Manipulação de Processos no Linux

Trabalho da disciplina de Sistemas Operacionais

Caio Miglioli, Caio Eduardo Theodoro, Alexandre Scrocaro

Coordenação do Curso de Bacharelado em Ciência da Computação - COCIC
Universidade Tecnológica Federal do Paraná - UTFPR
Campo Mourão, Paraná, Brasil

1 INTRODUÇÃO

O trabalho tem como objetivo entender o funcionamento dos processos no sistema GNU/Linux da distribuição Debian, realizando comandos de verificação dos processos ativos e gerando novos processos através de programas para teste gerado pelos alunos em questão.

2 CONFIGURAÇÕES UTILIZADAS NO PROCEDIMENTO

Hardware:
Memoria RAM: 8 GB
Processador: Intel® Core™ i5-7200U CPU @ 2.50GHz × 4
Sistema 64 bits

Distribuição:
Debian 10.9.0
Kernel compilado: linux-5.12.12

3 PARTE 1: MANIPULAÇÃO DE PROCESSOS

3.1 Execute o comando "ps aux"e identifique três programas do sistema (daemons) e três programas do usuário:

3.1.1 Programas do Usuário.

Na Figura 1 temos os processos relacionados ao programa de usuário **Libre Office Writer**, onde temos as colunas **USER, PID, %CPU, %MEM, VSZ, RSS, TTY, STAT, START, TIME e COMMAND**, respectivamente.

No caso do processo **Libre Office Writer**, temos o valor **'caiom-c+'** para a coluna **USER**, que representa o nome do usuário da máquina que iniciou o processo.
Já a coluna **PID** representa o ID único usado para referenciar o software em questão, seu valor neste caso é **'1536'**.
As colunas **%CPU** e **%MEM** representam a quantidade de recursos do processador e da memória RAM que o processo está utilizando, que neste caso são **'5.3%'** e **'3.6%'**, respectivamente.
As colunas **VSZ** e **RSS** nos diz a quantidade em Kilobytes de memória virtual e memória RAM utilizada no processo, no caso do LB Writer foram utilizadas **'791500KiB'** de VSZ e **'145860KiB'**, respectivamente.
Já a coluna **TTY** nos diz o terminal que iniciou o processo, em nosso caso temos um **'?'** pois não foi utilizado um TTY.
A coluna **STAT** nos diz qual é o estado de execução do programa, no nosso caso temos **'Sl'**, onde o **'S'** significa que o programa está aguardando (Interruptible Sleep) e o **'l'** significa que o programa é multi-thread.
As colunas **START** e **TIME** nos diz o horário em que o programa foi iniciado e o tempo total

que o processador utilizou em sua execução. Neste caso o processo foi iniciado às **'16:55'** e foram gastos **'0:00'** de tempo de uso da CPU, respectivamente.
E por fim temos o **COMMAND**, o comando utilizado para gerar o processo, que foi **'/usr/-lib/libreoffice/program/soffice.bin –writer –splash-pipe=5'**.

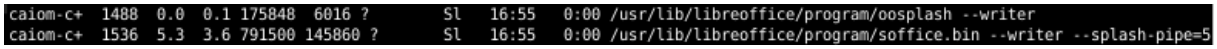

Figura 1 – Programa do Usuário: Libre Office Writer

Podemos ver na figura 2 os processos relacionados ao programa **Mozilla Firefox** e na figura 3 os processos relacionados ao programa de usuário **VSCode**.

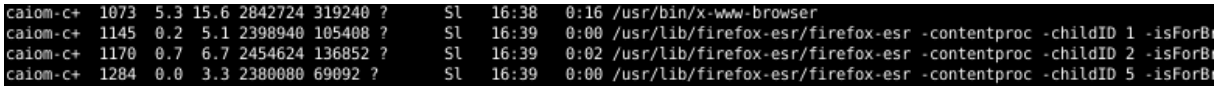

Figura 2 – Programa do Usuário: Mozilla Firefox

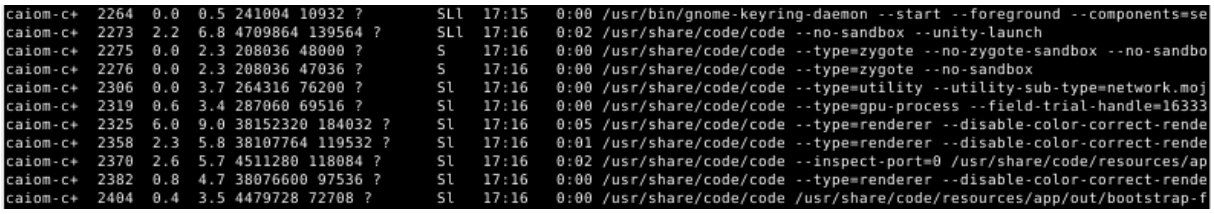

Figura 3 – Programa do Usuário: VSCode

3.1.2 Programas do Sistema (Daemons).

Na Figura 4 temo o processo relacionado ao programa de sistema **Init**, que é o programa responsável por inicializar todo o sistema operacional. Após realizar a inicialização, o software se comporta como um Daemon e o fechamento do processo gera uma finalização em cascata de todos os processos em execução, já que o Init é o processo **Pai** de todos os outros processos.

No caso do processo **Init**, temos o valor **'root'** para a coluna **USER**, que representa o nome do usuário da máquina que iniciou o processo.
Já a coluna **PID** representa o ID único usado para referenciar o software em questão, seu valor neste caso é **'1'**, pois este foi o primeiro processo a ser executado.
As colunas **%CPU** e **%MEM** representam a quantidade de recursos do processador e da memória RAM que o processo está utilizando, que neste caso são **'0.0%'** e **'0.4%'**, respectivamente.
As colunas **VSZ** e **RSS** nos diz a quantidade em Kilobytes de memória virtual e memória RAM utilizada no processo, no caso do Init foram utilizadas **'104028KiB'** de VSZ e **'10168KiB'**, respectivamente.
Já a coluna **TTY** nos diz o terminal que iniciou o processo, em nosso caso temos um **'?'** pois não foi utilizado um TTY.
A coluna **STAT** nos diz qual é o estado de execução do programa, no nosso caso temos **'Ss'**, onde o **'S'** significa que o programa está aguardando (Interruptible Sleep) e o **'s'** significa que o

programa é um iniciador de sessão.
As colunas **START** e **TIME** nos diz o horário em que o programa foi iniciado e o tempo total que o processador utilizou em sua execução. Neste caso o processo foi iniciado às **'16:27'** e foram gastos **'0:01'** de tempo de uso da CPU, respectivamente.
E por fim temos o **COMMAND**, o comando utilizado para gerar o processo, que foi **'/sbin/init'**.

```
root         1  0.0  0.4 104028 10168 ?        Ss   16:27   0:01 /sbin/init
```

Figura 4 – Programa do Sistema: Init

Podemos ver na figura 5 o processo relacionado ao programa **SSH** e na figura 6 o processo relacionado ao programa de sistema **TimeSync**.

```
root       555  0.0  0.3  15856  6332 ?        Ss   16:27   0:00 /usr/sbin/sshd -D
```

Figura 5 – Programa do Sistema: SSH

```
systemd+   419  0.0  0.3  95152  6308 ?        Ssl  16:27   0:00 /lib/systemd/systemd-timesyncd
```

Figura 6 – Programa do Usuário Sistema: TimeSync

3.2 Há processos zombies executando em seu sistema operacional? Posso eliminá-los do sistema usando o comando kill -SIGKILL pid_zombie? Justifique.

Não há processos zumbis executando em nosso sistema operacional, como pode ser observado nas figuras 7 e 8, nas quais estão contidos dois diferentes comandos que, de certo modo, fazem a busca por possíveis processos zumbis executando no sistema.

```
caiom-caiot-alexandre@debian:~$ ps aux | grep 'Z'
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
caiom-c+  3148  0.0  0.0   6116   816 pts/0    S+   18:31   0:00 grep Z
caiom-caiot-alexandre@debian:~$
```

Figura 7 – Procurando por processo zumbi

```
caiom-caiot-alexandre@debian:~$ ps axo stat,ppid,pid,comm | grep -w defunct
caiom-caiot-alexandre@debian:~$
```

Figura 8 – Procurando por processo zumbi

Não. Pois o processo zumbi já está morto, então você não pode matá-lo inserindo seu pid. Porém é possivel excluí-lo matando o processo pai desse zumbi. Depois que o pai morrer, o zumbi será herdado por pid 1, que o aguardará e limpará sua entrada na tabela de processos.

- Para descobrir o PID do zombie usamos
  ps aux | grep -w Z

- Depois usamos o comando abaixo para pegar o PID do processo pai
  ps o ppid PID do zumbi

- E por ultimo matamos o processo pai.
  kill -1 PID do pai

Outro método para matar o processo zumbi é reiniciar a maquina.

3.3 Quais os processos com maior utilização de CPU? Quais os processos com maior utilização de memória? Qual o processo do usuário está a mais tempo em execução?

Utilizando o programa **HTOP**, podemos ordenar todos os processos pelas informações desejadas. Na figura 9 temos o programa que mais estava utilizando CPU no momento da análise, o **'xorg'** responsável pelo gerenciamento de janelas em nosso sistema.

Figura 9 – Processo que mais utiliza a CPU.

Já na figura 10 temos o programa que mais estava utilizando memória naquele momento, que coincidentemente também foi o **'xorg'**.

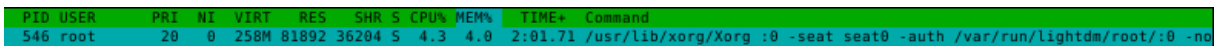

Figura 10 – Processo que mais utiliza a Memória RAM.

E por fim, na figura 11, temos o programa que mais utilizou tempo de processamento durante o período em que o sistema foi ligado até o momento da análise. E mais uma vez temos o **'xorg'**.

Figura 11 – Procurando que foi executado por mais tempo.

3.4 Como eu faço para suspender um processo no Linux? Como eu faço para retomar a execução novamente?

Para uma parada para o processo, podemos enviar tanto o SIGTSTP quanto o SIGSTOP:

- SIGTSTP:
  kill -TSTP [pid]

- SIGSTOP:
  kill -STOP [pid]

A diferença entre eles é que o SIGTSTP é uma requisição de parada que pode ser ignorada, enquanto o SIGSTOP, não. Já para retomar a execução do processo, enviamos o SIGCONT:

- SIGCOUNT:
  kill -CONT [pid]

3.5 O que aconteceria se um processo criasse recursivamente processos filhos indefinidamente? Implemente um programa em Linux que faça isso e apresente o resultado. (Sugestão: testar na máquina virtual).

A criação recursiva de processos filhos indefinidamente levaria o sistema á **inanição**, que basicamente congelaria o sistema, impedindo ele de realizar quaisquer ações. A prática desse problema é conhecida como Fork bomb.

```c
#include <stdio.h>
#include <sys/types.h>

int main()
{
    while(1)
        fork();
    return 0;
}

```

Figura 12 – Fork bomb

Um comando bem famoso também para a realização do fork bomb no bash é o **":(){ :|: & };:"**.

4 CONCLUSÃO

Nessa atividade nós realizamos a manipulação de processos no Linux através dos comandos apresentados, além de criar quatro programas conforme especificado no laboratório 02. Com a realização da atividade, foi possível compreender nos familiarizar ainda mais com o funcionamento dos processos no linux. Também implementamos diferentes códigos, anexados na entrega, que ajudaram no entendimento da parte prática do assunto.

5 REFERÊNCIAS

Sanjay. ps aux command and ps command explained. ComputerNetworkingNotes, 2019. Disponível em: <https://www.computernetworkingnotes.com/linux-tutorials/ps-aux-command-and-ps-command-explained.html>. Acesso em: 27/06/2021.

Fork() Bomb. GeeksForGeeks, 2018. Disponível em: <https://www.geeksforgeeks.org/fork-bomb/>. Acesso em: 27/06/2021.