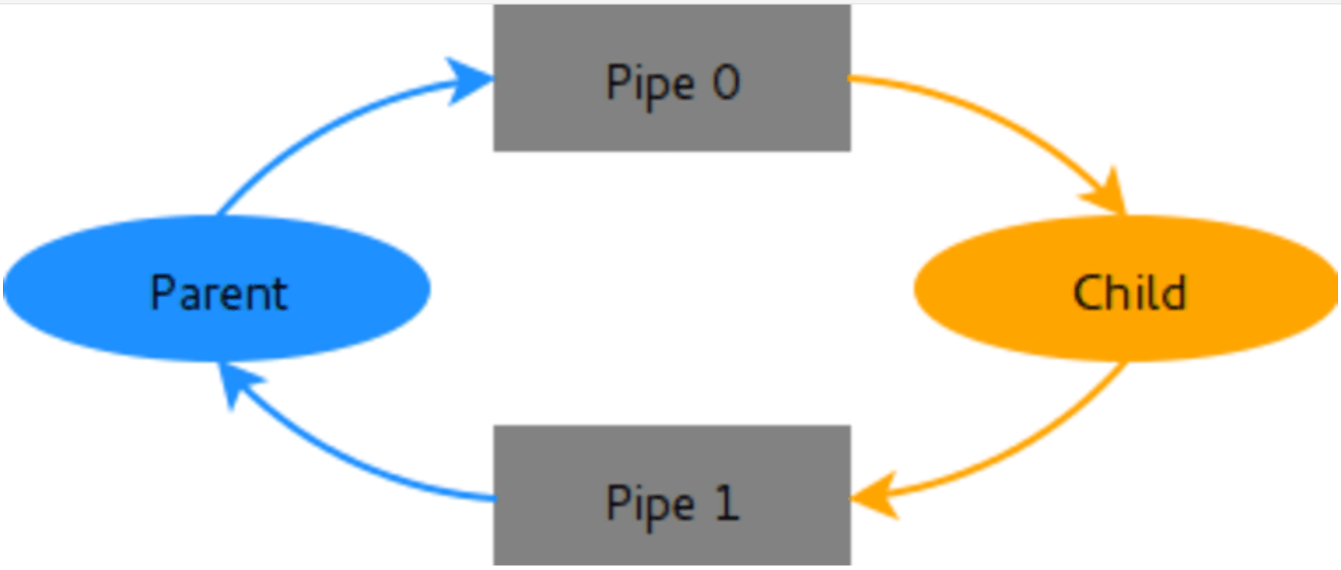


Leandro Proença
Posted on 13 de jul. de 2022

❤️ 8 🦄 2

Inter-process communication: pipes

#unix #linux

Computers, OS and networking (3 Part Series)

After learning how OS processes can use file descriptors for IPC, it's time to analyse another IPC approach: pipes.

Let's recap the following example:

```
$ echo 'my precious' > rawcontent.txt
$ base64 < rawcontent.txt
```

- the program `echo` sends data to the redirected STDOUT
- the `echo` output is used as the redirected STDIN for the program `base64`

Note the pattern here: it looks like a **pipeline of data transformation**, where the output of the first program is "enqueued" to the input of the second program.

UNIX pipelines

UNIX-like systems provide a mechanism for IPC called **pipeline**.

Instead of writing such a sentence in multiple lines, the OS allows us to write the entire sentence in a single line using the operator `|` between programs.

```
$ echo 'my precious' | base64

bXkgcHJlY2lvdXMK
```

You've seen this pipe stuff elsewhere, am I right?

It's called anonymous pipe.

Anonymous Pipe

Anonymous pipes employ a FIFO (first-in, first-out) communication channel for *one-way* IPC.

By *one-way*, it means the data flows in one-direction only. It's the opposite of *bi-direction* communication, where data flows in both directions in a *full-duplex* way.

```
$ ps ax | grep docker | tail -n 3

62374 s039  S+     0:05.31 /usr/local/bin/com.docker.cli run -it ubuntu bash
65442 s040  S+     0:02.93 docker run -it ubuntu bash
65445 s040  S+     0:02.86 /usr/local/bin/com.docker.cli run -it ubuntu bash
```

When a **pipe** (`|`) is created, it's opened a *pair of file descriptors*, one for *writing* and other for *reading*.

Because pipes are *anonymous*, both file descriptors last only as long as the processes, so they are automatically closed when both processes terminate.

Named pipes

Similar to anonymous pipes, **named pipes** also employ FIFO (first-in, first-out) communication channel for *one-way* IPC.

The main difference is that named pipes are created explicitly using the program `mkfifo <filename>`. A *single file* in the **filesystem** is then created, which will be opened for *reading* by one process and for *writing* by another process.

```
$ mkfifo myqueue
```

A file called **myqueue** is created. The *reader* process can then open the pipe:

```
$ cat myqueue
```

The process keeps blocked until a message arrives in the pipe.

Meanwhile, another process can *write* to the pipe:

```
# The echo message STDOUT is being redirected to the pipe, because it's a file!
$ echo 'some message' > myqueue
```

Now look at the message arriving in the *reader* process:

```
$ cat myqueue

some message
```

Be aware that named pipes *live beyond the processes* they are bound to, hence they should be removed manually when become unused:

```
rm myqueue
```

Yay!

Conclusion

In this article we learned how UNIX-like systems use pipes for IPC.

Primarily, pipes are a *one-way* communication channel for IPC.

Anonymous pipes use **a pair of file descriptors** and are closed automatically when the process is finished.

Named pipes use a *named file* in the filesystem and should be closed manually when they are no longer used.

I hope you could understand a bit more about IPC. In the next post I'll cover UNIX sockets for inter-process communication.

Computers, OS and networking (3 Part Series)

1 A brief history of modern computers, multitasking and operating …

2 Inter-process communication: files

3 **Inter-process communication: pipes**

Top comments (2)

Alexander Bombis • 11 de nov. de 22

Hej - that’s very helpful - thanks for your type to give us the knowledge about Files and Pipes❤️

Do you know, if you write the article about sockets?

Leandro Proença • 16 de nov. de 22

Hello, I'm planning to write about sockets soon, I'll let you know, thanks!

Code of Conduct • Report abuse

DEV Community

Trending in Linux

The Linux community is delving into thread creation in Linux x86_64, becoming proficient with the CLI, efficient use of `sed` command in shell scripts, packaging Go for Arch Linux, and the ease of Arch Linux installation for beginners.

How are threads created in Linux x86_64

beto-bit • Sep 23

#linux #assembly #c

🔮 Let's become CLI wizards

Theo • Sep 12

#beginners #linux

When to use the “sed” command efficiently in Shell scripts

Ramon Soarez • Sep 13

#linux #performance #bash

Packaging Go for Arch Linux Tutorial

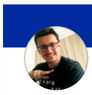
Talha Altınel • Sep 17

#linux #go #opensource

Arch Installation for Beginners

Talha Altınel • Sep 14

#linux #opensource



Leandro Proença

Programmer • I occasionally write blog posts in both English and Brazilian Portuguese.

LOCATION
Brazil

EDUCATION
BSc, Information Systems

WORK
Software Developer

JOINED
8 de ago. de 2017

More from Leandro Proença

[pt-BR] Fundamentos do Git, um guia completo
#git #linux #braziliandevs

Git fundamentals, a complete guide
#git #linux

Tekton CI/CD, part IV, continuous delivery
#kubernetes #docker #linux #agile