CCPS590 Lab 2 – Signals in Linux

Preamble

In this lab you will explore sending and handling signals in Linux. It should also serve as a refresher for working in the Linux shell/terminal. Signals are sent by the CPU (or more accurately, some program executing on the CPU), while interrupts are raised by hardware external to the CPU. An interrupt, when its handler is executed, might result in a signal being sent. Note that the interrupt handler is still a program executing on the CPU! Since we can't easily write our own interrupts, we'll have fun with signals instead.

Lab Description

1) A user can use the keyboard to send a signal to a running program. For example, CTRL-C sends the SIGINT signal to the process currently executing in the foreground. When a program receives SIGINT, it shuts down *gracefully* (closes files, etc. before terminating).

a) Look at P1.c, understand what it does, then compile and run it. Try sending it SIGINT from the keyboard.
b) Run P1 again, this time send it SIGSTOP by pressing CTRL-Z. Notice what happens to the program's output.
c) Run P1 again and send SIGSTOP again. You now have two instances of P1 in the background.
d) Find the job number of each P1 instance using the **`jobs`** command, and find the process IDs using **`ps -f`**.
e) Use **`fg`** to send SIGCONT to a given job. The command **`fg 1`** will send SIGCONT to job 1 (bring it to the foreground, resume its execution)
f) Use a combination of the previous signals to terminate all instances of P1.

2) We can also use the **`kill`** command to send signals (counterintuitively, **`kill`** can be used to send signals. It does not simply kill processes). This command takes two arguments – The signal to be sent, and the process ID to send it to. For example:

```
kill -SIGINT 9999
```

SIGINT can be replaced with any named signal (SIGSTOP, SIGCONT, etc.)

3) Run P1 with argument A in the background, and again with argument B in the foreground. To run a process in the background, pass & as an argument when running the program. For example:

```
./P1 A &
./P1 B
```

Even though P1 is in the background, it is running! It will still output to the terminal. You can type commands, and they will appear interleaved with P1's output. The output from P1 will not impact the next command you enter. Try it!

4) With the jobs from part 3) still running, open another terminal window. From the second window, find the process IDs of both P1s by using the **`ps -u <username>`** command. Stop both P1s from the second window, and then resume them again. Finally, terminate them both. How did you do all this?

5) Signals in Linux have both a mnemonic name and a numeric value. For example, SIGINT is 2. Thus, SIGINT could be sent to process with ID 9999 like so:

```
kill -2 9999
```

It is typically better to use the signal name rather than the signal number (for the same reasons it is good to have descriptive variable names). However, not every signal *number* corresponds to a named signal, and the list of named signals will vary by platform. What happens if you send signal 99 to a running instance of P1?

6) Next, we will look at signal handlers. The OS has default signal handler routines for all signals, but we can register our own and override the default handler.

Look at P2.c, understand what it does, see the syntax for registering a signal handler. Compile and run P2. Try terminating it with CTRL-C. What happened? How can we terminate this program? Are we doomed to let it end naturally? (*Hint: Google for signal number 9, or see the cartoon on the next page…)*

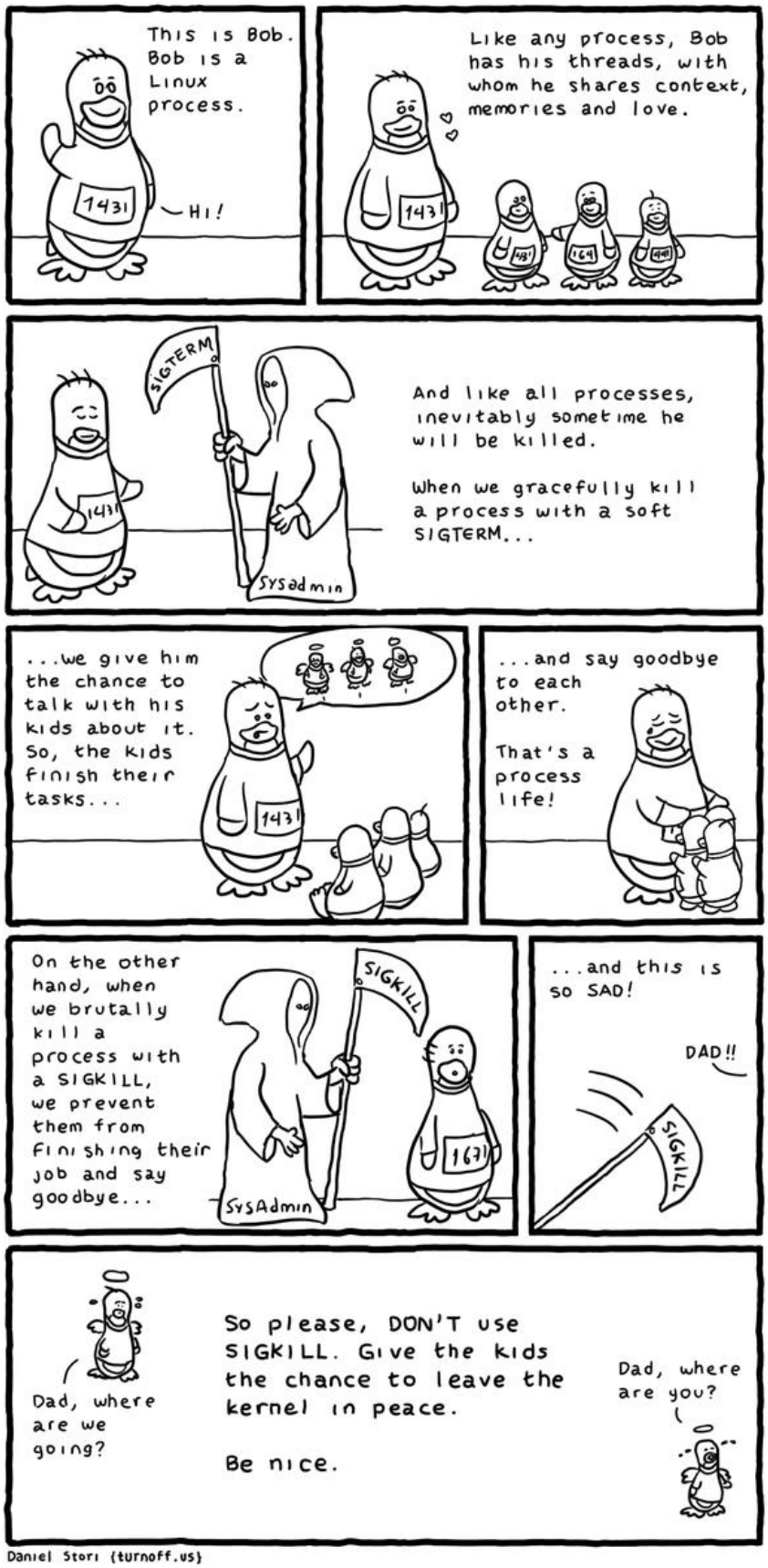
7) We've seen how to send signals using keyboard shortcuts (CTRL-C, CTRL-Z) and the **`kill`** command from the terminal. Signals can also be sent using the **`kill`** *function* in a C program. Look at P3.c and understand what it does. Try using P3 to terminate P2. Does it work? Why or why not? What change would you make to ensure it will *always* terminate any user process?

8) Give a brief description (1 or 2 sentences each) of the signals SIGINT, SIGTERM, and SIGKILL, with a focus on how they differ.

Submission

Work through the questions above and submit written answers to questions 4, 5, 6, 7, and 8.

Labs are to be submitted ***individually***! Submit your answers on D2L using any common digital format (PDF, doc, txt, etc). Make sure your answers are formatted cleanly and are easy to read.



Daniel Stori {turnoff.us}