

# Sessão 4: Processos



As atividades desta sessão serão realizadas na máquina virtual *Client\_Linux*.

## 1) Descobrindo o número de processos em execução

1. Quantos processos estão sendo executados na máquina no momento? Use o comando **wc** para contá-los.

```
# ps aux | sed -n '1!p' | wc -l
71
```

2. Faça um *script* que liste o número de processo que cada usuário está executando.

O *script shell* abaixo mostra um exemplo de solução para o problema proposto:

```
#!/bin/bash

users=( $( ps aux | awk '{ if (NR>1) print $1 }' | sort | uniq ) )

for (( i=0; i<${#users[@]}; i++ )); do
    nproc=$( ps aux | grep "${users[$i]}" | wc -l )
    echo "User ${users[$i]} has $nproc active processes"
done
```

## 2) Descobrindo o PID e o PPID de um processo

1. Quais os valores de **PID** e **PPID** do shell que você está utilizando no sistema?

```
$ echo -e "PID: $$\nPPID: $PPID"
PID: 1016
PPID: 1015
```

2. Faça um *script* que liste todos os processos que foram iniciados pelo processo **init**. A lista não deve conter mais de uma ocorrência do mesmo processo.

O *script shell* abaixo mostra um exemplo de solução para o problema proposto:

```
#!/bin/bash
```

```
pinit=( $( ps -eo ppid,comm | egrep -e "^ *1 " | sort | uniq | awk {'print $2'} ) )  
pinit_count=${#pinit[@]}
```

```
echo "$pinit_count processes started by init (1):"
```

```
for (( i=0; i<$pinit_count; i++ )); do  
    echo "  ${pinit[$i]}"  
done
```

### 3) Estados dos processos

1. Qual o status mais frequente dos processos que estão sendo executados no sistema? Você saberia explicar por quê?

```
$ ps aux | awk '{print $8}' | sort | uniq -c | sort -n | tac  
24 S  
23 S<  
16 Ss  
4 S+  
1 STAT  
1 Ssl  
1 Ss+  
1 SN  
1 R+  
1 D+
```

O estado mais frequente é *sleep*, porque apenas um processo pode estar sendo executado pela CPU em um dado momento.

### 4) Alternando a execução de processos

1. Execute o comando `$ sleep 1000` diretamente do terminal.

```
$ sleep 1000
```

2. Pare o processo e mantenha-o em memória.

Basta digitar a combinação de teclas **CTRL + Z**.

```
$ sleep 1000  
^Z  
[1]+  Parado
```

3. Liste os processos parados.

```
$ jobs
[1]+  Parado                  sleep 1000
```

4. Coloque-o em *background*.

```
$ bg
[1]+ sleep 1000 &
$ jobs
[1]+  Executando              sleep 1000 &
```

5. Verifique se o comando `sleep 1000` está rodando.

```
$ ps ax | egrep 'sleep 1000$'
2178 pts/0    S          0:00 sleep 1000
```

6. É possível cancelar a execução desse comando quando ele está rodando em *background*? Caso seja possível, faça-o.

```
$ kill 2178
$ ps ax | egrep 'sleep 1000$'
[1]+  Terminado              sleep 1000
```

## 5) Identificando o RUID e o EUID de um processo

1. Logado como o usuário `aluno`, execute o comando `passwd` no seu terminal. Antes de mudar a senha, abra uma segunda console e autentique-se como `root`. Verifique o `RUID` e o `EUID` associados ao processo `passwd`. Esses valores são iguais ou diferentes? Você saberia explicar por quê? Por fim, cancele a execução do processo `passwd`.

Na primeira console, execute:

```
$ passwd
Mudando senha para aluno.
Senha UNIX (atual):
```

Antes de digitar a senha, abra uma segunda console como `root` e execute:

```
# ps -eo user,ruser,comm | egrep '^USER | passwd$'
USER      RUSER      COMMAND
root      aluno      passwd

# which passwd
/usr/bin/passwd
# ls -lh /usr/bin/passwd
-rwsr-xr-x 1 root root 53K Mai 17 2017 /usr/bin/passwd
```

Os valores são diferentes porque o binário **passwd** possui o bit *SUID* ativado. O **RUID** (*real uid*) é do usuário que está executando o comando e o **EUID** (*effective uid*) é o do usuário **root**, que é o dono do arquivo.

## 6) Definindo a prioridade de processos

1. Verifique as opções do comando **nice** e em seguida, execute o comando abaixo, verificando sua prioridade, utilizando o comando **ps**:

```
# nice -n -15 sleep 1000 &
[1] 2289
```

Basta executar o comando **# ps lax** e buscar o processo relevante, verificando o valor da quinta coluna. Em uma única linha e de forma mais específica, podemos fazer:

```
# ps lax | egrep ' sleep 1000$' | awk '{print $5}'
2289 5
```

2. Repita o comando do primeiro item, passando para o comando **nice** o parâmetro **-n -5**. Verifique como isso afeta a prioridade do processo. Ela aumentou, diminuiu ou permaneceu a mesma?

```
# nice -n -5 sleep 1000 &
[2] 2312
# ps lax | egrep ' sleep 1000$' | awk '{print $3, $5}'
2289 5
2312 15
```

A prioridade diminuiu, porque quanto maior o valor na coluna **PRI**, menor a prioridade do processo.

## 7) Editando arquivos crontab para o agendamento de tarefas

Neste exercício, trabalharemos com o comando `crontab`, utilizado para editar os arquivos `cron` do agendador de tarefas do sistema. Esses arquivos serão verificados pelo *daemon* `cron` periodicamente em busca de tarefas para serem executadas pelo sistema.



Para entender o funcionamento do `crontab`, o primeiro passo é ler as páginas do manual relevantes. Para o comando `crontab` em si, consulte a seção 1 do manual:

```
$ man 1 crontab
```

Para o formato de um arquivo de configuração `crontab`, consulte a seção 5:

```
$ man 5 crontab
```

1. Existe alguma entrada de `crontab` para o seu usuário?

```
$ crontab -l
no crontab for aluno
```

2. Que opção deve ser usada para editar o seu arquivo de `crontab`?

```
$ crontab -e
no crontab for aluno - using an empty one

Select an editor. To change later, run 'select-editor'.
 1. /bin/nano      <---- easiest
 2. /usr/bin/vim.basic
 3. /usr/bin/vim.tiny

Choose 1-3 [1]: 1
No modification made
```

## 8) Agendando uma tarefa no daemon cron

Neste exercício, será necessário enviar mensagens de correio eletrônico. Para isso, você deverá utilizar o comando `mail`; o instrutor pode fornecer as informações básicas sobre ele. Um exemplo do uso desse comando para enviar uma mensagem ao endereço `fulano@dominio` com o assunto *Mensagem de teste* é:

```
$ mail fulano@dominio -s "Mensagem de teste" < /dev/null
```

1. Configure o `crontab` para que uma mensagem de correio eletrônico seja enviada automaticamente pelo sistema, sem interferência do administrador às 20:30 horas.

Utilize o comando `$ crontab -e` para editar o `crontab` e inserir a linha:

```
30 20 * * * mail fulano@dominio -s "Mensagem de teste" < /dev/null
```

2. Como verificar se a configuração foi feita corretamente?

```
$ crontab -l | egrep -v '^#'  
30 20 * * * mail fulano@dominio -s "Mensagem de teste" < /dev/null
```

3. Qual o requisito fundamental para garantir que a ação programada será executada?

O daemon do `cron` deve estar em execução e a sintaxe do `crontab`, incluindo a linha de comando utilizada, deve estar correta.

4. Há como confirmar se a mensagem foi efetivamente enviada, sem consultar o destinatário?

Verifique no arquivo `/var/log/syslog` se a tarefa foi executada no horário correto com sucesso. Você deve ver uma entrada do tipo:

```
/var/log/syslog:Aug 7 17:40:01 cliente CRON[2524]: (aluno) CMD (COMMAND)
```

Dependendo da distribuição Linux em uso, as mensagens relativas ao `cron` podem estar em `/var/log/syslog`, `/var/log/cron.log`, `/var/log/daemon.log` ou outros arquivos. Verifique na documentação do fabricante/mantenedor.

5. Dê dois exemplos de utilização desse mecanismo para apoiar atividades do administrador de sistemas.

Podemos, por exemplo, utilizar o `cron` para agendamento de backups e limpeza de diretórios temporários.

6. Faça um script que liste os arquivos sem dono do sistema e envie a lista por e-mail ao usuário `root`.

O *script shell* abaixo mostra um exemplo de solução para o problema proposto, com a característica adicional de guardar os logs enviados por e-mail em um diretório dentro do *home* do `root`:

```
#!/bin/bash

LOGDIR="/root/nouser_logs"

[ ! -d $LOGDIR ] && mkdir $LOGDIR

curlog="$LOGDIR/nouser_$( date +%Y%m%d ).log"
find / -nouser -print > $curlog
mail -s "Files without ownership for $( date )" root < $curlog
```

7. Agende no crontab do usuário `root` o script do item 6, de modo que ele seja executado de segunda a sexta às 22:30 horas.

Logado como usuário `root`, digite o comando `# crontab -e` para editar o `crontab` e insira a linha a seguir:

```
30 22 * * 1-5 /root/scripts/find_nouser.sh
```

## 9) Listando e removendo arquivos crontab

1. Liste o conteúdo do seu arquivo de `crontab` e, em seguida, remova-o. Quais as opções utilizadas para executar as ações demandadas?

```
$ crontab -l | egrep -v '^#'
30 20 * * * mail fulano@dominio -s "Mensagem de teste" < /dev/null

$ crontab -r
$ crontab -l
no crontab for aluno
```

## 10) Entendendo o comando exec

1. Execute o comando `$ exec ls -l`. Explique o que aconteceu.

```
# whoami
root
# exec ls -l /mnt/
total 0

$ whoami
aluno
```

O shell corrente foi finalizado. Sempre que um comando é executado, um novo processo é criado. Já quando um comando é executado como argumento do comando `exec`, a imagem do

shell corrente é substituída pela do processo invocado, e quando esse processo encerra sua execução já não há mais shell de retorno.