

# Processes

processes practice for CSC420 Operating Systems Spring 2022 Lyon College

## README

- This file accompanies lectures on the shell and `bash(1)`. To gain practice, you should type along in your own Org-mode file. You have to have Emacs and my `.emacs` file installed on your PC or the Pi you're working with.
- This section is based on chapter 10 of Shotts, *The Linux Command Line* (2e), NoStarch Press (2019).
- To make this easier, use the auto expansion (`<s`). This will only work if you have my `.emacs` file ([from GDrive](#)) installed.
- Add the following two lines at the top of your file, and activate each line with `C-c C-c` (this is confirmed in the echo area as `Local setup has been refreshed`):

```
#+PROPERTY: header-args: bash :results output
```

- Remember that `C-M-\` inside a code block indents syntactically (on Windows, this may only work if you have a marked region - set the mark with `C-SPC`).

## Overview

- Modern operating systems are *multitasking*, which means they create an illusion of doing more than one thing at once.
- They do this by rapidly switching from one executing program to another.
- The **kernel** manages this through clever **process management**, which really is clever **memory management**.
- This is illustrated in the figure [1](#). The simple program is all over the computer's memory.

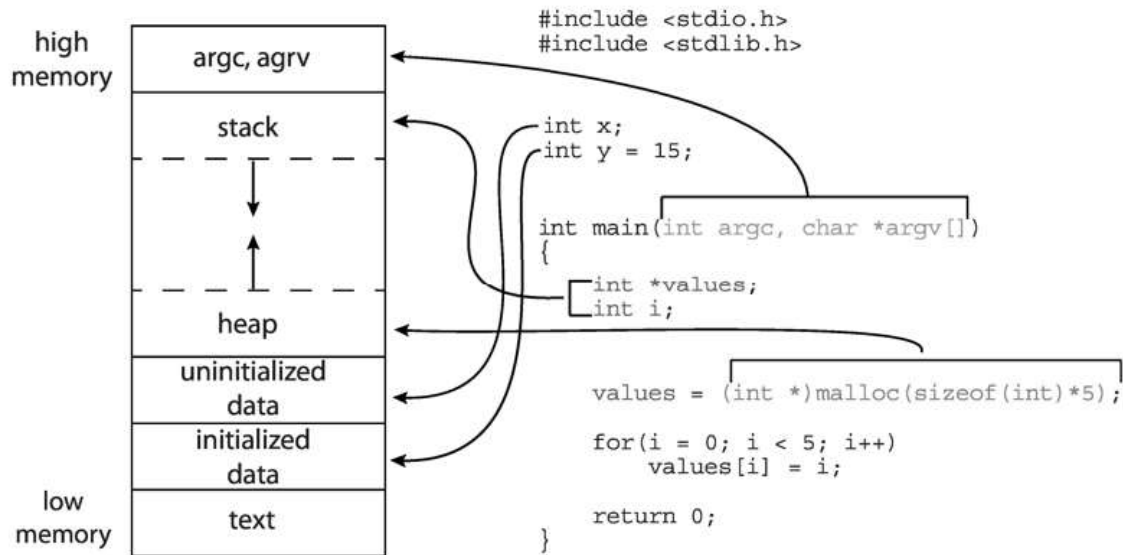


Figure 1: Memory Layout of a C Program (Source: Silberschatz et al)

- As always, let's focus on stuff we can do ourselves. This includes a bunch of new shell commands:

COMMAND	MEANING
ps	Report a snapshot of current processes
top	Display tasks dynamically
jobs	List active jobs
bg	Place a job in the background
fg	Place a job in the foreground
kill	Send a kill signal to process
killall	Kill processes by name
shutdown	Shut down or reboot the system

## How a process works

- When the OS starts up, the **kernel** launches the **init** program, which in turn runs a series of shell scripts (in `/etc`) that start all the system services.
- [ ] Check `/etc` out now - you find e.g. the directory `/etc/cups`, which contains scripts for the Common UNIX Printing System (CUPS).
- Many of the services are *daemon* programs - they just sit in the background and do their thing without a user interface (UI).
- **init** itself is a daemon, also called **systemd**. The shell program **systemctl** allows indirect access to all services.
- [ ]

## Grab a daemon!

In the code block 1,

1. run the command `systemctl status`,
2. tee its output to a text file `systemctl.txt`
3. `grep` for the login daemon program `login`

```
systemctl status
```

- If a program (like `init`) can launch other programs, it's a *parent process* producing a *child process*.
- **How does the kernel maintain control?** By assigning a *process ID* (PID) to every process.
- Processes are assigned in ascending order beginning with `init`, which has PID 1.
- The **kernel** also tracks process memory and readiness to resume execution. Like files, processes have owners and userIDs.

## Viewing processes statically

- The `ps` program has a lot of options (check `ps(1)`)
- [X]

Run `ps` without options.

```
ps
```

PID	TTY	TIME	CMD
13	tty1	00:00:00	bash
14	tty1	00:00:00	ps

- The result is confusing because you're inside another program now.
- [X]

Open a shell (in Emacs with `M-x shell` or a terminal) and type `ps`. You should see something like this:

PID	TTY	TIME	CMD
12254	pts/1	00:00:00	bash
12257	pts/1	00:00:00	ps

- **What this means:**
  - You see two PID - the shell program and the `ps` program
  - TTY ("teletype") is the *controlling terminal* for the process
  - TIME is the amount of CPU time consumed by the process
- [X]

Run `ps` again, this time add the option `x`

- `ps x` (no dash!) shows all processes regardless of what terminal they are controlled by. `?` indicates no terminal (like daemons).

- [X]

How many processes that you own that have no terminal?

- [X]

List only the first 5 lines of the `ps -x` listing.

- The column `STAT` reveals the current status of the process, see table 1.

STATE	MEANING
R	Running or ready to run
S	Sleeping, waiting for an event (e.g. keystroke)
D	Uninterruptible sleep, waiting for I/O (e.g. disk)
T	Stopped, received instruction to stop
Z	Zombie child process, abandoned by parent
<	High priority (not <i>nice</i> - more CPU time)
N	Low priority ( <i>nice</i> ) - served once < are done

There may be more characters denoting exotic process characteristics (see `ps(1)`). E.g. *s* is a *session leader*, *+* is a *foreground* process, and *l* is multi-threaded.

- [ ]

You get even more information with the option `aux`. Redirect the output of `ps aux` to a file `psaux.txt`, and print only the first 5 lines.

```
ps aux | tee psaux.txt | head -5
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.0	8940	332	?	Ss1	19:48	0:00	/init
root	21	0.0	0.0	8940	228	tty1	Ss	19:48	0:00	/init
marcus	22	0.0	0.0	16664	1568	tty1	S	19:48	0:00	bash
marcus	23	0.0	0.0	18660	1896	tty1	R	19:48	0:00	ps aux

- You should see PID 1, the `init` program. The `plash` options means that you can see a splash screen during boot.
- Table 1 shows some header definitions

HEADER	MEANING
USER	User ID - this is the process owner

HEADER	MEANING
%CPU	CPU usage in percent
%MEM	Memory usage in percent
VSZ	Virtual memory size
RSS	Resident set size - RAM use in kB
START	Process starting time and date

- [ ]

Why is the CPU usage of `init` zero, while the Memory usage is non-zero? How much RAM does the program actually use?

ANSWER:

## Viewing processes dynamically

- `ps` provides a snapshot, but `top` provides a real-time view.
- [ ] Open a terminal (in or outside of Emacs) and run `top`. You can stop the command with `C-c` or `q`.
- `top` refreshes every three seconds and shows the top system processes. It includes a summary at the top and a table sorted by CPU activity at the bottom.

```
top - 21:52:54 up 2 days, 9:47, 1 user, load average: 0.19, 0
Tasks: 180 total, 1 running, 179 sleeping, 0 stopped, 0 zo
%Cpu(s): 0.4 us, 1.7 sy, 0.0 ni, 97.8 id, 0.1 wa, 0.0 hi,
MiB Mem : 3787.2 total, 1964.6 free, 474.2 used, 1348.4
MiB Swap: 100.0 total, 100.0 free, 0.0 used. 2971.7
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM
1161	pi	20	0	302736	97504	77316	S	5.6	2.5
12974	pi	20	0	11356	3052	2600	R	1.0	0.1
10766	pi	20	0	121660	32344	27112	S	0.7	0.8
12	root	20	0	0	0	0	I	0.3	0.0
321	avahi	20	0	6916	2780	2520	S	0.3	0.1
1258	pi	20	0	291240	73884	58144	S	0.3	1.9
12783	root	20	0	0	0	0	I	0.3	0.0
12926	root	20	0	0	0	0	I	0.3	0.0
1	root	20	0	33832	8784	6868	S	0.0	0.2
2	root	20	0	0	0	0	S	0.0	0.0
3	root	0	-20	0	0	0	I	0.0	0.0
4	root	0	-20	0	0	0	I	0.0	0.0
8	root	0	-20	0	0	0	I	0.0	0.0
9	root	20	0	0	0	0	S	0.0	0.0
10	root	20	0	0	0	0	S	0.0	0.0
11	root	20	0	0	0	0	S	0.0	0.0

Figure 2: Top view

- The system summary contains a lot of good stuff. Table 1 gives a rundown.

ROW	FIELD	MEANING
1	top	Program name
	21:52:54	Current time of day
	up 2 days 9:49	<i>uptime</i> since last boot
	1 user	No. of users logged in
	load average	No. of processes waiting to run Values < 1.0 means not busy
2	Tasks:	No. of processes and their states total, running, sleeping, stopped
3	Cpu(s):	Activities that the CPU performs: us: user processes (not kernel) sy: system processes (kernel) ni: nice (low prio) processes id: idle processes wa: waiting for I/O
4	Mem:	Physical RAM used
5	Swap:	Swap space (virtual memory) used

- top accepts some keyboard commands like h (help) and q (quit).
- top is better than any graphical application (e.g. the Task Manager that you have on your Pi) - it is faster and consumes far less resources.

## Controlling processes

To **not** see the emphatic characters like ~ or \* or / in the Org file text, run the following code chunk (or put the code in your `/.emacs` file): if successful, you should see "t" in the minibuffer.

```
(setq-default org-hide-emphasis-markers t)
```

If you don't put it in your `/.emacs` file, the command will only work for the current Emacs session.

## Interrupting a process

- As a guinea pig program, we use `emacs`.
- [ ] Open a terminal (inside Emacs after splitting the screen with `C-x 2` or outside of Emacs), and enter `emacs` at the prompt. A new Emacs editor window appears. Notice that the terminal prompt does not return.
- [ ] Close the new Emacs editor manually by clicking on the x in the upper right corner. The prompt in the Shell returns.

- [ ] Enter emacs again in the shell, and interrupt it with CTRL-C (outside of Emacs, or with C-c C-c on the Emacs \*shell\*).
- Many programs can be interrupted this way by sending an **interrupt** signal to the **kernel**.

## Putting a process in the background

- The terminal has a *foreground* and a *background*. To launch a program so that it is immediately placed into the background, follow it with an ampersand & character
- [ ] Start Emacs from the shell *in the background*. An Emacs window should open. Look at the terminal.
- The message that appeared is part of shell *job control*. It means that we have started job number 1 with the PID 13899. If you check the process table with `ps`, you should see the process as shown below (with a different PID).

```
[1] 13899
```

- [ ]

`grep` the emacs process from the process table using the PID. Write the command in the code block below:

```
13928 pts/1    00:00:04 emacs
```

The output in my terminal looks like this:

- [ ]

The `jobs` command lists the jobs that were launched from our terminal. Try it. You should see something like this:

```
[1]+  Running                  emacs &
```

## Returning a process to the foreground

- A process in the background is immune from keyboard input - you cannot interrupt it with CTRL-C. To return it to the foreground, use the `fg` command.
- [ ] On the shell where you started it, return the process to the foreground with the command `fg %1`. The 1 is the `jobspec`.
- [ ] Kill the Emacs process with C-c C-c or CTRL-C on the shell where you started it.

## Stopping or pausing a process

- [ ]

Start an emacs process in a terminal (NOT in an Emacs shell) - it's now in the foreground. If you press CTRL-z in the shell (you may need to click in the terminal window), the process is stopped.

```
pi@raspberrypi:~ $ emacs
^Z
```

```
[1]+  Stopped                  emacs
pi@raspberrypi:~ $
```

- [ ] To bring the process back, you can either bring it into the foreground with `fg %1`, or resume the process in the background with `bg %`. Try both.
- **Why would you launch a graphical program from the shell?**
  - The program may not be listed in the GUI
  - You see error messages that otherwise are invisible
  - Some graphical programs have useful command line options

## Killing a process

- **Cygwin:** The process control commands below also work in Cygwin. Every process has a PID and a WINPID. The latter will kill the process.

```
birkenkrahe@LCjvz1b3 ~
$ ps aux
  PID   PPID  PGID   WINPID  TTY      UID    STIME  COMMAND
  _64/bin/emacs
    632    631   632     3432  pts/0    1110047 18:21:55 /usr/bin/bash
    631      1   631    18952  ?        1110047 18:21:55 /usr/bin/mintty
    672   632   672    16360  pts/0    1110047 18:50:58 /usr/bin/ps
```

- [ ]

`kill` is used to terminate processes using the PID. Start Emacs from the shell *in the background* (inside or outside of emacs), and then kill it with `kill PID`.

**In Cygwin:** open another cygwin terminal, run `ps` and kill the emacs process in the **other** cygwin shell.

*Tip: you get the PID with `ps`, or right after executing the background command.*

- `kill` does actually not "kill" the process, it sends it a signal. We have already used some of these signals:

SIGNAL	MEANING
INT	CTRL-C - interrupt process
TSTP	CTRL-Z - terminal stop
HUP	Hang up (used by daemons)
KILL	Kill without cleanup
TERM	Terminate with <code>kill</code>
STOP	Stop without delay

- Some of these signals are sent to the target program (identified by PID) while others are sent straight to the kernel.

## More process commands

Some fun commands to play with and explore. We already looked at `ps` and `ps`. You may have to install these.



COMMAND	MEANING
ps tree	Process list arranged as tree pattern
vmstat	System usage snapshot
xload	Draws a graph showing system load over time
tlod	Draws graph in terminal

## Summary

## References

- Silberschatz, Galvin and Gagne (2018). Operating System Concepts - 10th edition, Wiley.

Author: Marcus Birkenkrahe

Created: 2022-04-05 Tue 19:48