Juliano Cesar Petini
Michel Gomes de Souza

Manipulação de processos
Laboratório 02

Relatório técnico de atividade prática solicitado pelo professor Rodrigo Campiolo na disciplina de Sistemas Operacionais do Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Universidade Tecnológica Federal do Paraná – UTFPR

Departamento Acadêmico de Computação – DACOM

Bacharelado em Ciência da Computação – BCC

Campo Mourão

Junho / 2021

Resumo

O Presente trabalho tem como objetivo compreender a estrutura de um processos no **Linux** e como podemos manipular e criar novos processos utilizando os recursos como o *Fork* e *Execv* presente na linguagem de programação *C*.

Sumário

1 Introdução

Nesse projeto iremos realizar vários testes no Linux a fim de compreender a fundo o que são processos e como manipulá-los, para isso utilizaremos a linguagem de programação C e uma maquina virtual.

2 Parte 1

Nessa seção iremos apresentar os comandos que foram realizados para contemplar os objetos proposto pela parte 1 da atividade.

2.1 Utilizando o comando 'ps aux' para identificar alguns tipos de processos.

Para os programas *daemons* do sistema temos:

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root       761  0.2  0.5 1053048 47044 ?      Ssl  17:50   0:00 /usr/bin/containerd
root        34  0.0  0.0      0     0 ?        S    17:49   0:00 [kauditd]
root       774  0.0  0.0  72308  5800 ?        Ss   17:50   0:00 /usr/sbin/sshd -D
```

Iremos pegar a ultima linha como exemplo. O usurário responsável pela execução desse programa é o **root** já o seu **PID**(Identificador de Processo) é o **774**.

Esse programa esta ocupando 0.0% da **CPU** e da memória ram, já **VSZ** que o tamanho de memoria que o Linux deu para esse processo utilizar, que no caso é de 72308.

A tag **RSS** é a quantidade de memoria que de fato o programa esta utilizando, que é 5800.

TTY indica qual terminal aquele comando esta rodando, o **?** significa em nenhum.

STAT significa qual é o estado que aquele processo esta, no nosso caso **Ss** significa que esta esperando um evento para completar.

O **START** indica em qual horário o programa iniciou, 17:50 foi o horário iniciado pelo nosso comando.

Ja **COMMAND** é o comando executado pelo usuário **root**.

```
jmallone 3111  0.4  0.1  55744  9332 pts/0   S+   17:53   0:00 vim parte1.txt
jmallone 2666  0.0  0.0  22848  5220 pts/1   Ss   17:51   0:00 bash
jmallone 2187  8.8  0.2 1384656 16996 ?      S<l  17:50   0:17 /usr/bin/pulseaudio
    --start --log-tar
```

Já uma parte do programa dos usuários pode ser visto acima. E nos atentaremos

para a ultima linha também. Seu usuário responsável é o **jmallone**. Já o **PDI** é o de numero 2187. Tal programa esta ocupando 8.8% da **CPU** e 0.2% da memória ram.

O espaço de armazenamento destinado é de 1384656 e o espaço usado é de 16996. Seu estado de processo é **S<l** que significa que esse processo esta esperando um evento para ser completado, tem uma alta prioridade e ele é multi-threaded.

Iniciou sua execução em 17:50 e durou 0:17 segundos. O comando executado foi o */usr/bin/pulseaudio–start –log-tar*.

2.2 Verificando se exitem processos zombies executando no sistema operacional.

Não existe processos do tipo Zombies no sistema operacional, o comando **kill** não irá funcionar pois para sumir essa notificação do *ps aux* devemos matar o processo pai.

2.3 Verificando os processos com maior utilização de CPU, MEMÓRIA e TEMPO DE EXECUÇÃO no sistema operacional.

Para identificarmos o processo com maior uso da **CPU**, utilizamos o comando **ps aux –sort=%cpu**.

```
ps aux --sort=%cpu
USER      PID %CPU %MEM   VSZ   RSS TTY     STAT START  TIME COMMAND
jmallone 1994 64.9 4.3 65756916 345604 ?  Sl   17:50 11:35
   /opt/google/chrome/chrome
```

Com essa saída conseguimos identificar que o processo que no momento esta utilizando a maior quantidade de **CPU** é o processo com **PID 1994** e esta relacionado ao google chrome.

Na identificação do processo com maior uso de memoria novamente utilizamos o comando **ps aux –sort=%men**.

```
ps aux --sort=%mem
USER      PID %CPU %MEM   VSZ   RSS TTY     STAT START  TIME COMMAND
jmallone 1994 67.6 4.7 65756916 376956 ?  Sl   17:50 13:05
   /opt/google/chrome/chrome
```

Na saída resultando conseguimos identificar que o processo com maior uso de memoria no momento é o processo de **PID 1994** e também esta relacionado ao google chrome.

Para finalizar e descobrimos qual processo esta a mais tempo aberto utilizamos novamente o comando **ps aux –sort=time**.

```
ps aux --sort=time
USER       PID %CPU %MEM   VSZ   RSS TTY     STAT START  TIME COMMAND
jmallone 1994  1794 59 17:50 ?      00:09:06 /opt/google/chrome/chrome
    --type=renderer --field-trial-handle=2622820392331351867,1666132771
```

Com a saída resultante conseguimos identificar que novamente o processo com maior tempo de existência no momento e o processo de **PID 1994** no qual novamente esta ligado ao google chrome.

2.4 Como suspender e retomar processos do Linux.

Para suspender: **CTRL+z** Retomar um processo: Verifique os processos com o comando **jobs** e depois digite **fg** Numero_do_processo

2.5 Processo criando filhos recursivamente e indefinidamente.

Para realizar o experimento proposto pela atividade, utilizamos uma maquina virtual criada no *Virtual-Box* com o sistema *Debian* sem interface gráfica. Inicialmente criamos uma algoritmo simples que podemos ver na Figura 1, com esse algoritmo simulamos a criação recursivamente e indefinidamente de processos filhos, na Figura 2, conseguimos ver o *loop* gerado pelo algoritmo. Uma curiosidade que foi observada ao realizar esse experimento é que, ao entrar em *loop* não foi possível sair dele utilizando quaisquer comandos básicos do Linux como **CTRL+X** ou **CTRL+Z**, nenhum deles respondiam ao serem pressionados e a única maneira que achamos para finalizar o *loop*, foi reiniciar a maquina virtual.

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

int cria_filho(){
        pid_t pid = fork();
        if (pid){
                printf("Sou o pai");
        }
        else{
                printf("Filho criado!");
                cria_filho();
        }
}

int main(){
        cria_filho();
}
```

Figura 1 – Criando recursivamente filhos indefinidamente.

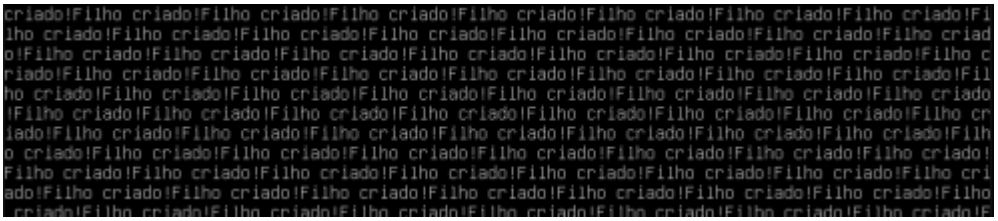

Figura 2 – Criando recursivamente filhos indefinidamente.

3 Conclusões

Nesta atividade foi apresentado o básico sobre processos e suas funcionalidades. Conhecer essas características irá tornar as *skills* de um programador em diversas áreas do conhecimento mais acuradas e logicas. Já em sua segunda parte que consta nos arquivos *.zip* a sua implementação se mostrou de fato útil para fixar os conhecimentos obtidos.