

João Vitor Moraski Lunkes

## **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

Bacharelado em Ciência da Computação – BCC

Campo Mourão

Abril / 2025

# Sumário

1	Introdução . . . . .	3
2	Parte 1 - Manipulação de processos . . . . .	4
2.1	Pergunta 01 . . . . .	4
2.2	Pergunta 02 . . . . .	6
2.3	Pergunta 03 . . . . .	6
2.4	Pergunta 04 . . . . .	6
2.5	Pergunta 05 . . . . .	7
3	Parte 2 - Programação . . . . .	8
3.1	Atividade 01 . . . . .	8
3.2	Atividade 02 . . . . .	9
3.3	Atividade 03 . . . . .	9
3.4	Atividade 04 . . . . .	10
4	Conclusão . . . . .	11
5	Referências . . . . .	12
	<b>Referências . . . . .</b>	<b>12</b>

## 1 Introdução

Saber sobre processos é importante no estudo de Sistemas Operacionais, pois são eles que representam as tarefas que estão sendo executadas, os mesmos podem ser vistos como caixas de algoritmos executadas para realizar tarefas, uma abstração de um programa.

A manipulação de processos é uma tarefa pouco utilizada por usuários comuns, uma vez que sistemas operacionais têm se tornado cada vez mais *User friendly*, o que de certa forma facilita a experiência do usuário, mas que também prejudica o controle sobre os processos.

Sistemas operacionais Windows, por exemplo, abstraem muitos dos comandos por meio de interfaces gráficas, o seu gerenciador de tarefas mostra informações resumidas, somente na aba de detalhes que são mostradas informações como PID, memória utilizada, etc.

Ter a noção de como funcionam os processos e de seus principais comandos pode ser crucial para ter um maior controle sobre o computador e evitar lentidões. Os sistemas atuais conseguem manusear razoavelmente bem seus processos, porém se formos olhar para alguns SOs um pouco mais antigos, podemos notar que com o tempo teremos diversos processos inúteis rodando em segundo plano, que a longo prazo poderá ocasionar em problemas com velocidade de processamento. Tal problema pode ser resolvido caso o usuário saiba sobre processos, pois assim ele poderá verificar e encerrar processos indesejados, o que libera memória e processamento. A seguir veremos alguns comandos e como funcionam os processos, na prática.

## 2 Parte 1 - Manipulação de processos

### 2.1 Pergunta 01

Execute o comando **ps aux** e identifique três processos do sistema (*daemons*) e três processos do usuário. Explique os valores da saída desse comando para um dos processos identificados.

```

root      24065  0.0  0.0 251580 10812 ?        Ssl  17:56   0:00 gdm-session-worker [pam/gdm-password]
moraski   24073  0.0  0.0 620612 10808 ?        Ssl+ 17:56   0:00 /usr/bin/gnome-keyring-daemon --foreground
root      24089  0.0  0.0      0 0 ?        I    17:56   0:00 [kworker/9:1-events]
moraski   24095  0.0  0.0 244712  6056 tty2    Ssl+ 17:56   0:00 /usr/libexec/gdm-x-session --run-script env
moraski   24105  3.5  0.2 25720628 170632 tty2    Ssl+ 17:56   1:02 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /
moraski   24122  0.0  0.0  39128 11668 ?        Ss   17:56   0:00 /snap/snapd-desktop-integration/253/usr/bin
root      24136  0.0  0.0      0 0 ?        I    17:56   0:00 [kworker/5:0-cgroup_bpf_destroy]
moraski   24181  0.0  0.0 439140 56160 ?        Ssl  17:56   0:00 /snap/snapd-desktop-integration/253/usr/bin
moraski   24238  0.0  0.0 307208 16952 tty2    Ssl+ 17:56   0:00 /usr/libexec/gnome-session-binary --session
moraski   24294  0.0  0.0 382992  8020 ?        Ssl  17:56   0:00 /usr/libexec/at-spi-bus-launcher
moraski   24301  0.0  0.0   9612 5044 ?        S    17:56   0:00 /usr/bin/dbus-daemon --config-file=/usr/sha
moraski   24307  0.0  0.0 100216 5620 ?        Ssl  17:56   0:00 /usr/libexec/gnome-session-ctl --monitor
moraski   24323  0.0  0.0 323280  8172 ?        Ssl  17:56   0:00 /usr/libexec/gvfsd
moraski   24331  0.0  0.0 468832  7396 ?        Ssl  17:56   0:00 /usr/libexec/gvfsd-fuse /run/user/1000/gvfs
moraski   24334  0.0  0.0 530444 19172 ?        Ssl  17:56   0:00 /usr/libexec/gnome-session-binary --systemd
moraski   24378  3.6  0.7 4642420 493152 ?        Ssl  17:56   1:03 /usr/bin/gnome-shell
root      24393  0.0  0.0      0 0 ?        S    17:57   0:00 [nvidia-drm/timeline-24]
moraski   24395  0.3  0.2 1106440 158432 ?        Ssl  17:57   0:06 /usr/libexec/mutter-x11-frames

```

Figura 1 – Execução do comando ps aux na máquina

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	24065	0.0	0.0	251580	10812	?	Ssl	17:56	0:00	gdm-session-worker [pam/gdm-password]
root	24089	0.0	0.0	0	0	?	I	17:56	0:00	[kworker/9:1-events]
root	24393	0.0	0.0	0	0	?	S	17:57	0:01	[nvidia-drm/timeline-24]
moraski	24073	0.0	0.0	244712	6056	tty2	Ssl+	17:56	0:0	/usr/libexec/gdm-x-session --run-script env
moraski	24378	3.6	0.7	4642420	493152	?	Ssl	17:56	1:03	/usr/bin/gnome-shell
moraski	24323	0.0	0.0	323280	8172	?	Ssl	17:56	0:00	/usr/libexec/gvfsd

Campos:

- USER: Usuário dono do processo
  - root: Processo iniciado pelo sistema.
  - moraski: Processo iniciado pelo usuário.
- PID: ID do processo
  - ID do processo de acordo com a ordem de inicialização.
  - Para o 4º processo: 24378
- %CPU: Porcentagem da CPU utilizada pelo processo
  - Quanto maior a porcentagem, maior o uso de memória. No caso, o uso do 4º processo foi 3.6%.
- %MEM: Porcentagem da memória que o processo está utilizando

- Quanto maior a porcentagem, maior o uso de memória. No caso, o uso do 4º processo foi 0.7%.
- VSZ: Tamanho virtual do processo (KiB)
  - Tamanho em KiB dos processos, temos que o 4º processo teve 493152kib.
- RSS: Total de memória utilizada pelo processo
  - Memória física utilizada pelo processo, temos que o 4º processo teve 4642420kib.
- TTY: Terminal de controle do processo
  - Indica se o comando está sendo gerado por algum terminal, sendo ? um programa não gerenciado por terminal.
- STAT: Estado atual do processo
  - S = Suspenso
  - < = Maior prioridade
  - R = Executável
  - s = Processo principal
  - l contém páginas bloqueadas pelo kernel
  - s é líder de sessão
  - Neste caso, o processo escolhido esta em espera, é líder de sessão e tem paginas de memórias travadas
- START: Horário de início do processos
- No nosso processo escolhido, foi 17:56
- TIME: Tempo de uso da CPU pelo processo até agora
- COMMAND: Comando que executa o processo
  - Nesse caso, `/usr/bin/gnome-shell` é o núcleo da interface gráfica do GNOME

## 2.2 Pergunta 02

Há processos **zombies** executando em seu sistema operacional? Posso eliminá-los do sistema usando o comando **kill -SIGKILL pid\_zombie**? Justifique.

**Resposta:** No momento de execução do **ps aux** não havia nenhum processo **zombie**. Contudo, se houvesse, não é possível eliminá-lo com o comando **kill -SIGKILL pid\_zombie**, pois tal processo já se encontra no estado finalizado. Para eliminar um processo **zombie**, devemos primeiro identificar qual é seu processo pai e em seguida executar o comando **kill -s SIGCHLD pid\_pai**, que vai executar o comando **wait()** do processo e limpar os processos **zombies**.

## 2.3 Pergunta 03

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

PID	%CPU	STAT	COMMAND
34208	13.2	S	chrome

- Maior utilização de memória

PID	%MEM	STAT	COMMAND
31448	4.0	S	/snap/clion/332/bin/clion

- Processo do usuário com maior tempo de execução

PID	TIME	STAT	COMMAND
24378	1:33:45	S	/usr/bin/gnome-shell

## 2.4 Pergunta 04

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

**Resposta:** Para suspender um processo no Linux podemos usar tanto o atalho CTRL + Z, já para retomar o processo basta usar o comando **jobs** para listar os processos parados, verificar o ID do processo e usar **fg [id]** para executar em primeiro plano ou **bg [id]** para segundo plano.

## 2.5 Pergunta 05

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).

**Resposta:** Devido ao processo pai sempre esperar a execução do filho, o mesmo demoraria muito para executar, além de que com o tempo iria necessitar de uma grande quantidade de poder de processamento para que todos os processos fossem executados sem travamento.

```
1 #include <stdio.h>
2 #include <sys/types.h>
3
4 int main() {
5     while(1) fork();
6     return 0;
7 }
```

Código para gerar um fork bomb([WIKIPEDIA, 2025](#)) em qualquer SO que possua uma unix shell

```
1 :(){ :|:& };;:
```

## 3 Parte 2 - Programação

Todos os códigos desta atividade podem ser encontrados aqui no [github](#)

### 3.1 Atividade 01

Faça um programa que crie uma hierarquia de processos com N níveis ( $1 + 2 + 4 + 8 + \dots + 2^{N-1}$ ) processos. Visualize a hierarquia usando um comando do sistema (pstree).

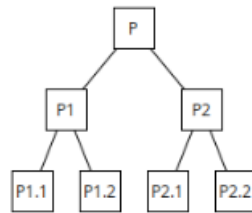


Figura 2 – Exemplo com N = 3

```
make run01
moraski@moraski-B550M-AORUS-ELITE ~/faculdade/sistemas-operacionais/lab02/codigos <master>
$ make run01
gcc ativ01.c -o ativ
./ativ 3
Pid raiz = 187675
█
```

Figura 3 – Exemplo da execução da árvore com 3 níveis

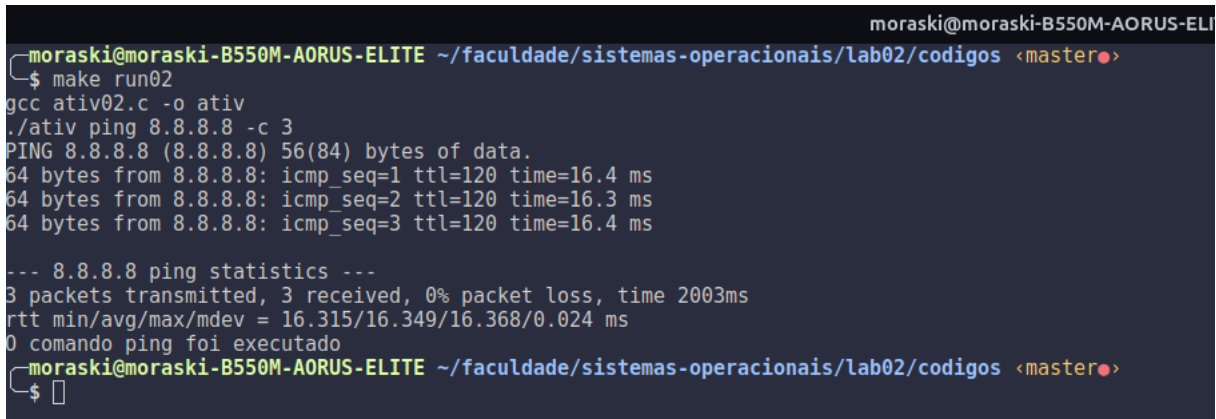
```
moraski@moraski-B550M-AORUS-ELITE:~
$ pstree -p 187675
ativ(187675)─ativ(187676)─ativ(187678)
               │          │          │
               │          │          └─ativ(187680)
               │          └─ativ(187677)─ativ(187679)
               │                          │
               │                          └─ativ(187681)
moraski@moraski-B550M-AORUS-ELITE ~
$ █
```

Figura 4 – Exemplo da hierarquia com 3 níveis



### 3.2 Atividade 02

Faça um programa que receba como entrada um comando (p. ex. `ls`, `ping`, ...) via terminal e execute como seu filho. O processo pai deve aguardar o término da execução do comando.

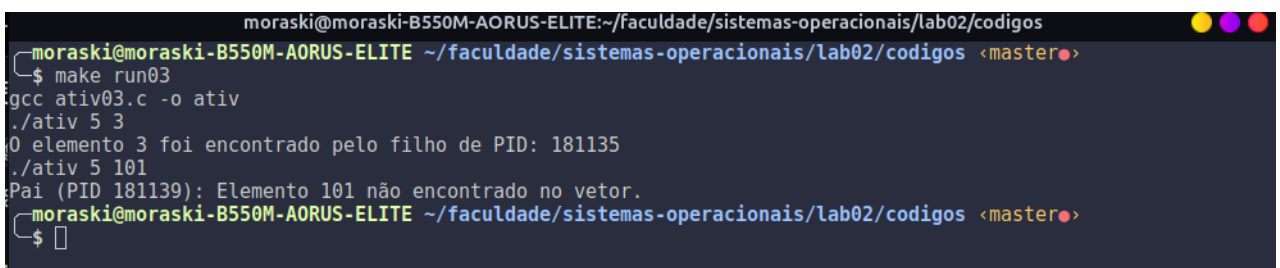


```
moraski@moraski-B550M-AORUS-ELITE ~/faculdade/sistemas-operacionais/lab02/codigos <master>  
$ make run02  
gcc ativ02.c -o ativ  
./ativ ping 8.8.8.8 -c 3  
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.  
64 bytes from 8.8.8.8: icmp_seq=1 ttl=120 time=16.4 ms  
64 bytes from 8.8.8.8: icmp_seq=2 ttl=120 time=16.3 ms  
64 bytes from 8.8.8.8: icmp_seq=3 ttl=120 time=16.4 ms  
  
--- 8.8.8.8 ping statistics ---  
3 packets transmitted, 3 received, 0% packet loss, time 2003ms  
rtt min/avg/max/mdev = 16.315/16.349/16.368/0.024 ms  
0 comando ping foi executado  
moraski@moraski-B550M-AORUS-ELITE ~/faculdade/sistemas-operacionais/lab02/codigos <master>  
$
```

Figura 5 – Exemplo da execução do *ping*, mostrando o *print* no final de qual comando foi executado

### 3.3 Atividade 03

Faça um programa que processe um vetor e divida para N filhos partes iguais de processamento para localizar um item. Cada filho deve exibir seu PID e a posição que o item foi localizado. Um mesmo filho pode encontrar o item em mais de uma posição. Se nenhum filho localizar, o pai deve exibir a mensagem não encontrado. Sugestão: gerar o vetor com valores aleatórios e mostrar na tela antes da busca.

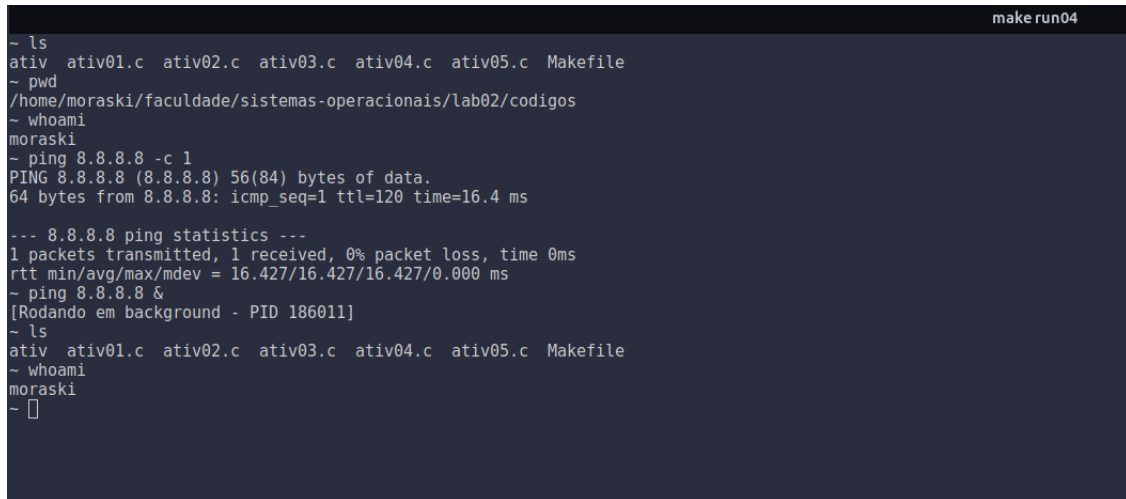


```
moraski@moraski-B550M-AORUS-ELITE ~/faculdade/sistemas-operacionais/lab02/codigos  
moraski@moraski-B550M-AORUS-ELITE ~/faculdade/sistemas-operacionais/lab02/codigos <master>  
$ make run03  
gcc ativ03.c -o ativ  
./ativ 5 3  
0 elemento 3 foi encontrado pelo filho de PID: 181135  
./ativ 5 101  
Pai (PID 181139): Elemento 101 não encontrado no vetor.  
moraski@moraski-B550M-AORUS-ELITE ~/faculdade/sistemas-operacionais/lab02/codigos <master>  
$
```

Figura 6 – Execução da atividade 03

### 3.4 Atividade 04

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.



```
make run04
~ ls
ativ ativ01.c ativ02.c ativ03.c ativ04.c ativ05.c Makefile
~ pwd
/home/moraski/faculdade/sistemas-operacionais/lab02/codigos
~ whoami
moraski
~ ping 8.8.8.8 -c 1
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=120 time=16.4 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 16.427/16.427/16.427/0.000 ms
~ ping 8.8.8.8 &
[Rodando em background - PID 186011]
~ ls
ativ ativ01.c ativ02.c ativ03.c ativ04.c ativ05.c Makefile
~ whoami
moraski
~ □
```

Figura 7 – Exemplo da execução da *shell*

## 4 Conclusão

Através da atividade de laboratório aqui apresentada, podemos verificar na prática os conceitos já vistos em aula, onde é possível compreender a importância de processos, seu manuseio e também visualizar como processos podem inundar o sistema e acabar com os recursos disponíveis rapidamente, como um fork bomb([WIKIPEDIA, 2025](#)), que cria processos indefinidamente até que não haja mais recursos.

A manipulação de processos é importante, pois possibilita o controle mais preciso dos mesmos, já o entendimento dos conceitos é necessário em locais em que há poucos recursos disponíveis ou em cenários onde existem muitos processos zumbis que não ocupam memória ou processamento, mas deixam a tabela de processos exclusiva para eles, pois continuam utilizando IDs de processo, que são recursos finitos.

Dessa maneira, se fixa a importância do manuseio dos processos através das atividades práticas demonstradas durante o trabalho, onde utilizamos conceitos vistos em aula, tanto na Parte 1, mais teórica, apenas manipulando processos, quanto na Parte 2, mais prática, com a implementação de programas que criam processos.

## 5 Referências

CAMPIOLO, R. A. G. R. *Aula 2.2: Processos - Criação e manipulação de processos*. [S.l.], 2025. Disponível em: <[https://moodle.utfpr.edu.br/course/format/tiles/mod\\_view.php?cmid=1489322](https://moodle.utfpr.edu.br/course/format/tiles/mod_view.php?cmid=1489322)>[https://moodle.utfpr.edu.br/course/format/tiles/mod\\_view.php?cmid=1489322](https://moodle.utfpr.edu.br/course/format/tiles/mod_view.php?cmid=1489322). Acesso em: 12.04.2025. Nenhuma citação no texto.

CORNEL BOWERS. *Writing Your Own Shell*. [S.l.], 2025. Disponível em: <<https://www.cs.cornell.edu/courses/cs414/2004su/homework/shell/shell.html>><https://www.cs.cornell.edu/courses/cs414/2004su/homework/shell/shell.html>. Acesso em: 12.04.2025. Nenhuma citação no texto.

WIKIPEDIA. *Forkbomb*. [S.l.], 2025. Disponível em: <[https://en.wikipedia.org/wiki/Fork\\_bomb](https://en.wikipedia.org/wiki/Fork_bomb)>[https://en.wikipedia.org/wiki/Fork\\_bomb](https://en.wikipedia.org/wiki/Fork_bomb). Acesso em: 12.04.2025. Citado 2 vezes nas páginas 7 e 11.