Sistemas Operacionais – DI/UFES

ROTEIRO LAB ZERO – Processos no Linux

Um pouco de história sobre o UNIX...

https://drive.google.com/file/d/1pwV-A1XtrEPTHJl_GtFrxhlnsnwEGcBO/view?usp=sharing

Conceito de Processo

Em sistemas multitarefa (como o Unix), múltiplas tarefas podem estar sendo realizadas simultaneamente, por exemplo a formatação de um disco, a impressão de um arquivo e a edição de um texto. A cada uma delas corresponde a 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 muito pequena de segundo, congelá-lo (coloca-se ele no estado "pronto" ou "*ready*") e passando-se em seguida a executar um outro processo durante uma outra fração de segundo, e assim por diante. Isso cria a **ilusão** de simultaneidade.

O comando ps

Você precisa antes de tudo de um um shell. O shell é um interpretador de comandos que funciona como um interface entre o usuário e o sistema operacional. Existem vários tipos de Shell: o sh (chamado Bourne shell), o bash (Bourne again shell), o csh (C Shell), o Tcsh (Tenex C shell), etc. Normalmente, seus nomes correspondem ao nome do executável.

Caso você tenha pouca (ou zero!) intimidade com terminais Linux, assista este vídeo: https://youtu.be/mgs92GtkQCE?t=66

Gostaria de saber mais sobre como o conceito de Terminais surgiu?

http://bacana.one/o-que-e-um-tty-no-linux-e-como-usar-o-comando-tty

Vamos lá! Por meio do comando ps pode-se 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
```

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 `6276`. 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 `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 dizem respeito apenas à administração do sistema.

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 (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).

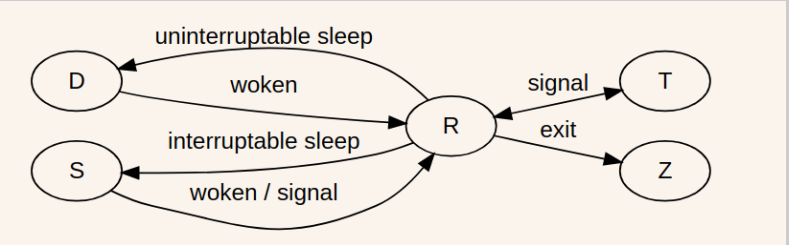
***** TAREFAS *****

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' (também há o estado I, indicando *idle*) na coluna STAT, e pouquíssimos (1 ou 2?!?) a letra 'R'? [RESPONDER NO GOOGLE FORMS]

Diagrama de estados (simplificado) no Linux



PROCESS STATE CODES

```
R running or runnable (on run queue)
D uninterruptible sleep (usually IO)
S interruptible sleep (waiting for an event (not IO) to complete)
Z defunct/zombie, terminated but not reaped by its parent
T stopped, either by a job control signal or because it is being traced
```

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? [RESPONDER NO GOOGLE FORMS]

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".

***** TAREFA *****

Exercício 4: Crie um subdiretório, coloque nele vários arquivos e use-o para disparar a compressão em background como indicado acima e 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 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 Unix atribui a cada processo uma prioridade, que pode ser alterada quando necessário (vamos entender melhor isso quando estudarmos Escalonamento de Processos). Por default, os processos são criados **com a prioridade igual à do seu "pai"** (processo pai é o processo que solicitou ao SO a criação de um novo processo; por exemplo, `bash` é pai dos processos `ps` e `gzip`).

Nesta prática iremos limitar-nos ao comando "nice". Com ele, pode-se disparar um processo com **prioridade mais baixa**, 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" poderia ser assim:

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

***** TAREFA *****

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 --adjustment=15 gzip -9 * &
mypc:~$ ps l
```

… mas seja rápido… senão o seu processo `gzip` pode terminar antes do seu comando `ps l` ! **Outra opção mais fácil** é utilizar o seguinte exemplo com o comando ping (observe que estamos jogando a saída do ping para dentro do arquivo log.txt para não poluir o console):

```
mypc:~$ nice --adjustment=15 ping www.google.com > log.txt &
mypc:~$ 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 (ou ping) foi alterado em relação aos outros processos de usuário do seu sistema (na minha VM Ubuntu tenho 20+**15**! O mesmo 15 passado no parâmetro `--adjustment`). Mas atenção! No Linux, os valores de prioridades são invertidos! Com isso, você **abaixou** a prioridade do `gzip/ping`. Você também poderia ter aumentado a prioridade do processo, passando um valor negativo no `--adjustment`, mas para isso você precisaria ser superusuário. Você saberia dizer por que há essa restrição? [RESPONDER NO GOOGLE FORMS]

> **Extra:** Para saber como alterar a prioridade de um processo após o seu disparo, leia a man page do comando `renice`

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) **em todos os diretórios** relacionados na variável `PATH`.

Frequentemente, torna-se necessário para o usuário lidar com essas variáveis para customizar as aplicações que utiliza. Vejamos um exemplo simples. O comando "man" normalmente 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
```

***** TAREFAS *****

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`" (também é comum definirmos um arquivo `.bashrc` no próprio diretório HOME). 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, ou listar os processos em execução, envolvem o disparo de pelo menos um novo processo (lembra? quando executamos "ps …", o próprio ps é listado como um processo!). No entanto, a troca do diretório corrente (comando "`cd`"), que consiste em mudar a variável de ambiente PWD, não provoca o disparo de processo algum. Você saberia o porquê? [RESPONDER NO GOOGLE FORMS]

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 principal (parte dele na RAM, e parte em disco). 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, com PID=0 e que não é exibido pelo comando ps) 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.

***** TAREFAS *****

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*... (outra opção legal também é instalar o comando "`htop`")

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;
}
```

***** TAREFAS *****

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-4.18.0-24-generic/include/linux`). Analise a struct `task_struct`. Agora cole aqui a linha de código da `task_struct` em que é definido o campo usado para armazenar o estado corrente do processo (normalmente é uma variável long). [RESPONDER NO GOOGLE FORMS]