Aula 14 - Gerenciamento de processos

14.1 - Processos e seus estados

Da teoria de sistemas operacionais sabemos que todos os programas ou aplicativos em execução dentro de um sistema são manipulados através de uma entidade denominada de processo.

O Linux e o Unix manipulam os processos através de várias informações de controle. Dentre elas é possível citar:

- *Estado*
- *Mapa de endereços*
- *Prioridade*
- *Informações sobre os recursos de um processo*
- *Sinais para um processo*
- *Proprietário de um processo*

Essas variáveis e muitas outras, compõem um cenário que chamamos de contexto de um processo. Elas são de suma importância para o bom funcionamento de um processo, pois em sistemas operacionais modernos, o chaveamento de processos em execução, faz necessário com que as informações referentes a cada um desses processos fiquem disponíveis para serem carregadas ou retiradas da CPU a qualquer momento.

Para melhorar a organização dessas informações, os sistemas operacionais, costumam manipular uma estrutura de dados conhecida como bloco de *controle de processos* (*BCP*).

Por ora, é importante entender que as informações do bloco de controle podem ser vistas facilmente no Linux. Isso se deve pela existência do diretório /*proc*, que é um pseudo sistema de arquivos e que serve para armazenar as informações de cada um dos processos do sistema.

Dentro de /*proc* existem vários subdiretórios identificados por numerais. Esses números simbolizam os identificadores dos processos. A respeito dos números e identificadores podemos entendê-los como:

- *PID : identificador do processo.*
- *PPID : identificador do processo pai.*

Portanto, dentro de um sistema /*proc*, iremos encontrar subdiretórios com números de PID. Ainda mais a frente, serão descritas as principais informações de um processo nestes diretórios.

14.2 Comandos e chamadas para manipular processos

Uma forma de gerar um novo processo na memória, em Linux, pode ser feita através de uma chamada de sistema: *fork( )*. Esta função, cria um processo filho a partir do processo pai responsável pela chamada.

Portanto, nessa criação do processo filho, os valores e propriedades atribuídas a este são idênticas às do pai. Caso um processo fique sem pai, o sistema deve se encarregar de atribuir a este processo órfão um pai inicial. No caso padrão, atribuir a herança de qualquer processo ao processo pai de todos, que é o processo *init*. Ele é o primeiro processo executado e de onde

derivam todos outros processos de usuário. Isso pode ser visualizado através da árvore de processos, que é vista com o comando *pstree*

Outros comandos, importantes, utilizados para a verificação e manipulação de processos em Linux são:

- ***ps***: lista os processos existentes conforme os parâmetros de entrada
Ex: ps -ef | grep daemon

- ***top***: monitora os processos e recursos em tempo real
Ex: top -i

- ***vmstat***: verifica a memória virtual e funcionamento do sistema, como por exemplo a troca de contexto entre processos.
Ex: vmstat -n 2

- ***pstree***: mostra a árvores de processos.
Ex: pstree

- ***&:*** quando associado a um comando, envia essa execução para segundo plano.
Ex: tail -f /var/log/messages &

- ***fg***: traz alguma execução para plano principal.
Ex: fg

- ***bg***: leva a execução de um processo para pano de fundo.
Ex: bg

- ***pidof***: exibe o pid de um processo.
Ex: pidof pulseaudio

- ***nice***: executa um processo com uma prioridade pré-definida.
Ex: nice -n -19 runner.sh

- ***renice***: altera a prioridade de um processo para a desejada.
Ex: renice -19 -p 1677

- ***chrt:*** altera parâmetros de processos real time.
Ex: crht -p 85 2467

14.2.1 Sinais

A comunicação entre os processos ou dois processos para com o sistema e o usuário pode ser feita de muitas formas. Este é outro tópico bastante importante no campo da teoria de sistemas operacionais. É aqui que se estudam problemas clássicos como a troca de mensagens, estruturas de semáforos e recursos compartilhados.

Para efeitos práticos, no Linux, serão apresentados os principais sinais que o sistema pode passar a um processo. Esses sinais, são na verdade instruções com finalidades únicas. Cada processo deve ter a capacidade de capturar os sinais e interpretá-los. Quando isso não é possível, então o sistema (kernel) poderá estabelecer uma ação padrão para um sinal não interpretado.

Os sinais mais comuns são:

- HUP (1) : sinal passado para que o processo se reinicie.
- ***SEGV(11)***: sinal passado quando há violação de memória. Em geral causa um core dump (cópia da memória para registro e auditoria)
- KILL(9): destrói o processo.
- ***INT(2):*** interrompe o processo limpando seus registros e retornando ao contexto anterior.

Os comandos utilizados para a passagem de sinais são:

- ***kill***
Ex: *#> kill -9 23*

- ***killall***
Ex*: #> killall -9 apache*

Os sinais são capazes de alterar os estados de um processo. Sabe-se da teoria de sistemas operacionais que os processos possuem estados a fim de que o kernel possa escaloná-los de forma correta e acionar ou liberar os recursos adequadamente.
Esses estados dos processos em Linux são conhecidos por:

- *Executável*: pronto para ser executado ou em vias de.
- *Dormente*: aguardando algum recurso.
- *Zumbi*:tentando se destruir.
- *Parado*: processo suspenso. Sem permissão para continuar.

14.3 Entendendo o /proc

O diretório /proc é quem mantém as informações para os processos. Dessa forma, é possível afirmar que estas informações, armazenadas ali, são os conteúdos do bloco de controle dos processos. A saber, podemos encontrar alguns arquivos importantes:

/proc/PID/cmdline – linha de comando ou nome do processo invocado
/proc/PID/cwd – atalho para o diretório corrente
/proc/PID/environ – variáveis de ambiente do processo
/proc/PID/mem – memoria utilizada pelo processo
/proc/PID/status - estado do processo
/proc/PID/statm – estado da memória do processo

Existem muitos documentos de ajuda e guias dentro do sistema. Para esclarecimento e aprofundamento nas informações de processos, é interessante que se faça consulta aos manuais de *sinais*, *kill*, *fork* entre outros.

14.4 Exercícios

1- Execute o comando ps e exiba somente os processos pertencentes ao usuário root

2- O resultado do comando "*uptime -a"* exibe um valor de *"load average":*? Explique o significado desse valor.

3 – Encontre os processos que estejam no diretório /proc e identifique 3 deles pelos seus nomes de executáveis. Agora informe quantas trocas de contexto ocorreram para esses processos até o presente momento. Informe o passo a passo para chegar a essas conclusões.

4- Qual é a estrutura mantida pelo kernel que também possui grande parte das informações que estão armazenadas no diretório */proc*?

14.5 Atividades

Atividade 1

Controle de prioridade dos processos:

* a) Crie um programa em sua plataforma Linux para que você possa fazer um teste de consumo de CPU. Você pode compilar o código do **exemplo** a seguir e gerar dois executáveis distintos. Por exemplo um dos executáveis pode se chamar ***executacpu*** e o outro ser chamado de ***executacpu2*** (***"#> gcc -o executacpu exmplo.c"***).

```
#include <stdio.h>
int main()
{
        int i;

        while ( 1  )
        {
                i++;
                if(i==10000) {i=0; printf("blah\n");}
        }
        return 1;
}
```

Depois de gerado os dois executáveis, execute cada um deles em dois terminais de console diferentes. Mantenha os dois consoles abertos e rodando ambos os programas. A partir daí, você poderá abrir um terceiro um terminal de console e monitorar ambos processos através do comando ***top*** ou do comando ***ps.***

* Se preferir, você utilizar outros programas, como o aplicativo ***yes,*** para ter um processo que executa em um laço infinito.

b) Dados os dois processos, identifique o PID de ambos e defina o primeiro processo como aquele de PID menor, e por consequência o segundo processo, aquele de PID maior.

c) Agora faça, de minuto em minuto, com que a prioridade do primeiro processo seja alterada, de forma que a mesma se decremente de 5 em 5 enquanto a prioridade para o segundo se incremente de 5 em 5. Tente fazer com que os incrementos ocorram simultaneamente aos decrementos (utilizar o ***renice***).

Utilize o *ps* ou top para obter informações dos processos.

Ex: #> ps -eo comm,pid,nice,time | grep -w "yes"

#> ps -C yes*

#> top -i

d) Trace um gráfico com no mínimo 4 medições de uso de CPU por tempo decorrido (**TIME+**) para os dois processos simultaneamente. Os intervalos de tempo podem ser fixos. Uma sugestão é fazer um *log* de seu monitoramento dos processos . Se você estiver utilizando programas de teste baseados em entrada e saída, note que é importante fazer com que as janelas que rodam esse processos (como é o caso do yes) tenham o foco ativo, caso contrário os programas não consumirão os recursos de maneira efetiva, o que pode invalidar o experimento.

Ex:

cumulativo de CPU (*TIME+) x Tempo (s)*

```
5                    *
4               *
3          *
2
1       *
0       1       2       3       4       5  (tempo min)
```

cumulativo de CPU (*TIME+) x Tempo (s)*

```
5                                   +
4                               +
3                       +
2               +
1       +
0       1       2       3       4       5  (tempo min)
```

Processo 1 - *

Processo 2 - +

e) Verifique o *pid* de um dos processos *teste* e diga porque, dentro do diretório correspondente em ***/proc/[PID_teste]/sched,*** o valor de ***prio*** não é o mesmo que você atribui como o comando *renice*.

f) Termine os dois processos com o comando kill e informe a linha utilizada

g) O gráfico teve uma curva esperada? Discuta o resultado.

Atividade 2

Conhecendo os estados dos processos, especialmente em Linux, é possível afirmar que existem dois estados que são definidos pelo parentesco e pela forma como essa relação é mantida entre processos.
Em particular deve-se lembrar dos estados orfão e zumbi. Esses estados são alcançados respectivamente quando:

- Um pai morre deixando processos filhos em execução. Os filhos em execução agora devem ser adotados por um processo pai qualquer. Nesse caso o processo que faz a adoção é o init.
- Quando um processo filho termina inesperadamente sem que seu pai receba seu retorno, é como se o pai não soubesse da morte do filho e de certa forma o mantivesse vivo. Na verdade, morto vivo, pois o processo já não executa efetivamente nada.

  Se necessário, utilize como referência o texto abaixo:
  http://www.yolinux.com/TUTORIALS/ForkExecProcesses.html

a) Sabendo dessas condições crie um programa em C que através de um processo pai faz a geração de vários filhos (*fork*). Por sua vez, esses filhos deverão morrer sem enviar sinais ao pai, ocasionando a geração de vários processo zumbis (defuntos).

b) Uma vez rodando o programa em C criado no item a, faça a verificação dos processos através do comando *top* e através do comando *ps* . Discuta o resultado da monitoração de ambos os comandos. Verifique e discuta se a saída foi tal como esperada

Atividade 3

É possível realizar o envio de sinais aos processos através do comando *kill.* Em contrapartida, é possível definir ou modificar ações específicas para os processos através da definição de *traps* (TRAP).

a) Sabendo disso, crie um processo (*shell script*) que ao receber um sinal de QUIT, vai enviar uma mensagem ao terminal informando que o mesmo não terá sua execução terminada. Por outro lado, quando o mesmo receber o sinal de INT, o mesmo deve terminar a sua execução e imprimir uma mensagem de despedida na tela.

b) Verifique se o seu sistema possui o comando *signaltest* devidamente instalado. Caso não tenha, realize a instalação do mesmo e o execute algumas vezes. Com isso, discuta como os sinais são recebidos pelos processos. Isto é, eles são processados de maneira uniforme? Existe atraso?