Sistemas Operacionais – DI/UFES

ROTEIRO LAB ZERO – Processos no Linux

O Conceito de Processo

Nas aulas teóricas vimos o conceito de processo, que pode ser entendido como uma instância de um programa em execução. Em um sistema operacional multitarefa, como o Unix/Linux, múltiplas tarefas podem estar sendo realizadas simultaneamente por diferentes processos, como a formatação de um disco, a impressão de um arquivo e a edição de um texto.

Neste laboratório vamos tratar de alguns comandos de gerenciamento de processos no Linux.

O comando ps

O comando ps permite examinar os processos correntes. Por exemplo:

```
  mypc:~$ ps
   PID TTY          TIME CMD
  6276 pts/0    00:00:00 bash
 12475 pts/0    00:00:00 ps
```

Neste exemplo há um processo (o shell, que neste caso é o bash) com o qual você dialoga a fim de executar comandos. Esse é o processo de número `6276`. O comando solicitado consiste na execução do programa ps. Esse programa, ao ser executado, informa que neste momento há dois processos: ele próprio (de número `12475`) e o shell.

Na verdade, além desses dois pode haver muitos outros que o ps não exibe para não poluir a saída. Por meio das opções ‘a’ e ‘x’, entretanto, pode-se exibir todos os processos correntes:

```
  mypc:~$ ps ax
   PID TTY      STAT   TIME COMMAND
     1 ?        Ss     0:08 /sbin/init splash
     2 ?        S      0:00 [kthreadd]
     3 ?        I<     0:00 [rcu_gp]
     4 ?        I<     0:00 [rcu_par_gp]
    ...
  6265 ?        Rsl    0:14 /usr/lib/gnome-terminal/gnome-terminal-server
  6276 pts/0    Ss     0:00 bash
    ...
 12410 pts/0    R+     0:00 ps ax
```

Nesse caso, todos os processos, além do próprio shell e o ps, são exibidos. Muitos desses processos dizem respeito apenas à administração do sistema.

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

Na coluna **TTY** temos o terminal ou pseudoterminal associado ao processo. Vejam que o nosso shell representa um pseudo terminal (no exemplo, pts/0).

Estudo: Ler o seguinte texto e assistir ao vídeo constante no final do mesmo:

- Comandos ps e pstree – visualizando processos e threads no sistema Linux
  http://www.bosontreinamentos.com.br/linux/certificacao-lpic-1/comandos-ps-e-pstree-visualizando-processos-e-threads-no-sistema-linux/

Exercício 1: Reproduza os exemplos acima de uso do ps, e experimente também usá-lo com a opção "u":. Use "ps u" e depois "ps aux". Qual a diferença que você observou na saída?

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'; para sair do manual, pressione 'q').

Agora olhe novamente a saída do comando "ps aux" e responda: por que a grande maioria dos processos apresenta a letra 'S' na coluna STAT, e pouquíssimos (1 ou 2?!?) a letra 'R'?

Exercício 3: Em geral há um limite superior pequeno para o PID 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 caractere '&' no final do comando que instrui o shell a dispará-lo em "background".

Estudo: Ler o seguinte texto e assistir ao vídeo constante no final do mesmo:

- Controle de Tarefas no shell – comandos bg, fg, jobs:
  http://www.bosontreinamentos.com.br/linux/controle-de-tarefas-no-shell-comandos-bg-fg-jobs/

Exercício 4: Crie um subdiretório, coloque nele vários arquivos e use-o para disparar a compressão em background, como indicado acima. Em seguida, execute outros comandos, por exemplo:

```
mypc:~$ cd
mypc:~$ mkdir lixo
mypc:~$ cd lixo
mypc:~$ cp /bin/* .
mypc:~$ gzip -9 * &
mypc:~$ ps u
(... outros comandos ... o comando abaixo vai apagar seu diretório lixo)
mypc:~$ cd ; rm -rf lixo
```

Prioridade

Num sistema operacional em que vários processos podem estar simultaneamente em execução, surge às vezes a necessidade de 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 Unix atribui a cada processo uma prioridade, que pode ser alterada quando necessário (vamos entender melhor isso quando estudarmos Escalonamento de Processos).

Nesta prática, iremos limitar-nos aos comandos "nice" e "renice". Com eles, pode-se disparar um processo com uma prioridade mais baixa do que a padrão, ou alterá-la (diminuí-la) durante a sua execução, o que significa dizer que o sistema operacional vai privilegiar a execução dos outros processos em detrimento do processo alvo do comando.

O uso típico do "nice" é impedir que um processo demorado atrapalhe o uso interativo do sistema pelo(s) usuário(s) nele logados. Sua sintaxe é:

```
nice -n ajuste comando
```

onde `ajuste` corresponde ao número nice que será adicionado ou subtraído ao processo relativo ao `comando` (programa) a ser executado.

No exemplo de compressão mostrado anteriormente, o disparo por meio do "nice" poderia ser assim feito:

```
mypc:~$ nice -n 15 gzip -9 * &
```

Estudo: Assistir ao seguinte vídeo:

- Prioridades de Processos e comandos nice e renice:
  http://www.bosontreinamentos.com.br/linux/prioridades-de-processos-e-comandos-nice-e-renice/

Exercício 5: De dentro do seu diretório ~/lixo, execute os seguintes comandos (mas antes, esvazie seu diretório lixo e copie novamente os arquivos do /bin para dentro dele):

```
mypc:~$ nice -n 15 gzip -9 * &
mypc:~$ ps l
```

… mas seja rápido… senão o seu processo `gzip` pode terminar antes do seu comando `ps l` !

Você acabou de criar um processo em background usando o comando `nice` e alterando sua prioridade. Em seguida, você listou os processos ("`ps l`") visualizando, entre outros atributos, suas prioridades (colunas `PRI` e `NI`). Observe que o valor de `PRI` para o processo gzip foi 20+15 (o mesmo 15 que você passou no parâmetro `ajuste`), o que resulta em um número maior do que dos outros processos. Como no Linux os valores de prioridades são invertidos (número menor, prioridade maior), com este procedimento, você reduziu a prioridade do `gzip`. Você também poderia ter aumentado a prioridade do processo, passando um valor negativo no parâmetro `ajuste` mas, para isso, você precisaria ser superusuário. Você saberia dizer por que há essa restrição?

Extra: Leia a man page do comando `renice` para saber como alterar a prioridade de um processo após o seu disparo.

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 possui seu conjunto de variáveis de ambiente, que em geral é herdado (copiado) do pai do processo. Uma das variáveis mais conhecidas, comum ao Unix e ao MS-DOS, 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 *MS-DOS* 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
```

Estudo: Assistir ao vídeo abaixo e, em seguida, ler o texto indicado.

- Variáveis de Ambiente no Linux e comandos echo, env e export
  http://www.bosontreinamentos.com.br/linux/prioridades-de-processos-e-comandos-nice-e-renice/
- Como listar as variáveis de ambiente no Linux
  http://www.bosontreinamentos.com.br/linux/como-listar-as-variaveis-de-ambiente-no-linux/

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 exibe 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 -alF`". Defina esse comando no seu "`.bashrc`" (basta adicionar a linha "`alias ll='ls -alF'`").

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?

<u>O que é swap? (OPCIONAL)</u>

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.

Estudo (opcional): Assistir ao vídeo abaixo`;

- Linux básico - Aula 21 - Comando free
  https://www.youtube.com/watch?v=KwozEyv7jOE

Exercício 11 (opcional): 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 (opcional): 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*...

<u>Estruturas de Controle (OPCIONAL)</u>

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 (opcional): 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-4.18.0-24-generic/include/linux`). Analise a struct `task_struct`. Mostre a linha de código da `task_struct` em que é definido o campo usado para armazenar o estado corrente do processo.

RESUMO DA ÓPERA: Por que é importante aprender Linux? (OBRIGATÓRIO)

Assista ao vídeo abaixo e faça as suas próprias considerações da importância de aprender Linux.

https://youtu.be/UsHiWIgxj2M