Sistemas Operacionais – Profa. Roberta Lima Gomes
Lista de Exercícios Práticos
Linux – Processos

1. **[o comando ps]** Através do comando ps pode-se examinar os processos correntes. Por exemplo:

```
...:~$ ps
  PID TTY STAT  TIME COMMAND
   45 v02 S     0:00 -bash
  105 v02 R     0:00 ps
```

   Vamos entender o que está acontecendo: há um processo (o shell) com o qual você dialoga a fim de executar comandos. Esse é o processo de número 45. O comando solicitado consiste na execução do programa ps. Esse programa, ao ser executado, descobre que neste momento há dois processos: ele próprio (de número 105) e o shell.

   Na verdade além desses dois pode haver muitos outros que o ps não exibe. Através das opções a e x entretanto pode-se exibir todos os processos correntes:

```
...:~$ ps ax
  PID TTY STAT  TIME COMMAND
    1  ?  S     0:00 init [5]
    6  ?  S     0:00 bdflush (daemon)
    7  ?  S     0:00 update (bdflush)
   25  ?  S     0:00 /usr/sbin/crond -l10
   36  ?  S     0:00 /usr/sbin/syslogd
   38  ?  S     0:00 /usr/sbin/klogd
   40  ?  S     0:00 /usr/sbin/inetd
  101 v01 S     0:00 /sbin/agetty 38400 tty1 linux
  106  ?  Z     0:00 (atrun)
   45 v02 S     0:00 -bash
  107 v02 R     0:00 ps ax
```

   Neste caso, todos os processos além do shell e o ps dizem respeito apenas à administração do sistema. O agetty (por exemplo) é o processo que está controlando o login no console virtual 1. É a ele que você informa o seu username e password ao logar nesse console.

   O número de um processo (PID) é usado para identificá-lo dentre os outros, por exemplo quando é necessário interromper prematuramente a sua execução. O unix vai numerando os processos em ordem crescente, à medida em que vão sendo criados.

2. Reproduza os exemplos acima de uso do ps, e experimente também usá-lo com a opção "u" (por exemplo "ps u" ou "ps aux".

3. Invoque a man page do ps com "man ps" e leia nela o que significa o campo "STAT". Lembre-se que você pode localizar palavras ao ler uma man page com o comando "/" (para uma lista de comandos pressione "h").

4. **[execução em background]** Por vezes um processo pode ser de execução demorada. Enquanto ele não terminar, o shell permanecerá aguardando, e você também. Para evitar esse problema, pode-se disparar o processo em "background". Vejamos um exemplo. O comando abaixo irá comprimir todos os arquivos do diretório corrente. Compressão de arquivos é uma típica operação exigente em termos de cpu, por isso um comando assim pode ser demorado:

```
...:~$ gzip -9 *
```

   Enquanto a compressão não terminar, você não poderá usar o shell para disparar novos comandos. Se, por outro lado, essa compressão for colocada em background, o shell permanecerá livre para novos comandos:

```
...:~$ gzip -9 * &
```

É a ocorrência do caracter "&" no final do comando instrui o shell para dispará-lo em "background".

Crie um subdiretório, povoe ele com arquivos e use-o para disparar a compressão em background como indicado acima. Por exemplo:

```
...:~$ cd
...:~$ mkdir lixo
...:~$ cd lixo
...:~$ cp /bin/* .
...:~$ gzip -9 * &
...:~$ ps u
(... outros comandos ...)
...:~$ cd ; rm -rf lixo
```

5. **[Prioridade]** Em um sistema operacional onde vários processos podem estar simultaneamente em execução, surge às vezes o problema de ser necessário privilegiar ou desprivilegiar a execução de um processo ou um grupo de processos.

   Para agilizar a execução de um processo urgente ou para evitar que um processo demorado atrapalhe a execução de outros, o sistema operacional atribui a cada processo uma prioridade, que pode ser alterada quando necessário.

   Nesta atividade iremos limitar-nos ao comando "nice". Com ele, pode-se disparar um processo com baixa prioridade, o que significa que o sistema operacional irá privilegiar a execução dos outros processos em detrimento a ele.

   O uso típico do "nice" é impedir que um processo demorado atrapalhe o uso interativo do sistema pelo(s) usuário(s) logados nele. No exemplo que demos antes da compressão, o disparo através do nice seria assim:

```
hal:~$ nice gzip -9 * &
```

6. **[Herança]** O “process ID” não se altera após um exec(), mas o novo programa herda alguma propriedades adicionais do processo chamador:

- Process ID and parent process ID
- Real user ID and real group ID
- Process group ID
- Session ID
- Controlling terminal
- Time left until alarm clock
- Current working directory
- Root directory
- File mode creation mask
- File locks
- Process signal mask
- Pending signals
- Resource limits
- Nice value (on XSI-conformant systems)
- Values for tms_utime, tms_stime, tms_cutime, and tms_cstime

PARA PENSAR...

--- Em geral há um limite superior pequeno para o número que um processo pode ter (por exemplo 32767). O que você acha que acontece quando esse limite é atingido?

--- Em UNIX, mesmo algumas operações extremamente simples como listar os arquivos de um diretório envolvem o disparo de pelo menos um novo processo. No entanto a troca do diretório corrente (comando "cd") não provoca o disparo de processo algum. Você saberia explicar porquê?