EXECUTING AND KILLING
PROGRAMS TOGETHER

My personal notes on

List Of Topics

By

0xLeo (github.com/0xleo)

September 12, 2020

DRAFT X.YY
Missing: . . .

Contents

1 Path priority and executing binaries together

1.1 Motivation

I was browsing the Arch wiki on how to automatically launch `blockify` when `spotify` is launched and how to kill it in the same manner.

The wiki instructs to place the following script at `/usr/local/bin/spotify`:

```
#!/bin/sh

spotify=/usr/bin/spotify

if [[ -x $spotify && -x /usr/bin/blockify ]];
then
  blockify &
  block_pid=$!
  $spotify
  trap "kill -9 $block_pid" SIGINT SIGTERM EXIT
fi
```

First, let's try to recreate something similar with some simple dummy executables.

1.2 How path priority works

By path priority, I refer to the sub-paths in the system's `PATH` environmental variable. The latter variable tells a Unix system which directories to search when executing a program (command). The directories in the path are separated by `:`. The latter variable looks like:

`PATH="dir1:dir2:...:dirN"`

When executing a command, its executable is first searched in `dir1`, then `dir2`, then ... `dirN` so the priority is:

$$Pr(dir_1) > Pr(dir_2) > \ldots > Pr(dir_N)$$

A typical `PATH` in a Unix system contains the following:

`/usr/local/sbin:/usr/local/bin:/usr/bin:...`

, which shows what the priority is.

Going back to the blockify script, since `/usr/local/bin/spotify` has greater priority than `/usr/bin/spotify`, the script gets executed instead of plain `spotify`. The script executes blockify and stores its PID (`$!`) in a variable

1.3 Traps; waiting for signal to execute action

The next goal of the script is to kill `blockify` when `spotify` terminates, achieved by the line:

`trap "kill -9 $block_pid" SIGINT SIGTERM EXIT`

In Unix, traps are used to activate handlers when a particular signal is received. The syntax to set up a trap is the following:

`trap <"handler command"> <signal list>`

In this case, the handler is to kill blockify, i.e. `kill -9 $block_id`. We want the handler to be reached when signals `SIGINT, SIGTERM, EXIT` are received, i.e. when spotify terminates. Signals `SIGINT`, `SIGTERM` are received when a command terminates the program and `EXIT` is a pseudo-signal received when the program exits. That's it.

1.3.1 Traps; a simple example

Let's say we have a script keeps writing random numbers from 0 to 99 to a file and is meant to remove the file when it exits, either manually or by itself.

```
#!/bin/bash

output_file=/tmp/output.txt
> $output_file

```

```
# --- trap handler
cleanup() {
    echo "Exit received. Cleaning up..."
    rm $output_file
}
# ---

trap "cleanup" SIGINT SIGTERM EXIT


#--- main work
for i in `seq 1 10`; do
    echo $[ $RANDOM \% 100 ] >> $output\_file
    sleep 1
done
#---
```

When the user kills the program by Ĉ or when the program exits, it calls the handler `cleaup()` and removes the output file.