Universidade Federal de Uberlândia
Disciplina: Sistemas Operacionais (SO)
Prática: Processos em Linux

Processos LINUX (GABARITO)

Parte 1: Processos em Linux

1. Atividades em Ambiente LINUX

1.1 – A prioridade é exibida pela coluna **pr**.

1.2 – Com o uso do comando kill, foi possível finalizar o processo **4984**. Após o uso do comando, o mesmo já não aparece com o comando **top**.

```
    PID USUARIO   PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TEMPO+ COMANDO
   5895 user1     20   0 2794188 350516 183008 R 116,3  17,2   0:09.65 Web Content
   4948 user1     20   0 3133728 285816 138160 S  50,2  14,0   2:45.48 firefox
```

```
user1@ABS:~$ kill 4984
user1@ABS:~$
```

```
    PID USUARIO   PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TEMPO+ COMANDO
   1587 root      20   0 1556060 239356  25096 S   3,3  11,8   5:57.31 Xorg
   1854 user1     20   0  155588   1000   1000 S   1,7   0,0   0:28.66 VBoxClient
   1922 user1     20   0 1270184  19200  12136 S   1,7   0,9   1:18.92 xfwm4
   2992 user1     20   0  389892  25644  16792 S   1,0   1,3   0:11.11 xfce4-terminal
   5884 root      20   0       0      0      0 I   1,0   0,0   0:00.74 kworker/u8:0-flush-8
   5935 root      20   0       0      0      0 I   1,0   0,0   0:00.11 kworker/u8:3-events_
   6187 user1     20   0  234848  29492  23720 S   1,0   1,4   0:00.85 xfce4-screensho
   1759 user1     20   0    8004   3548   2472 S   0,7   0,2   0:04.41 dbus-daemon
   1956 user1     20   0  511864  17624  12264 S   0,7   0,9   0:10.09 panel-10-pulsea
     10 root      20   0       0      0      0 I   0,3   0,0   2:27.80 rcu_sched
```

1.3 – Neste diretório é possível obter informações como o grupo a qual o processo pertence, onde está montado, mapeamento na memória e muito mais.

```
user1@ABS:~$ cat /proc/6589/
arch_status      cmdline          exe              loginuid
attr/            comm             fd/              map_files/
autogroup        coredump_filter  fdinfo/          maps
auxv             cpuset           gid_map          mem
cgroup           cwd/             io               mountinfo
clear_refs       environ          limits           mounts
```

2. Praticando Comandos

2.1.a – O código realizou 41 chamadas. A chamada **getpid** retorna o **ID** de um processo e a chamada **getppid** retorna o **ID** do processo pai. 21 syscalls

```
Eu fui criado pelo processo 9960
Eu sou o pai 10248 e eu vou criar um filho
Eu sou o processo filho 0 e vou executar o comando ls

aaa  exec1  exec2  exec2_1  exec3  exec3_1  exec4  exec5

Processo Filho Completo
```

2.1.b – O código não foi finalizado com o **return 0,** pois o processo filho nunca finaliza ao se usar o **for(;;)**.
Ao cancelar a execução do código (**ctrl + c** no terminal), o **strace** nos informa que o código fez 36 chamadas de sistema.

```
Eu sou o pai 8583 e eu vou criar um filho
        Oi, eu sou o processo 8584, o filho. Meu pai é 8583
        O dia esta otimo hoje, nao acha?
        Bom, desse jeito vou acabar me instalando para sempre
        Ou melhor, assim espero!
```

Com o uso do **sleep(1)**, o processo finaliza e o programa se encerra com **return 0**. Com o código executado até o final, 40 chamadas de sistema são realizadas.

```
user1@ABS:~/Downloads/Adriano-Material/laboratorio2/Lab2_Processo$ ./exec2_1
Eu sou o pai 8865 e eu vou criar um filho
        Oi, eu sou o processo 8866, o filho. Meu pai é 8865
        O dia esta otimo hoje, nao acha?
        Bom, desse jeito vou acabar me instalando para sempre
        Ou melhor, assim espero!
As luzes comecaram a se apagar para mim, 8865
Minha hora chegou : adeus, 8866, meu filho
```

2.1.c – O processo pai tem o **value = 20** e o processo filho tem **value = 35**. Eles possuem valores diferentes porque, ao se utilizar o **fork()** para criar um processo filho, uma cópia do programa executado também é criada. Esse processo filho é criado com um **PID = 0** e, por isso, a instrução **if** o identifica como filho. Já o processo pai possui um **PID** maior do que zero (o PID do sistema operacional), o qual é identificado como um pai no nosso **else if.**

```
 filho: 8969
Valor-filho: 35

 Pai: 8968
Valor-pai: 20
```

2.1.d – A função **adjustX** serve para aumentar o valor do filho em 1 e reduzir o do pai em 1. Essas alterações ocorrem em intervalos de tempo aleatórios. A saída do programa exibe continuamente as atualizações nos valores para os processos pai e filho. No comando **ps xl**, o PID dos processos pai e filho podem ser vistos na terceira e quarta coluna. Neste exemplo, o processo pai tem **PID = 10656** e o filho tem **PID = 10657**.

```
0  1002   10565    1735  39  19 629828 19088 poll_s SNl  ?          0:00 /usr/lib/
0  1002   10629    9955  20   0  10612  5024 do_wai Ss   pts/1      0:00 bash
0  1002   10656    9960  20   0   2488   720 -      R+   pts/0      0:03 ./exec4
1  1002   10657   10656  20   0   2488    88 -      R+   pts/0      0:03 ./exec4
4  1002   10658   10629  20   0  11508  3276 -      R+   pts/1      0:00 ps xl
```

Após usar a chamada kill e finalizar o processo filho, apenas o código do processo pai continua em execução.

```
filho: 366
pai: -267
filho: 367
pai: -268
filho: 368
pai: -269
filho: 369
pai: -270
pai: -271
pai: -272
pai: -273
pai: -274
pai: -275
pai: -276
pai: -277
```

Ao finalizar o processo pai, o código encerra sua execução.

```
pai: -520
pai: -521
pai: -522
Morto
user1@ABS:~/Downlo
```

Ao finalizar o processo pai, o processo filho continua sua execução normalmente. Não há diferença na ordem em que eles são finalizados.

2.1.e – Ao executar o código, o processo pai é criado, após isso, o código cria 7 processos filhos.

```
user1@ABS:~/Do
PID = 11353
PID = 11354
PID = 11352
PID = 11357
PID = 11358
PID = 11355
PID = 11356
PID = 11359
```

Ao modificar o código, apenas 5 processos são criados.

```
user1@ABS:~/Down
PID = 11522
PID = 11523
PID = 11525
PID = 11524
PID = 11526
user1@ABS:~/Down
```

2.1.f – A variável global é incrementada no processo filho e essa alteração é vista no processo pai. Isso ocorre pois o processo filho utiliza os mesmos registros, arquivos abertos e contadores de programas que o processo pai.

```
Eu fui criado pelo processo 11662
Eu sou o pai 11668 e eu vou criar um filho
Antes de usar o vfork
Eu sou o processo filho 11669

PID Pai = 11668, Variavel Global = 7
```

3. Programação

O programa irá exibir um valor para a variável **value** para o processo-pai com valor igual a 5 e para o processo-filho com valor da variável **value** igual a 20. Isso ocorre devido ao que o processo filho ter uma cópia que é modificada e não afeta o pai.