

## Conceito de Processo

Em sistemas multitarefa (como o unix), múltiplas tarefas podem estar sendo realizadas simultaneamente, por exemplo a formatação de um disquete, a impressão de um arquivo e a edição de um texto. A cada uma delas corresponde um programa que a executa. Um programa em execução é um processo.

Num computador que possui apenas uma cpu, na verdade apenas um processo pode estar sendo executado em cada instante. O que se faz é executar um processo durante uma fração fixa de segundo, congelá-lo (coloca-se ele no estado “pronto” ou “ready”) e passar a executar um outro processo durante essa mesma fração de segundo, e assim por diante. Isso cria a **ilusão** de simultaneidade.

## O comando ps

Por meio do comando ps pode-se examinar os processos correntes. Por exemplo:

```
mypc:~$ 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 exhibe para não poluir a saída. Por meio das opções ‘a’ e ‘x’, entretanto, pode-se exhibir todos os processos correntes:

```
mypc:~$ ps ax
  PID TTY STAT  TIME COMMAND
    1 ?    S    0:00 init [5]
    6 ?    S    0:00 bdf flush (daemon)
    7 ?    S    0:00 update (bdf flush)
   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 pts/0 S    0:00 /sbin/agetty 38400 tty1 linux
  106 ?    Z    0:00 (atrun)
   45 pts/0 S    0:00 -bash
  107 pts/0 R    0:00 ps ax
```

...

Nesse caso, todos os processos, além do shell e o ps, são exibidos, muitos dizem respeito apenas à administração do sistema. O agetty (por exemplo) é o processo que está controlando o login no *pseudoterminal* 0. 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.

**Exercício 1:** Reproduza os exemplos acima de uso do ps, e experimente também usá-lo com a opção "u" (por exemplo "ps u" ou "ps aux").

**Exercício 2:** 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").

**Exercício 3:** 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?

## 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:

```
mypc:~$ 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:

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

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

**Exercício 4:** Crie um subdiretório, povoe ele com vários arquivos e use-o para disparar a compressão em background como indicado acima. Por exemplo:

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

## Prioridade

Num sistema operacional em que vários processos podem estar simultâneamente 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.

Neste curso 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 por meio do nice seria assim:

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

**Exercício 5:** Execute o comando acima. Agora leia a man page do comando `renice` para saber como alterar a prioridade de um processo após o seu disparo.

**Exercício 6:** Crie um processo em background usando o comando `nice` e alterando sua prioridade. Em seguida, execute o comando "ps 1" ou "ps lax" para verificar o valor dessa prioridade (veja as colunas "PRI" e "NI" printadas no comando "ps").

## O que é *Environment*?

*Environment* é uma coleção de "variáveis de ambiente" que servem de parâmetros para os programas que o sistema executa. Cada processo possuiu seu conjunto de variáveis de ambiente, que em geral é herado (copiado) do pai do processo. Uma das variáveis mais conhecidas, comum ao unix e ao msdos, é a variável PATH. Quando se solicita na linha de comandos do shell a execução de um programa, o shell irá procurar o arquivo executável (correspondente ao comando) nos diretórios relacionados na variável PATH.

Frequentemente, torna-se necessário para o usuário lidar com essas variáveis para customizar os aplicativos que utiliza, por isso convém ter algumas noções precisas nesse campo.

Vejamos um exemplo simples. O comando "man" via de regra consulta a variável "PAGER" para saber qual paginador é o preferido do usuário. O paginador padrão costuma ser o "more", mas há outros, como o "less" da Free Software

Foundation. Ao invocar uma man page (por exemplo "man ps") você poderá identificar o paginador em uso por meio do prompt de comandos ":" (típico do "less") no canto inferior esquerdo ou pela string "more" ou "mais" (típica do comando "more") no mesmo canto.

Uma pessoa que prefira usar o more irá atribuir "more" à variável PAGER, enquanto uma pessoa que prefira o less irá atribuir "less".

Uma diferença importante entre as variáveis PATH e PAGER citadas é que PATH é consultada principalmente pelo shell, enquanto que PAGER é consultada por uma aplicação (man) disparada através do shell. No *msdos* isso seria irrelevante porque as variáveis de environment são "do sistema", pouco importando quem as está consultando. No Unix, o environment é "*per-process*", portanto o shell precisa ser "avisado" de que aquela variável deve ser exportada para as aplicações:

```
mypc:~$ PAGER=more          %%% or "set PAGER=more"
mypc:~$ export PAGER
```

Um outro detalhe é que nesse ponto a sintaxe depende do shell em uso. O Unix conta com basicamente duas famílias de shells, uma baseada no **Bourne Shell** e outra no **C Shell**. Num shell com sintaxe estilo C-shell, os comandos anteriores teriam que ser substituídos pelo "setenv":

```
mypc:~$ setenv PAGER=more
```

**Exercício 7:** Examine o valor da variável PAGER. Um modo de fazer isso é executar o comando "echo \$PAGER". O comando "set" executado sem argumentos exhibe todas as variáveis atualmente definidas... Execute esse comando também.

**Exercício 8:** Atribua repetidamente "more" e "less" para a variável PAGER conforme os exemplos acima executando também uma leitura de man page (por exemplo "man ps") para confirmar a troca do paginador.

**Exercício 9:** Examine o(s) arquivo(s) de inicialização do shell que você usa para saber quais variáveis são criadas ou inicializadas neles. Esses arquivos são citados em geral no final da man page (seção "FILES"). No Linux, temos os arquivos ".bashrc", que estão presentes no home, no diretório "/etc/skel/" e "/root". Além de variáveis de ambientes, o usuário também pode definir "alias" para comandos. Um excelente alias é o "ll" para executar o comando "ls -aF". Defina esse comando no seu ".bashrc" (basta adicionar a linha "alias ll='ls -aF)').

**Exercício 10:** No 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?

## O que é swap?

A memória é um dos componentes mais caros de um computador. Ao mesmo tempo, a situação habitual é que possuímos menos memória do que necessitamos para fazer tudo o que queremos.

Se pudéssemos visualizar os acessos à memória do computador, perceberíamos que algumas regiões dela permanecem sem serem acessadas durante períodos de tempo relativamente longos. Por exemplo: se você estiver com várias aplicações disparadas mas usando apenas uma delas, as áreas de memória ocupadas pelas outras não estarão sendo acessadas.

Por causa disso sistemas operacionais como o unix usam o conceito de "memória virtual". Cria-se um espaço de memória fictício bem maior do que o que realmente existe. Por exemplo: se o seu computador possui 2 gigabytes de memória, ele passará a ter, digamos, 4 gigabytes de "memória virtual". Isso significa que a soma de toda a memória requisitada por todos os processos em execução pode ser 4 gigabytes ao invés de 2.

Bem... e os outros 2GB? Serão alocados 2 gigabytes do disco para eles. Assim, num determinado momento, um processo pode estar apenas **parcialmente** residindo na memória. O sistema operacional determina uma estratégia para escolher o que manter na memória física e o que manter no disco. Havendo necessidade, pode-se subitamente enviar ao disco parte do que está na memória ou vice-versa. Esse é um processo especial (de kernel) chamado "swap" ou

“swapper”, e a região do disco alocada é chamada “área de swap”. Essa região no Ubuntu, nas versões anteriores à 17.04, era criada em uma partição física. A partir da versão 17.04, passou-se a permitir o uso de um arquivo texto especial “/swapfile”.

Assim, se você permanecer algum tempo sem usar uma aplicação, poderá acontecer que, ao tentar usá-la novamente, haverá um delay significativo para a sua resposta (isto é... se o seu sistema estiver sobrecarregado, com memória insuficiente). É o sistema operacional trazendo de volta à memória partes desse processo que estavam residindo em disco.

**Exercício 11:** No Linux há um comando que apresenta dados de ocupação da memória e da área de swap (“free” ou “free -h”). Dê uma olhada na man page dele e experimente usá-lo. Descubra se na sua máquina é utilizada uma partição ou um arquivo para implementar a área de swap: para isso, verifique se existe ou não o arquivo “/swapfile”. Não havendo o arquivo, outra opção é usar o comando “swapon -show” (caso você tenha permissão de superusuário, “sudo swapon -show”).

**Exercício 12:** Agora use o comando “top”. Nele você também pode visualizar o estado da memória além do “consumo” de cpu. Você consegue observar o uso de cada *core*...

## Estruturas de controle

Todas as informações que o S.O. precisa para poder controlar a execução do processo (atributos do processo) encontram-se no Bloco de Controle do Processo (ou *Process Control Block*). No Linux, essa estrutura de dados é implementada por meio da *task\_struct*, que é definida no arquivo *sched.h*.

```
struct task_struct {
    unsigned did_exec:1;
    pid_t pid;
    pid_t tgid;
    ...
    char hide;
}
```

**Exercício 13:** Procure e abra o arquivo fonte do *sched.h* no seu Linux. No Ubuntu, ele fica no diretório *include* (por exemplo: `/usr/src/linux-headers-4.15.0-58-generic/include/linux`). Analise a `struct task_struct`.