

Breno Farias da Silva

Felipe Archanjo da Cunha Mendes

Pamella Lissa Sato Tamura

Thaynara Ribeiro Falcão dos Santos

## **Laboratório 02: Manipulação de processos**

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

Março / 2022

# Sumário

1	Introdução . . . . .	3
2	Objetivos . . . . .	3
3	Fundamentação . . . . .	3
4	Materiais . . . . .	3
5	Procedimentos e Resultados . . . . .	4
	5.1 Parte 1: Manipulação de Processos . . . . .	4
	5.2 Parte 2: Programação . . . . .	10
6	Conclusões . . . . .	14
7	Referências . . . . .	14

## 1 Introdução

O relatório está dividido em duas partes, a primeira reservada para a manipulação de processos além de ser composta por perguntas e respostas. Já a segunda parte, está relacionada à programação da qual engloba o desenvolvimento de um programa que cria uma hierarquia de processos com N níveis (5), entre outros programas que envolvam a criação de processos de pais e filhos.

## 2 Objetivos

O objetivo do trabalho consiste em, primeiramente, aprimorar os conhecimentos estudados em sala de aula sobre a funcionalidade de alguns comandos Linux para a manipulação de processos. Além de construir programas que desenvolvam determinadas soluções envolvendo processos.

## 3 Fundamentação

As perguntas e respostas foram fundamentadas pelas aulas e materiais fornecidos pelo próprio Prof. Dr. Rodrigo Campiolo, assim como o desenvolvimento do programa e códigos descritos na parte dois do documento. Além disso, as definições de comandos e conceitos que foram obtidos baseadas em aulas e, parcialmente, por pesquisas sequenciais por cada membro da equipe.

## 4 Materiais

- Debian 11 LTS (ISO)
- VirtualBox 6.1.32
- Debian GNU/Linux 11 (Bullseye)
- Ryzen 7 3800x
- 8GB de RAM

## 5 Procedimentos e Resultados

### 5.1 Parte 1: Manipulação de Processos

1. Execute o comando `ps aux` (Figura 1) e identifique três programas do sistema (daemons)(Figura 2) e três programas do usuário, explicando os valores cada uma das colunas para um de cada tipo (sistema e usuário).

Daemons: accounts-daemon, rtkit-daemon, chroot helper.

Usuário: firefox-esr, gnome-shell, pipewire.

- PID: Identificador do processo. “Process ID”.
- USER: Indica qual usuário iniciou o processo.
- PRI: Prioridade do processo
- NI: “Nice” do processo, o qual é o nível amigável do processo que está relacionado com a sua prioridade.
- VIRT: A quantidade de memória virtual usada pelo processo.
- RES: A memória física que não é swap que uma tarefa usou. (Resident Size)
- SHR: A quantidade de memória compartilhada usada pelo processo.
- S ou STAT: Estado do processo.
- CPU%: Tempo de CPU.
- MEM%: O percentual de memória física usada pelo processo.
- Time: Tempo em que o processo está ativo.
- Command: Nome do programa.

```

brenofarias@breno:~$ ps aux
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.1 164100 10284 ?        Ss   18:04   0:00 /sbin/init
root           2  0.0  0.0  0  0 ?        S    18:04   0:00 [kthreadd]
root           3  0.0  0.0  0  0 ?        I<   18:04   0:00 [rcu_gp]
root           4  0.0  0.0  0  0 ?        I<   18:04   0:00 [rcu_par_gp]
root           6  0.0  0.0  0  0 ?        I<   18:04   0:00 [kworker/0:0H]
root           8  0.0  0.0  0  0 ?        I    18:04   0:00 [kworker/u16:0]
root           9  0.0  0.0  0  0 ?        I<   18:04   0:00 [mm_percpu_wq]
root          10  0.0  0.0  0  0 ?        S    18:04   0:00 [rcu_tasks_kt]
root          11  0.0  0.0  0  0 ?        S    18:04   0:00 [rcu_tasks_ru]
root          12  0.0  0.0  0  0 ?        S    18:04   0:00 [rcu_tasks_tr]
root          13  0.0  0.0  0  0 ?        S    18:04   0:00 [ksoftirqd/0]
root          14  0.0  0.0  0  0 ?        I    18:04   0:01 [rcu_preempt]
root          15  0.0  0.0  0  0 ?        S    18:04   0:00 [migration/0]
root          16  0.0  0.0  0  0 ?        S    18:04   0:00 [cpuhp/0]
root          17  0.0  0.0  0  0 ?        S    18:04   0:00 [cpuhp/1]
root          18  0.0  0.0  0  0 ?        S    18:04   0:00 [migration/1]
root          19  0.0  0.0  0  0 ?        S    18:04   0:00 [ksoftirqd/1]
root          21  0.0  0.0  0  0 ?        I<   18:04   0:00 [kworker/1:0H]
root          22  0.0  0.0  0  0 ?        S    18:04   0:00 [cpuhp/2]
root          23  0.0  0.0  0  0 ?        S    18:04   0:00 [migration/2]
root          24  0.0  0.0  0  0 ?        S    18:04   0:00 [ksoftirqd/2]
root          26  0.0  0.0  0  0 ?        I<   18:04   0:00 [kworker/2:0H]
root          27  0.0  0.0  0  0 ?        S    18:04   0:00 [cpuhp/3]
root          28  0.0  0.0  0  0 ?        S    18:04   0:00 [migration/3]
root          29  0.0  0.0  0  0 ?        S    18:04   0:00 [ksoftirqd/3]
root          31  0.0  0.0  0  0 ?        I<   18:04   0:00 [kworker/3:0H]
root          32  0.0  0.0  0  0 ?        S    18:04   0:00 [cpuhp/4]
root          33  0.0  0.0  0  0 ?        S    18:04   0:00 [migration/4]
root          34  0.0  0.0  0  0 ?        S    18:04   0:00 [ksoftirqd/4]
root          36  0.0  0.0  0  0 ?        I<   18:04   0:00 [kworker/4:0H]
root          37  0.0  0.0  0  0 ?        S    18:04   0:00 [cpuhp/5]
root          38  0.0  0.0  0  0 ?        S    18:04   0:00 [migration/5]
root          39  0.0  0.0  0  0 ?        S    18:04   0:00 [ksoftirqd/5]
root          41  0.0  0.0  0  0 ?        I<   18:04   0:00 [kworker/5:0H]
root          42  0.0  0.0  0  0 ?        S    18:04   0:00 [cpuhp/6]
root          43  0.0  0.0  0  0 ?        S    18:04   0:00 [migration/6]

```

Figura 1 – Demonstração do comando px aux

```

0% 0.0% 4[| 1.3%
1[| 1.3% 5[||||| 8.5%
2[| 2.0% 6[ 0.0%
3[| 0.7% 7[ 0.7%
Mem| 2.456/7.676 Tasks: 113, 761 thr: 1 running
Swp| 0K/7.456K Load average: 0.14 0.42 0.50
Uptime: 00:29:20

  PID USER   PRI  NI  VIRT   RES   SHR  S  CPU% MEM%   TIME+  Command
 516 root    20   0  231M  9480  6712 S   0.0  0.1  0:00.07 /usr/libexec/accounts-daemon
 518 avahi    20   0  7272  3528  3176 S   0.0  0.0  0:00.02 avahi-daemon: running [breno.local]
 520 messagebu 20   0  9784  5932  3900 S   0.0  0.1  0:00.75 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-acti
 521 root    20   0  249M 19648 14796 S   0.0  0.2  0:00.33 /usr/sbin/NetworkManager --no-daemon
 544 avahi    20   0  7092  348  0 S   0.0  0.0  0:00.00 avahi-daemon: chroot helper
 546 root    20   0  231M  9480  6712 S   0.0  0.1  0:00.04 /usr/libexec/accounts-daemon
 573 root    20   0  231M  9480  6712 S   0.0  0.1  0:00.01 /usr/libexec/accounts-daemon
 576 root    20   0  249M 19648 14796 S   0.0  0.2  0:00.03 /usr/sbin/NetworkManager --no-daemon
 577 root    20   0  249M 19648 14796 S   0.0  0.2  0:00.12 /usr/sbin/NetworkManager --no-daemon
 655 rtkit    21   1 150M  2936  2684 S   0.0  0.0  0:00.04 /usr/libexec/rtkit-daemon
 656 rtkit    20   0 150M  2936  2684 S   0.0  0.0  0:00.01 /usr/libexec/rtkit-daemon
 657 rtkit    RT   1 150M  2936  2684 S   0.0  0.0  0:00.00 /usr/libexec/rtkit-daemon
1082 brenofari 9 -11 2409M 30640 21660 S   0.0  0.4  0:01.20 /usr/bin/pulseaudio --daemonize=no --log-target=journal
1089 brenofari 20   0  9724  6084  3940 S   0.0  0.1  0:00.69 /usr/bin/dbus-daemon --session --address=systemd: --nofork --nopidfile --systemd-acti
1093 brenofari 20   0  232M  7740  6800 S   0.0  0.1  0:00.10 /usr/bin/gnome-keyring-daemon --daemonize --login
1094 brenofari 20   0  232M  7740  6800 S   0.0  0.1  0:00.00 /usr/bin/gnome-keyring-daemon --daemonize --login
1097 brenofari 20   0  232M  7740  6800 S   0.0  0.1  0:00.05 /usr/bin/gnome-keyring-daemon --daemonize --login
1154 brenofari 20   0  537M 40904 32428 S   0.0  0.5  0:00.05 /usr/libexec/goa-daemon
1167 brenofari 20   0  537M 40904 32428 S   0.0  0.5  0:00.00 /usr/libexec/goa-daemon
1170 brenofari 20   0  537M 40904 32428 S   0.0  0.5  0:00.00 /usr/libexec/goa-daemon
1174 brenofari 20   0  537M 40904 32428 S   0.0  0.5  0:00.00 /usr/libexec/goa-daemon
1187 brenofari -6   0 2409M 30640 21660 S   0.7  0.4  0:01.02 /usr/bin/pulseaudio --daemonize=no --log-target=journal
1190 brenofari -6   0 2409M 30640 21660 S   0.0  0.4  0:00.02 /usr/bin/pulseaudio --daemonize=no --log-target=journal
1228 brenofari 20   0  232M  7740  6800 S   0.0  0.1  0:00.00 /usr/bin/gnome-keyring-daemon --daemonize --login
1243 brenofari 20   0  8172  4380  3904 S   0.0  0.1  0:00.04 /usr/bin/dbus-daemon --config-file=/usr/share/defaults/at-spi2/accessibility.conf --
1522 brenofari 20   0 304M 11752 7060 S   0.0  0.1  0:00.59 ibus-daemon --panel disable -r --xim
1527 brenofari 20   0 304M 11752 7060 S   0.0  0.1  0:00.00 ibus-daemon --panel disable -r --xim

```

Figura 2 – Demonstração do comando htop usando filtro

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 Zombies. Não é possível criar um processo zumbi usando o comando: `kill -SIGKILL pid_zombie` pois os processos zumbis já estão mortos. Para resolver o problema, é necessário executar o comando: `kill -s SIGCHLD ppid`, sendo "ppid" o "parent\_id", ou seja, o pid do processo pai. (Figura 3)

```

top - 18:52:25 up 48 min, 1 user, load average: 0.44, 0.52, 0.54
Tarefas: 238 total, 1 em exec., 237 dormindo, 0 parado, 0 zumbi
%Cpu(s): 2.4 us, 0.5 sy, 0.4 ni, 96.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MB mem : 7859.2 total, 3608.5 livre, 2971.3 usados, 1279.4 buff/cache
MB swap: 7628.0 total, 7628.0 livre, 0.0 usados, 4525.1 mem dispon.

  PID USUARIO PR NI VIRT RES SHR S %CPU %MEM TEMPO+ COMANDO
 1235 brenofa+ 20 0 5855120 372444 142644 S 18.9 4.6 10:35.84 gnome-shell
 4487 brenofa+ 20 0 790804 76656 48004 S 3.3 0.9 0:02.97 nautilus
 5491 brenofa+ 39 19 1283808 42164 25804 S 3.3 0.5 0:00.10 tracker-extract
 5484 brenofa+ 20 0 433152 28444 15408 S 2.3 0.4 0:00.07 tracker-store
 1082 brenofa+ 9 -11 2467544 38808 21828 S 0.7 0.4 0:06.44 pulseaudio
 2002 brenofa+ 20 0 4389216 843944 224156 S 0.7 10.5 10:00.73 firefox-esr
 524 root 20 0 236020 9948 6564 S 0.3 0.1 0:01.08 polkitd
 1099 brenofa+ 20 0 237232 7748 6768 S 0.3 0.1 0:00.06 gvfsd
 1360 brenofa+ 20 0 1194644 32092 24124 S 0.3 0.4 0:00.18 gsd-media-keys
 1548 brenofa+ 20 0 347480 27672 18756 S 0.3 0.3 0:01.02 ibus-extension-
 2148 brenofa+ 20 0 3359200 637684 151068 S 0.3 7.9 2:36.69 Web Content
 2152 brenofa+ 20 0 2827820 208532 113732 S 0.3 2.6 0:32.41 Web Content
 3021 brenofa+ 20 0 1123560 105084 41840 S 0.3 1.3 0:09.24 evince
 3095 brenofa+ 20 0 2852144 219240 132900 S 0.3 2.7 0:19.71 Web Content
 3130 brenofa+ 20 0 2968264 261008 135132 S 0.3 3.2 0:35.80 Web Content
 4173 brenofa+ 20 0 405488 50464 39692 S 0.3 0.6 0:06.46 gnome-terminal-
 1 root 20 0 164100 10324 7632 S 0.0 0.1 0:00.71 systemd
 2 root 20 0 0 0 0 S 0.0 0.0 0:00.00 kthreadd
 3 root 0 -20 0 0 0 I 0.0 0.0 0:00.00 rcu_gp
 4 root 0 -20 0 0 0 I 0.0 0.0 0:00.00 rcu_par_gp
 6 root 0 -20 0 0 0 I 0.0 0.0 0:00.00 kworker/0:0H-events_highpri
 8 root 20 0 0 0 0 I 0.0 0.0 0:01.23 kworker/u16:0-writeback
 9 root 0 -20 0 0 0 I 0.0 0.0 0:00.00 mm_percpu_wq
 10 root 20 0 0 0 0 S 0.0 0.0 0:00.00 rcu_tasks_kthre
 11 root 20 0 0 0 0 S 0.0 0.0 0:00.00 rcu_tasks_rude
 12 root 20 0 0 0 0 S 0.0 0.0 0:00.00 rcu_tasks_trace
 13 root 20 0 0 0 0 S 0.0 0.0 0:00.04 ksoftirqd/0
 14 root 20 0 0 0 0 I 0.0 0.0 0:02.12 rcu_preempt
 15 root rt 0 0 0 0 S 0.0 0.0 0:00.01 migration/0
 16 root 20 0 0 0 0 S 0.0 0.0 0:00.00 cpuhp/0
 17 root 20 0 0 0 0 S 0.0 0.0 0:00.00 cpuhp/1

```

Figura 3 – Demonstração do comando `kill -SIGKILL pid_zombie`

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?

- Maior utilização de CPU: Firefox

`ps -eo pid,cmd,%mem,%cpu,time -sort=-%cpu` (Figura 4)

- Maior utilização de Memória: Firefox

`ps -eo pid,cmd,%mem,%cpu,time -sort=-%mem` (Figura 5)

- Maior executando a mais tempo: Firefox

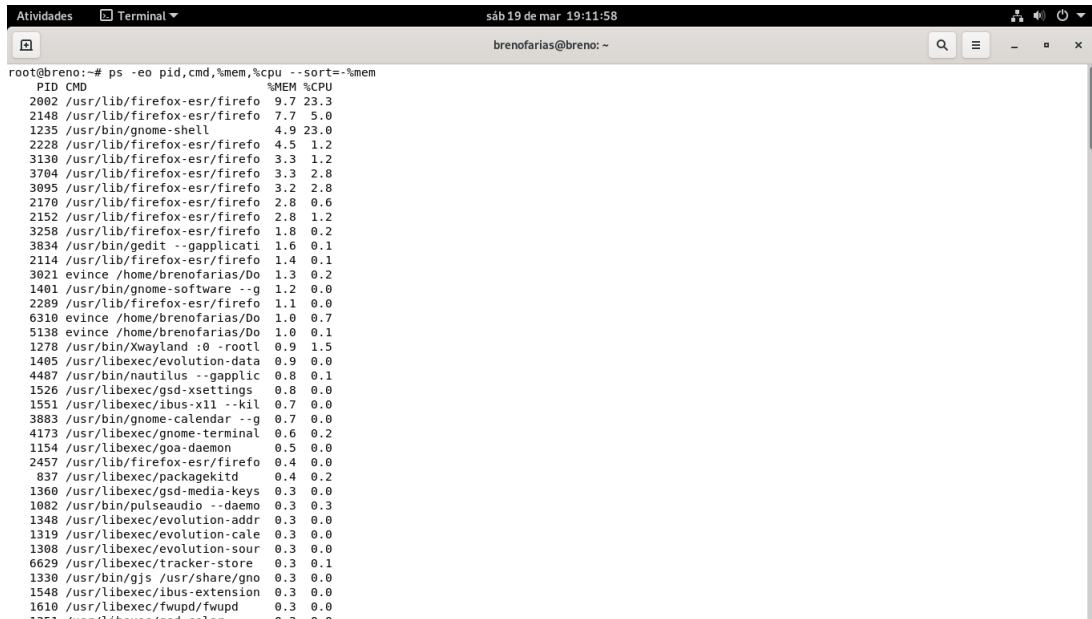
`ps -eo pid,cmd,%mem,%cpu,time -sort=-time` (Figura 6)

```

root@breno:~# ps -eo pid,cmd,%mem,%cpu --sort=-%cpu
  PID CMD                                %MEM %CPU
 2002 /usr/lib/firefox-esr/firefo        9.8 23.5
 1235 /usr/bin/gnome-shell                4.8 23.2
 2148 /usr/lib/firefox-esr/firefo        7.8  5.0
 3095 /usr/lib/firefox-esr/firefo        3.2  2.8
 3704 /usr/lib/firefox-esr/firefo        3.3  2.8
 1278 /usr/bin/Xwayland :0 -rootl        0.9  1.5
 2152 /usr/lib/firefox-esr/firefo        2.8  1.2
 2228 /usr/lib/firefox-esr/firefo        4.5  1.2
 3130 /usr/lib/firefox-esr/firefo        3.2  1.2
 6310 evince /home/brenofarias/Do        1.0  0.8
 2170 /usr/lib/firefox-esr/firefo        2.8  0.6
 1082 /usr/bin/pulseaudio --daemon      0.3  0.3
  837 /usr/libexec/packagekitd          0.4  0.2
 3021 evince /home/brenofarias/Do        1.3  0.2
 3258 /usr/lib/firefox-esr/firefo        1.8  0.2
 4173 /usr/libexec/gnome-terminal        0.6  0.2
 2114 /usr/lib/firefox-esr/firefo        1.4  0.1
 3834 /usr/bin/gedit --gapplicati        1.6  0.1
 4487 /usr/bin/nautilus --gapplic        0.8  0.1
 5138 evince /home/brenofarias/Do        1.0  0.1
    1 /sbin/init                        0.1  0.0
    2 [kthreadd]                          0.0  0.0
    3 [rcu_gp]                            0.0  0.0
    4 [rcu_par_gp]                       0.0  0.0
    6 [kworker/0:0H-events_highpri]      0.0  0.0
    9 [mm_percpu_wq]                     0.0  0.0
   10 [rcu_tasks_kthre]                  0.0  0.0
   11 [rcu_tasks_rude_]                  0.0  0.0
   12 [rcu_tasks_trace]                  0.0  0.0
   13 [ksoftirqd/0]                      0.0  0.0
   14 [rcu_preempt]                      0.0  0.0
   15 [migration/0]                      0.0  0.0
   16 [cpuhp/0]                          0.0  0.0
   17 [cpuhp/1]                          0.0  0.0
   18 [migration/1]                      0.0  0.0
   19 [ksoftirqd/1]                      0.0  0.0
   21 [rcu_tasks_cpu_migrate_kthre]      0.0  0.0

```

Figura 4 – Demonstração do comando `ps -eo pid,cmd,%mem,%cpu,time -sort=-%cpu`



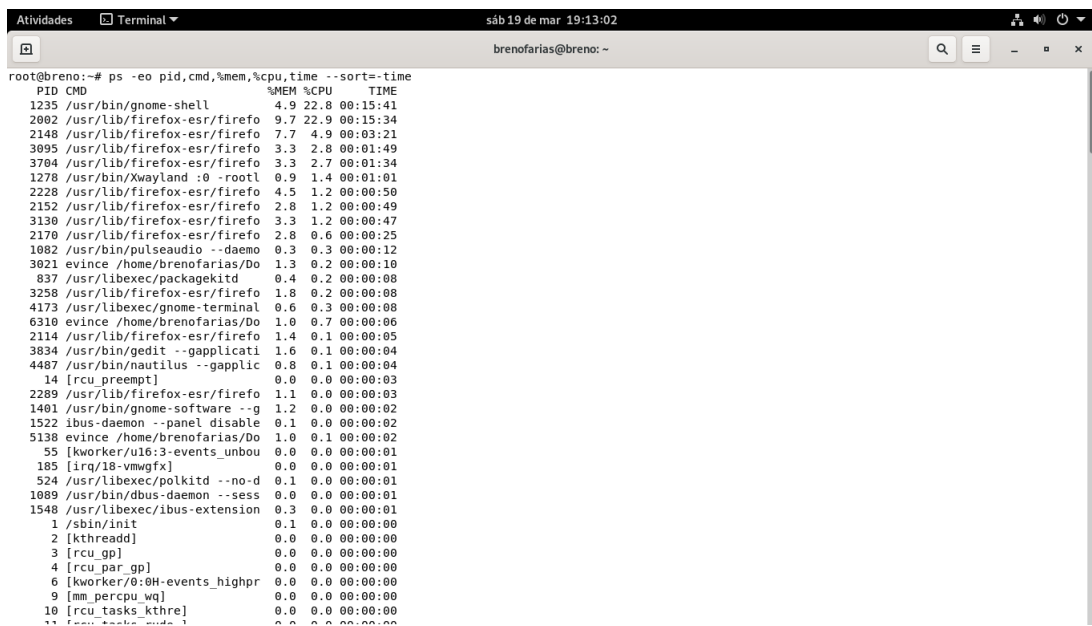
```

Atividades Terminal sáb 19 de mar 19:11:58
brenofarias@breno: ~

root@breno:~# ps -eo pid,cmd,%mem,%cpu --sort=-%mem
PID CMD %MEM %CPU
2002 /usr/lib/firefox-esr/firefo 9.7 23.3
2148 /usr/lib/firefox-esr/firefo 7.7 5.0
1235 /usr/bin/gnome-shell 4.9 23.0
2228 /usr/lib/firefox-esr/firefo 4.5 1.2
3130 /usr/lib/firefox-esr/firefo 3.3 1.2
3704 /usr/lib/firefox-esr/firefo 3.3 2.8
3095 /usr/lib/firefox-esr/firefo 3.2 2.8
2170 /usr/lib/firefox-esr/firefo 2.8 0.6
2152 /usr/lib/firefox-esr/firefo 2.8 1.2
3258 /usr/lib/firefox-esr/firefo 1.8 0.2
3834 /usr/bin/gedit --gapplicati 1.6 0.1
2114 /usr/lib/firefox-esr/firefo 1.4 0.1
3021 evince /home/brenofarias/Do 1.3 0.2
1401 /usr/bin/gnome-software --g 1.2 0.0
2289 /usr/lib/firefox-esr/firefo 1.1 0.0
6310 evince /home/brenofarias/Do 1.0 0.7
5138 evince /home/brenofarias/Do 1.0 0.1
1278 /usr/bin/Xwayland :0 -rootl 0.9 1.5
1405 /usr/libexec/evolution-data 0.9 0.0
4487 /usr/bin/nautilus --gapplic 0.8 0.1
1526 /usr/libexec/gsd-xsettings 0.8 0.0
1551 /usr/libexec/ibus-x11 --kil 0.7 0.0
3883 /usr/bin/gnome-calendar --g 0.7 0.0
4173 /usr/libexec/gnome-terminal 0.6 0.2
1154 /usr/libexec/goa-daemon 0.5 0.0
2457 /usr/lib/firefox-esr/firefo 0.4 0.0
837 /usr/libexec/packagekitd 0.4 0.2
1360 /usr/libexec/gsd-media-keys 0.3 0.0
1082 /usr/bin/pulseaudio --daemo 0.3 0.3
1348 /usr/libexec/evolution-addr 0.3 0.0
1319 /usr/libexec/evolution-cale 0.3 0.0
1308 /usr/libexec/evolution-sour 0.3 0.0
6629 /usr/libexec/tracker-store 0.3 0.1
1330 /usr/bin/gjs /usr/share/gno 0.3 0.0
1548 /usr/libexec/ibus-extension 0.3 0.0
1610 /usr/libexec/fwupd/fwupd 0.3 0.0
1235 /usr/bin/gnome-shell 0.3 0.0

```

Figura 5 – Demonstração do comando `ps -eo pid,cmd,%mem,%cpu,time --sort=-%mem`



```

Atividades Terminal sáb 19 de mar 19:13:02
brenofarias@breno: ~

root@breno:~# ps -eo pid,cmd,%mem,%cpu,time --sort=-time
PID CMD %MEM %CPU TIME
1235 /usr/bin/gnome-shell 4.9 22.8 00:15:41
2002 /usr/lib/firefox-esr/firefo 9.7 22.9 00:15:34
2148 /usr/lib/firefox-esr/firefo 7.7 4.9 00:03:21
3095 /usr/lib/firefox-esr/firefo 3.3 2.8 00:01:49
3704 /usr/lib/firefox-esr/firefo 3.3 2.7 00:01:34
1278 /usr/bin/Xwayland :0 -rootl 0.9 1.4 00:01:01
2228 /usr/lib/firefox-esr/firefo 4.5 1.2 00:00:50
2152 /usr/lib/firefox-esr/firefo 2.8 1.2 00:00:49
3130 /usr/lib/firefox-esr/firefo 3.3 1.2 00:00:47
2170 /usr/lib/firefox-esr/firefo 2.8 0.6 00:00:25
1082 /usr/bin/pulseaudio --daemo 0.3 0.3 00:00:12
3021 evince /home/brenofarias/Do 1.3 0.2 00:00:10
837 /usr/libexec/packagekitd 0.4 0.2 00:00:08
3258 /usr/lib/firefox-esr/firefo 1.8 0.2 00:00:08
4173 /usr/libexec/gnome-terminal 0.6 0.3 00:00:08
6310 evince /home/brenofarias/Do 1.0 0.7 00:00:06
2114 /usr/lib/firefox-esr/firefo 1.4 0.1 00:00:05
3834 /usr/bin/gedit --gapplicati 1.6 0.1 00:00:04
4487 /usr/bin/nautilus --gapplic 0.8 0.1 00:00:04
14 [rcu_preempt] 0.0 0.0 00:00:03
2289 /usr/lib/firefox-esr/firefo 1.1 0.0 00:00:03
1401 /usr/bin/gnome-software --g 1.2 0.0 00:00:02
1522 ibus-daemon --panel disable 0.1 0.0 00:00:02
5138 evince /home/brenofarias/Do 1.0 0.1 00:00:02
55 [kworker/u16:3-events_unbou 0.0 0.0 00:00:01
185 [irq/18-vmmgfx] 0.0 0.0 00:00:01
524 /usr/libexec/polkitd --no-d 0.1 0.0 00:00:01
1089 /usr/bin/dbus-daemon --sess 0.0 0.0 00:00:01
1548 /usr/libexec/ibus-extension 0.3 0.0 00:00:01
1 /sbin/init 0.1 0.0 00:00:00
2 [kthreadd] 0.0 0.0 00:00:00
3 [rcu_gp] 0.0 0.0 00:00:00
4 [rcu_par_gp] 0.0 0.0 00:00:00
6 [kworker/0:0H-events_highpr 0.0 0.0 00:00:00
9 [mm_percpu_wq] 0.0 0.0 00:00:00
10 [rcu_tasks_kthre] 0.0 0.0 00:00:00
11 [rcu_tasks_rude] 0.0 0.0 00:00:00

```

Figura 6 – Demonstração do comando `ps -eo pid,cmd,%mem,%cpu,time --sort=-time`



#### 4. Como eu faço para suspender um processo no Linux? Como eu faço para retomar a execução novamente?

- Para suspender um processo, basta usar o comando:

```
kill -STOP process_id
```

- Para continuar um processo, basta usar o comando:

```
kill -CONT process_id , utilizado para continuar o processo
```

#### 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).

Ao criar um programa que gera processos filhos indefinidamente, o que aconteceria seria que, como cada processo vai continuar a execução de forma independente, cada filho criaria mais filhos. Cada filho estaria usando seu próprio espaço de memória. Eventualmente o computador iria ficar sem recursos suficientes para manter o processo e ele iria ser finalizado. (Figura 7)

```
#include <stdio.h> // printf()
#include <stdlib.h> // exit()
#include <stdbool.h> // true
#include <unistd.h> // fork()
#include <sys/types.h> // pid_t

int main(){
    pid_t pid;

    pid = fork(); // cria um processo e devolve pid do filho para o pai e 0 para o filho

    if (pid){
        /* este trecho é executado pelo pai */
        printf("Eu Sou o Processo Pai - Filho PID %d \n", pid);
    } else{
        /* este trecho é executado pelo filho */
        printf("Eu Sou o Processo Filho - Filho PID %d \n", pid);
        while (true){
            pid = fork( );
        }
    }

    return 0;
}
```

Figura 7 – Demonstração do código fork indeterminado



2. Faça um programa que receba um comando Linux como parâmetro e execute como um filho do processo. O processo pai deve aguardar o término da execução do comando.

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

int main(){
    pid_t pid;

    char *comando = "ls";           // comando a ser executado
    char *argumentos[] = {"ls", "-l", NULL}; // argumentos do comando

    pid = fork();                   // cria um processo filho

    if(pid == 0){                   // processo filho
        execvp(comando, argumentos); // executa o comando
    } else{                         // processo pai
        wait(NULL);                 // espera o processo filho
    }

    printf("Fim do processo pai!\n");
    return 0;
}
```

Figura 10 – Código do programa criando hierarquia de processos de 5 níveis

```
felipolis@DESKTOP-7JJSSPL:~/Parte02$ gcc ex2.c -o ex2
felipolis@DESKTOP-7JJSSPL:~/Parte02$ ./ex2
total 116
-rwxr-xr-x 1 felipolis felipolis 16784 Mar 20 17:12 ex1
-rw-r--r-- 1 felipolis felipolis 419 Mar 20 17:13 ex1.c
-rw-r--r-- 1 felipolis felipolis 573 Mar 20 09:56 ex11.c
-rw-r--r-- 1 felipolis felipolis 726 Mar 20 10:02 ex111.c
-rwxr-xr-x 1 felipolis felipolis 16776 Mar 20 15:44 ex1111
-rw-r--r-- 1 felipolis felipolis 593 Mar 20 15:47 ex1111.c
-rwxr-xr-x 1 felipolis felipolis 16864 Mar 20 17:19 ex2
-rw-r--r-- 1 felipolis felipolis 717 Mar 20 17:17 ex2.c
-rw-r--r-- 1 felipolis felipolis 2670 Mar 20 13:03 ex3.c
-rw-r--r-- 1 felipolis felipolis 2995 Mar 20 12:12 ex3gab.c
-rwxr-xr-x 1 felipolis felipolis 17224 Mar 20 17:06 ex4
-rw-r--r-- 1 felipolis felipolis 3177 Mar 20 17:08 ex4.c
-rw-r--r-- 1 felipolis felipolis 1505 Mar 20 00:36 teste.c
Fim do processo pai!
felipolis@DESKTOP-7JJSSPL:~/Parte02$
```

Figura 11 – Execução do programa enunciado acima

3. Faça um programa que receba um vetor e divida para N filhos partes iguais de processamento para localizar um item. Exibir o PID dos filhos que encontrarem o valor procurado.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/wait.h>

#define VALOR_BUSCADO 23
#define N_FILHOS 5
#define N_VETOR 30

int* create_random_array(int size){
    int* array = (int*) malloc(sizeof(int) * size);
    int i;
    for(i = 0; i < size; i++){
        array[i] = rand() % size;
    }
    return array;
}

void print_array(int* array, int size){
    printf("\n");
    int i;
    for(i = 0; i < size; i++){
        printf("%d ", array[i]);
    }
    printf("\n");
}

int main(){
    //Criação do vetor com valores aleatórios
    int* vet = create_random_array(N_VETOR);
    print_array(vet, N_VETOR);

    //Cálculo da parte do vetor recebida pelos filhos
    int quant = N_VETOR / N_FILHOS;
    int qnt_aux = N_VETOR % N_FILHOS;

    //Inicialização dos filhos
    pid_t pid, pldret;
    int status = 0, proc = VALOR_BUSCADO;
    int i = 0, j = 0, x = 0;
    int aux = 0;

    pid = getpid();

    for (i = 0; i < N_FILHOS & pid != 0; i++){
        pid = fork();

        if(pid == 0){
            if(pid == 0){
                //O filho recebe sua parte do vetor
                if(i < qnt_aux){
                    aux = quant + 1;
                } else{
                    aux = quant;
                }

                //Cálculo da posição em que o vetor deve começar
                if(i < qnt_aux){
                    x = 1 + (quant);
                } else{
                    x = 1 * aux + qnt_aux;
                }

                //Passagem na respectiva parte do filho
                for(j = 0; j < x & x < aux; j++){
                    if(vet[j] == proc){
                        printf("O valor %d foi encontrado na posicao %d do vetor\n", proc, j);
                        return j;
                    }
                }
                return 0;
            }
        } else{
            printf("Erro ao criar o processo filho\n");
            exit(0);
        }
    }

    //Verifica qual filho encontrou o valor procurado
    if(pid){
        x = 0;

        do{
            pldret = wait(&status);
            if(WEXITSTATUS(status) != 0){
                printf("O filho %d encontrou o valor procurado\n", pldret);
            }
            x++;
        } while(x < N_FILHOS);
    }

    return 0;
}
```

Figura 12 – Execução do programa enunciado acima

```
felipolis@DESKTOP-7JJSSPL:~/Parte02$ gcc ex3.c -o ex3
felipolis@DESKTOP-7JJSSPL:~/Parte02$ ./ex3
[13 16 27 25 23 25 16 12 9 1 2 7 20 19 23 16 0 6 22 16 11 8 27 9 2 20 2 13 7 25 ]
0 valor 23 foi encontrado na posicao 4 do vetor
0 valor 23 foi encontrado na posicao 14 do vetor
0 filho 7199 encontrou o valor procurado
0 filho 7201 encontrou o valor procurado
felipolis@DESKTOP-7JJSSPL:~/Parte02$
```

Figura 13 – Execução do programa recebendo um vetor e dividindo para N filhos

4. Faça uma interface de shell simples que fornece um prompt ao usuário para executar comandos do shell do sistema. Se o comando for executado em segundo plano (&), a interface deve possibilitar a execução de outros comandos. Caso contrário, a interface deve esperar o retorno do comando e, em seguida, exibir o prompt novamente.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <wordexp.h>
#include <sys/wait.h>

#define MAX_LINE_LENGTH 100

int main(){
    char* params[MAX_LINE_LENGTH/2 + 1]; //Vetor de parâmetros
    char cmd[MAX_LINE_LENGTH];           //Vetor de comando
    int saida= 0;                         //Variável de controle de saída
    int espera = 1;                      //Variável de controle de espera
    int i;                                //Variável de controle de laço
    int status;                           //Variável de controle de status
    wordexp_t aux;                        //Variável auxiliar

    do{
        printf("shell> ");                //Prompt
        fflush(stdout);                    //Limpa o buffer de saída
        fgets(cmd, MAX_LINE_LENGTH, stdin); //Recebe o comando

        if(strcmp(cmd, "exit") == 0){      //Verifica se o comando é exit
            saida = 1;
        } else{
            if(cmd[strlen(cmd)-2] == '&'){ //Caso não seja exit
                cmd[strlen(cmd)-2] = '\0'; //Verifica se o comando é em segundo plano
                espera = 0;                //Remove o & do comando
            } else{
                cmd[strlen(cmd)-1] = '\0'; //Não espera o retorno do comando
            }

            wordexp(cmd, &aux, 0);         //Expande o comando
            for(i = 0; i < aux.we_wordc; i++){ //Percorre o vetor de parâmetros
                params[i] = aux.we_wordv[i]; //Copia o parâmetro para o vetor de parâmetros
            }
            params[i] = NULL;              //Finaliza o vetor de parâmetros

            pid_t pid = fork();             //Cria um processo filho
            if(pid == 0){
                execvp(params[0], params); //Executa o comando
                exit(0);                   //Finaliza o processo filho
            }

            if(espera == 1){                //Verifica se o comando é em segundo plano
                waitpid(pid, &status, 0);  //Espera o retorno do comando
            } else{
                espera = 1;                 //Reinicia a variável de controle de espera
            }
        }
    } while(!saida);

    return 0;
}
```

Figura 14 – Execução do programa enunciado acima

```
felipolis@DESKTOP-7JJSSPL:~/Parte02$ gcc ex4.c -o ex4
felipolis@DESKTOP-7JJSSPL:~/Parte02$ ./ex4
shell> ls
ex1.c  ex2.c  ex3.c  ex4  ex4.c
shell> pstree
init──init──init──bash──ex4──pstree
    │   │   │
    │   │   └──init──init──bash
    │   └──5*[{init}]
shell> _
```

Figura 15 – Execução do shell

**OBS.** Segue anexo, no mesmo diretório desse documento, todos os códigos-fonte referente aos exercícios da parte 2.

## 6 Conclusões

Após realizados os procedimentos solicitados na descrição da parte 1 e 2 da atividade, finalizou-se a prática do laboratório 2.

## 7 Referências

- <http://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media%5Bsocm:socm-livro.pdf>
- [https://moodle.utfpr.edu.br/pluginfile.php/688550/mod\\_resource/content/2/aula-2.2.pdf](https://moodle.utfpr.edu.br/pluginfile.php/688550/mod_resource/content/2/aula-2.2.pdf)