com que `ps` exiba todos os processos do usuário independente de qual terminal (se tiver) o estiver controlado

O símbolo de interrogação (`?`) na coluna `TTY` indica que nenhum terminal o controla

```
$ ps x
   PID TTY  STAT   TIME COMMAND
 2799 ?    Ssl    0:00 /usr/libexec/bonobo-activation-server -ac
 2820 ?    Sl     0:01 /usr/libexec/evolution-data-server-1.10 --
15647 ?    Ss     0:00 /bin/sh /usr/bin/startkde
15751 ?    Ss     0:00 /usr/bin/ssh-agent /usr/bin/dbus-launch -
15754 ?    S      0:00 /usr/bin/dbus-launch --exit-with-session
15755 ?    Ss     0:01 /bin/dbus-daemon --fork --print-pid 4 -pr
15774 ?    Ss     0:02 /usr/bin/gpg-agent -s -daemon
15793 ?    S      0:00 start_kdeinit --new-startup +kcminit_start
15794 ?    Ss     0:00 kdeinit Running...
15797 ?    S      0:00 dcopserver -nosid
...
```

Dica: use `less` para paginar a saída

Exibindo Mais Processos

- Uma nova coluna chamada `STAT` (acrônimo para estado) foi adicionada à saída que revela o estado atual do sistema; a tabela a seguir mostra algumas dessas opções

Estado	Significado
`R`	**Executando**: o processo está executando ou pronto para executar
`S`	**Dormindo**: o processo não está executando; está esperando por um evento, como uma tecla ou pacote de rede
`D`	**Dormindo ininterruptamente**: o processo está esperando por um I/O como um disco
`T`	**Parado**: o processo foi instruído para parar
`Z`	**Defeituoso (*zombie*)**: um processo filho que terminou mas não foi coletado propriamente pelo seu processo pai
`<`	**De alta prioridade**: é possível conceder mais importância (mais tempo de CPU) para um processo; essa propriedade é chamada de *niceness*
`N`	**De baixa prioridade**: um processo com baixa prioridade irá receber tempo de processador apenas depois que processos de alta prioridade tenham sido servidos

Exibindo Ainda Mais Processos

- Um outro conjunto popular de opções é `aux` que mostra os processos de todos os usuários

```
$ ps aux
USER PID %CPU %MEM  VSZ RSS TTY STAT START   TIME COMMAND
root   1  0.0  0.0 2136 644 ?   Ss   Mar05   0:31 init
root   2  0.0  0.0    0   0 ? S<   Mar05   0:00 [kt]
root   3  0.0  0.0    0   0 ? S<   Mar05   0:00 [mi]
root   4  0.0  0.0    0   0 ? S<   Mar05   0:00 [ks]
root   5  0.0  0.0    0   0 ? S<   Mar05   0:06 [wa]
root   6  0.0  0.0    0   0 ? S<   Mar05   0:36 [ev]
root   7  0.0  0.0    0   0 ? S<   Mar05   0:00 [kh]
...
```

Usar as opções que não começam com traço invocam `ps` no modo (estilo) BSD que a versão Linux consegue emular esse comportamento

Exibindo Ainda Mais Processos

- Novas colunas foram adicionadas a saída que são sumarizadas na tabela a seguir

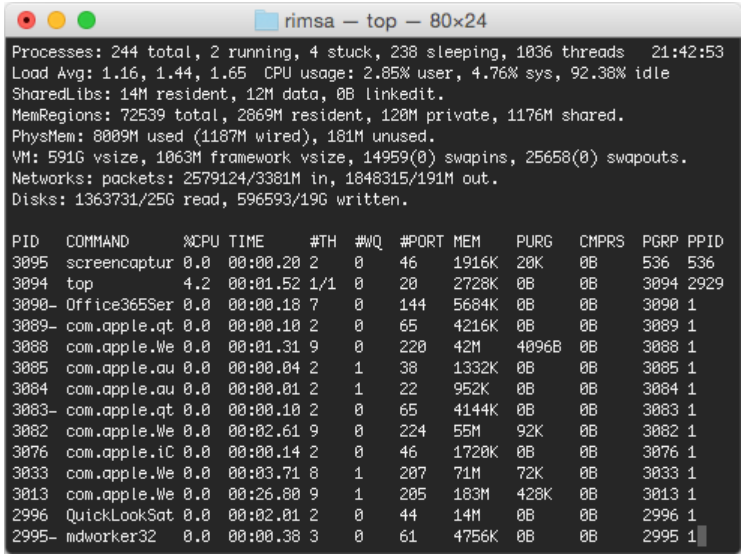
Cabeçalho	Significado
`USER`	Identificador do usuário, ou seja, o dono do processo
`%CPU`	Uso de CPU em percentagem
`%MEM`	Uso de memória em percentagem
`VSZ`	Tamanho da memória virtual
`RSS`	A quantidade de memória física (RAM) que o processo está usando (em *kilobytes*)
`START`	Hora de início do processo (maiores que 24h usa-se data)

Exibindo Processos Dinamicamente

- O comando `ps` permite apenas uma visualização instantânea do sistema (no momento em que é executado), enquanto o comando `top` permite uma visão mais dinâmica da atividade da máquina

```
$ top
```

- Esse programa mostra contínuas atualizações (a cada 3 segundos) dos processos do sistemas listados em ordem crescente de atividade



`top` aceita um conjunto de comandos de teclado, como `h` para ajuda e `q` para sair

Curiosidade: o nome `top` (acima) vem do fato que os programas com mais uso de CPU são vistos acima dos outros

CONTROLANDO PROCESSOS

Shell Scripting

Um Processo Cobaia

- Agora que já se sabe como ver e monitorar processos, será visto como ganhar algum controle sobre eles

- Para isso, será usado um pequeno programa chamado de `xlogo` que será usado como cobaia
  - Esse programa é uma amostra gráfica fornecido pelo *X Window System* que simplesmente mostra uma janela redimensionável com o logo do *X*

```
$ xlogo
```

Repare que o terminal fica preso enquanto o programa está em execução

Interromper um Processo

- Com o programa ainda em execução, volte para o terminal e aperte no teclado `CTRL-C`

```
$ xlogo
^C
```

  - Em um terminal, pressionar `CTRL-C` interrompe um programa
    - Isso faz com o que haja um pedido educado para que o programa termine sua execução
  - Com isso, a janela é fechada e o *prompt* é retornado

Cuidado: nem todo programa pode ser fechado com `CTRL-C`

Executar um Processo em Plano de Fundo

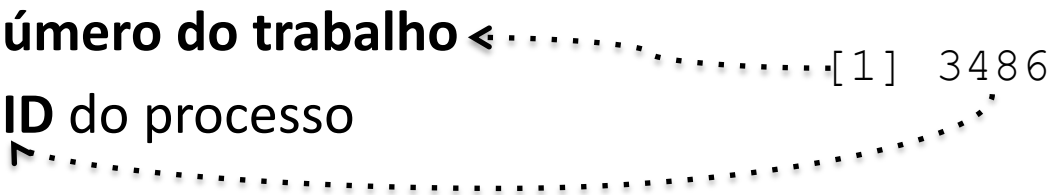
- O terminal possui dois modos nos quais os programas são executados
  - **Primeiro plano:** com as coisas visíveis na superfície, como o *prompt* da *shell*
  - **Segundo plano/plano de fundo:** com as coisas escondidas abaixo da superfície
- Por exemplo, para obter o *prompt* da *shell* sem terminar o programa, pode-se colocá-lo em plano de fundo
- Para executar um programa e colocá-lo imediatamente no plano de fundo, basta seguir o comando com caractere E comercial (`&`)

```
$ xlogo &
[1] 3486
```

O que significam esses números na saída?

Informações sobre Processos no Plano de Fundo

- A mensagem de saída faz parte de uma funcionalidade da *shell* chamada de controle de trabalho (*job control*), que indica
  - O **número do trabalho**
  - O **PID** do processo



- Para ver o processo pode-se executar o comando `ps`

```
$ ps
  PID TTY           TIME CMD
 3264 ttys000    0:00.03 -bash
 3486 ttys000    0:00.01 xlogo
 3496 ttys000    0:00.01 ps
```

- O controle de trabalho da *shell* permite visualizar os trabalhos lançados pelo próprio terminal através do comando `jobs`

```
$ jobs
[1]+  Running                 xlogo &
```

Retornar um Processo ao Primeiro Plano

- Um processo em plano de fundo é imune a entradas de teclado, inclusive tentativas de interrompê-lo com `CTRL-C`

- Para retornar um processo para o primeiro plano, usa-se o comando `fg` conforme exemplo a seguir

```
$ jobs
[1]+  Running                 xlogo &
$ fg 1
xlogo
```

Agora é possível terminar o programa com `CTRL-C`

  - `fg` seguido do símbolo de porcentagem mais o número do trabalho (*jobspec*) traz o trabalho para primeiro plano
  - Se houver apenas um trabalho, o uso de *jobspec* é opcional

Parar (Suspender) um Processo

- As vezes é interessante parar um processo sem terminá-lo; isso é comum para permitir que um processo seja movido do primeiro plano para o plano de fundo

- Para parar um processo no primeiro plano, digite `CTRL-Z` depois de executar o programa

```
$ xlogo
^Z
[1]+  Stopped                 xlogo
```

Pode-se verificar que o programa realmente parou ao tentar redimensionar a janela, que não é mais renderizada

Resumir a Execução de um Processo

- Depois de suspenso, pode-se restaurar o processo no primeiro plano (usando `fg`) ou mover o processo para o plano de fundo (usando `bg`)

```
$ bg 1
[1]+ xlogo &
```

  - Assim como no comando `fg`, o *jobspec* é opcional se houver apenas um trabalho

Dica: mover um programa do primeiro plano para o plano de fundo é útil quando se executa um programa gráfico na linha de comando, mas se esquece de colocar o & no final do comando

Por que alguém executaria um programa gráfico a partir da linha de comando?

Por Que Executar Programas Gráficos do Terminal?

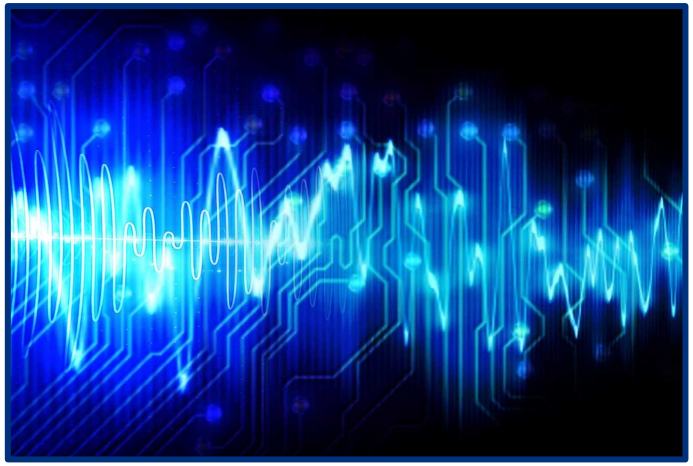
- Duas principais razões

1) O programa que se deseja executar pode não estar listado nos menus do sistema de janelas (como `xlogo`)

2) Executar um programa da linha de comando permite ver mensagens de erro que seriam invisíveis se o programa fosse lançado graficamente
   - Às vezes, um programa pode falhar na sua inicialização, pelo terminal pode-se ver uma mensagem de erro para ajudar a diagnosticar o problema

Dica: programas gráficos podem ter opções em linha de comando úteis e interessantes

SINAIS



Matar Processos

- O comando `kill` é usado para matar (terminar) processos
  - Isto permite terminar a execução de um programa que está se comportando mal ou se recusa a terminar por conta própria

```
$ xlogo &
[1] 28401
$ kill 28401
[1]+ Terminated xlogo
```

Dica: pode-se matar o processo usando o número do trabalho (`%1`)

Sinais

- O comando `kill` não "mata" exatamente processos, ao invés disso ele manda sinais para eles; sinais são uma das formas que o sistema operacional comunica com programas

- Já se usou sinais com o uso dos atalhos `CTRL-C` e `CTRL-Z`: quando o terminal recebe uma das combinações, ele manda um sinal para o programa no primeiro plano
  - No caso de `CTRL-C` é enviado o sinal `INT` (*Interrupt*)
  - No caso de `CTRL-Z` é enviado o sinal `TSTP` (*Terminal Stop*)

- Programas por sua vez "escutam" por sinais e tomam ações baseados neles quando recebidos
  - Isso permite que programas possam salvar o progresso quando recebe um sinal de terminação

Enviar Sinais para Processos

- A sintaxe mais comum para o comando `kill` é dada por

```
kill [-signal] PID
```

  - Se nenhum sinal for especificado na linha de comando, então é enviado por padrão o sinal `TERM` (*Terminate*)

Sinais Comuns

- Alguns sinais comuns enviados para processos:

#	Nome	Significado
1	`HUP`	**Desligar:** indica que o terminal que controla o programa desligou (como um modem); alguns *daemons* usam esse sinal para reinicialização (como Apache)
2	`INT`	**Interromper:** usualmente termina um programa; tem o mesmo efeito de `CTRL-C` enviado pelo terminal
9	`KILL`	**Matar:** enquanto programas podem escolher como tratar sinais (até ignorar), esse nunca é enviado para o programa alvo, ao invés disso o *kernel* termina imediatamente o programa; nesse caso, o programa não tem a oportunidade de fazer uma saída limpa; deve ser usado somente em último caso
15	`TERM`	**Terminar:** sinal padrão enviado pelo programa `kill`; se o programa estiver "vivo" o suficiente para receber sinais irá terminar
18	`CONT`	**Continuar:** restaura um processo após um sinal de `STOP`
19	`STOP`	**Parar:** pausa um processo sem terminá-lo; assim como `KILL` esse sinal não é enviado para o processo alvo

Exemplo

- Enviando o sinal `HUP` com o comando `kill` através de seu número (no caso `1`) para o programa `xlogo`

```
$ xlogo &
[1] 13546
$ kill -1 13546
[1]+ Hangup xlogo
```

Dica: pode-se usar o comando `nohup` antes do comando para torná-lo imune a sinais `HUP`:

```
$ nohup xlogo &
```

Outro Exemplo

- Além de especificar um sinal pelo seu número, pode-se usar pelo seu nome, inclusive o prefixando com `SIG`

```
$ xlogo &
[1] 13601
$ kill -INT 13601
[1]+ Interrupt xlogo
```

```
$ xlogo &
[1] 13608
$ kill -SIGINT 13608
[1]+ Interrupt xlogo
```

Cuidado: processos, assim como arquivos, possuem dono; para isso deve-se ser o dono ou superusuário para enviar sinais ao processo

Existem mais sinais que podem ser enviados?

Mais Sinais

- Mais alguns sinais comuns enviados para processos:

#	Nome	Significado
3	`QUIT`	**Sair:** fecha normalmente o programa
11	`SEGV`	**Violação de segmentação:** sinal enviado ao programa quando faz uso ilegal de memória, como tentar ler de uma memória não mapeada ou escrever em uma memória que não tenha permissão
20	`TSTP`	**Parar do terminal:** sinal enviado pelo terminal quando pressionado `CTRL-Z`; esse sinal é recebido pelo programa que pode escolher ignorá-lo
28	`WINCH`	**Mudança na janela:** sinal enviado quando há um redimensionamento da janela; programas como `top` e `less` se redesenham quando recebem esse sinal

Dica: uma lista completa de sinais pode ser vista pelo comando:

```
$ kill -l
```

Enviar Sinais para Múltiplos Processos

- É possível enviar sinais para múltiplos processos para um determinado programa de usuário usando o comando `killall`

```
killall [-u user] [-signal] name...
```

- Por exemplo, iniciar vários processos `xlogo` e terminar todos simultaneamente

```
$ xlogo &
[1] 18801
$ xlogo &
[2] 18802
$ killall xlogo
[1]- Terminated
[2]+ Terminated
```

MAIS COMANDOS

Shell Scripting

Comandos

- Como monitorar processos é uma importante tarefa administrativa, existem muitos outros comandos para auxiliar nessa tarefa

Comando	Propósito
`pstree`	Mostra uma lista de processo em formato de árvore exibindo a relação de pai/filho entre processos
`vmstat`	Mostra a utilização instantânea do sistema incluindo memória, swap e uso de disco; para ver os dados continuamente, use o comando seguido de um tempo de atraso (em segundos) para atualizações (ex.: `vmstat 5`)
`xload`	Um programa gráfico que desenha um gráfico mostrando a carga do sistema ao longo do tempo
`tload`	Similar ao programa `xload`, mas desenha o gráfico no terminal

ISSO É TUDO, PESSOAL!