Systems Programming

updated : 28 sept 2019

Signals

Overview

Last Time

- Shells

Readings for today

- Chapter 20 - Signals: Fundamental Concepts

Interlude

- Kill Dash Nine - by Monzy -

Signals

- One process can send a signal and interrupt another process
- signal is a small int typically < 32

- The process receiving the signal has to set up ahead of time (using `sigaction` system call)
  - when this signal is received do this action(function)

- Can you name programs you use that catch signals?

What Signals?

- *A signal* is a notification to a process that an event has occurred
- One process can (if it has suitable permissions) send a signal to another process
- Each signal is defined as a unique (small) integer, starting sequentially from 1.

Signal Generation

- **When is the kernel generate signals?**
  - A hardware exception occurred, meaning that the hardware detected a fault condition that was notified to the kernel
  - The user typed one of the terminal special characters that generate signals (e.g ctrl+z)
  - A software event occurred. For example, input became available on a file descriptor, the terminal window was resized, a timer went off, the process’s, CPU time limit was exceeded, or a child of this process terminated.

Signal Actions

- **Upon delivery of a signal, a process carries one of the following default actions, depending on the signal:**
  - The signal is ignored;
  - The process is terminated (killed).
  - A core dump file is generated
  - The process is stopped
  - Execution of the process is resumed after previously being stopped.

SIGNAL(2) Linux Programmer's Manual SIGNAL(2)

NAME signal - ANSI C signal handling

SYNOPSIS

```
    #include <signal.h>
   typedef    void   (*sighandler_t)(int);

   sighandler_t   signal(int signum, sighandler_t handler);
```

DESCRIPTION

The behavior of signal() varies across Unix versions, and has also varied historically across different versions of Linux.

Avoid its use: use sigaction(2) instead.

SEE ALSO

kill(1), alarm(2), kill(2), killpg(2), pause(2), sigaction(2), signalfd(2), sigpending(2), sigprocmask(2), sigqueue(2), sigsuspend(2), bsd_signal(3), raise(3), siginterrupt(3), sigsetops(3), sigvec(3), sysv_signal(3), feature_test_macros(7), signal(7)

The Signals (POSIX.1-1990)

Signal	Value	Action	Comment
SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Term	Interrupt from keyboard
SIGQUIT	3	Core	Quit from keyboard
SIGILL	4	Core	Illegal Instruction
SIGABRT	6	Core	Abort signal from abort(3)
SIGFPE	8	Core	Floating point exception
SIGKILL	9	Term	Kill signal
SIGSEGV	11	Core	Invalid memory reference
SIGPIPE	13	Term	Broken pipe: write to pipe with no readers

- TERM=terminate, CORE=terminate and dump

The Signals *continued*

SIGALRM	14	Term	Timer signal from alarm(2)
SIGTERM	15	Term	Termination signal
SIGUSR1	30,10,16	Term	User-defined signal 1
SIGUSR2	31,12,17	Term	User-defined signal 2
SIGCHLD	20,17,18	Ign	Child stopped or terminated
SIGCONT	19,18,25	Cont	Continue if stopped
SIGSTOP	17,19,23	Stop	Stop process
SIGTSTP	18,20,24	Stop	Stop typed at tty
SIGTTIN	21,21,26	Stop	tty input for background process
SIGTTOU	22,22,27	Stop	tty output for background process

- The signals SIGKILL and SIGSTOP cannot be caught, blocked, or ignored.
- IGN : Ignore Signal, STOP : Stop the Process

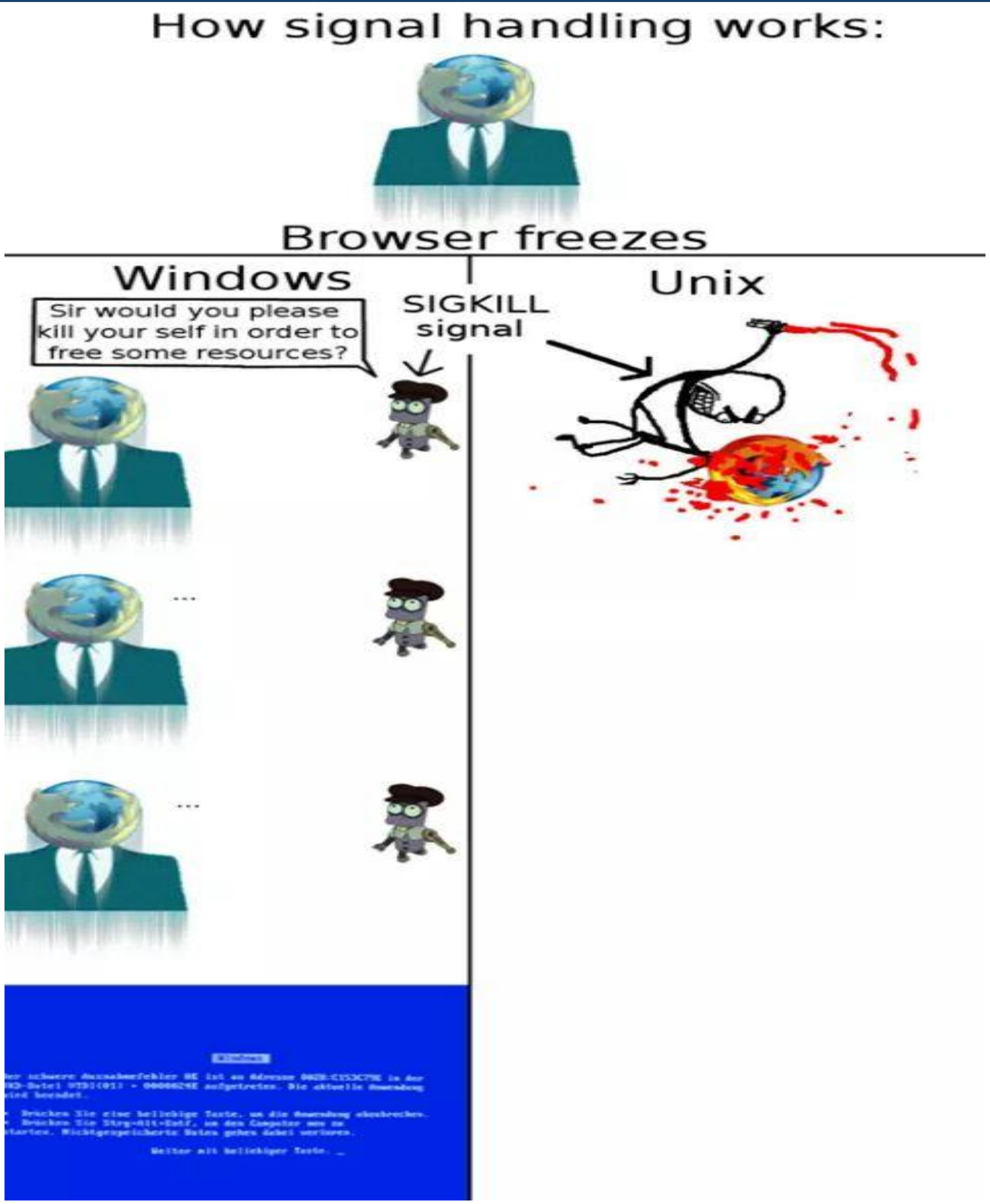
SUSv2 and POSIX.1-2001 Signals

SIGBUS	10,7,10	Core	Bus error (bad memory access)
SIGPOLL		Term	Pollable event (Sys V). Synonym for SIGIO
SIGPROF	27,27,29	Term	Profiling timer expired
SIGSYS	12,-,12	Core	Bad argument to routine (SVr4)
SIGTRAP	5	Core	Trace/breakpoint trap
SIGURG	16,23,21	Ign	Urgent condition on socket (4.2BSD)
SIGVTALRM	26,26,28	Term	Virtual alarm clock (4.2BSD)
SIGXCPU	24,24,30	Core	CPU time limit exceeded (4.2BSD)
SIGXFSZ	25,25,31	Core	File size limit exceeded (4.2BSD)

Most Common Signals

Name	Number	Meaning
HUP	1	Hang Up. The controlling terminal has gone away.
INT	2	Interrupt. The user has pressed the interrupt key (usually Ctrl-C or DEL).
QUIT	3	Quit. The user has pressed the quit key (usually Ctrl-\). Exit and dump core.
KILL	9	Kill. This signal cannot be caught or ignored. Unconditionally fatal. No cleanup possible.
TERM	15	Terminate. This is the default signal sent by the kill command.
EXIT	0	Not really a signal. In a shell script, an EXIT trap is run on any exit, signalled or not.

Intermezzo


How signal handling works:
Browser freezes
Windows
Unix
Sir would you please kill your self in order to free some resources?
SIGKILL signal
...
...

20-1 Signal delivery and handling

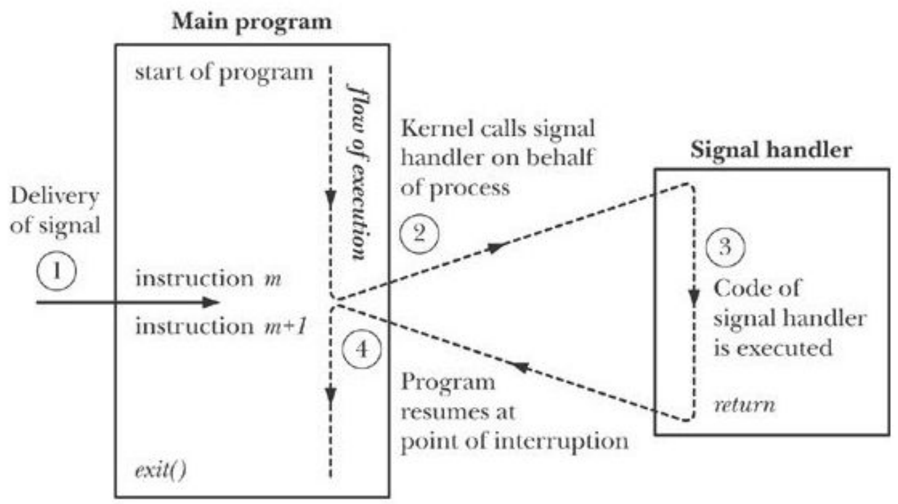
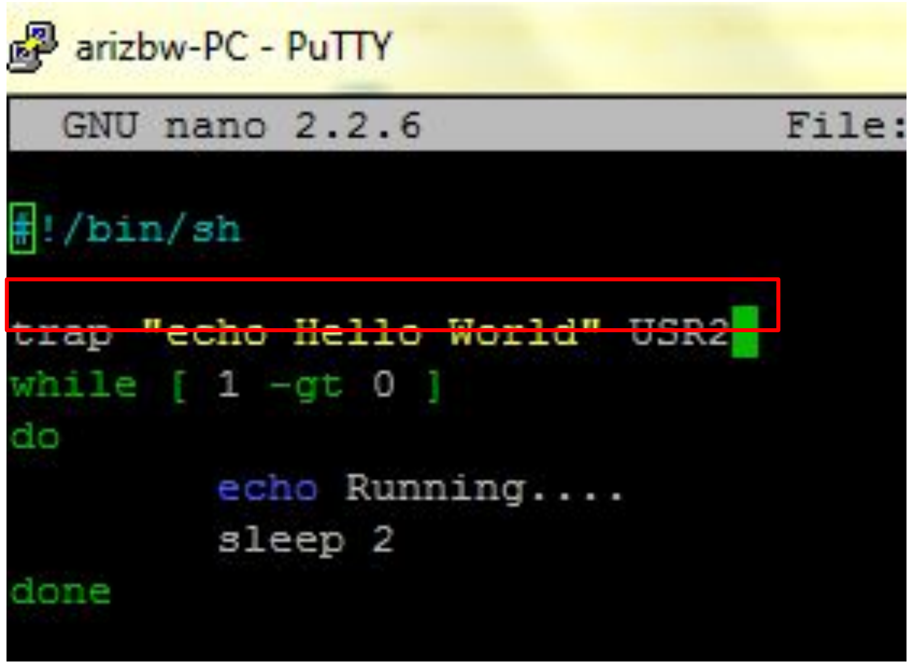

Main program
start of program
flow of execution
Kernel calls signal handler on behalf of process
Signal handler
Delivery of signal
1
2
3
instruction m
instruction m+1
4
Code of signal handler is executed
Program resumes at point of interruption
return
exit()

Figure 20-1. Signal delivery and handler execution

- When do you interrupt?
- mid instruction?
- mid system call?

E.g Signal with trap (1)

Write this code



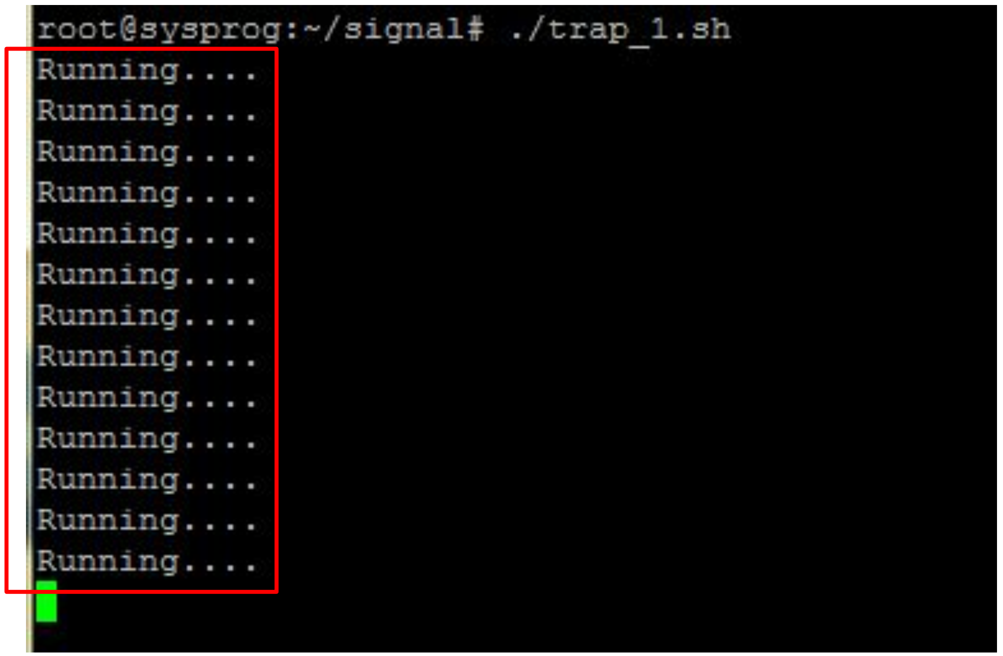
arizbw-PC - PuTTY

```
  GNU nano 2.2.6                    File:

#!/bin/sh

trap "echo Hello World" USR2
while [ 1 -gt 0 ]
do
        echo Running....
        sleep 2
done
```

- Execute the script (./trap.sh)

E.g Signal with trap (1)

The program will loop forever…..



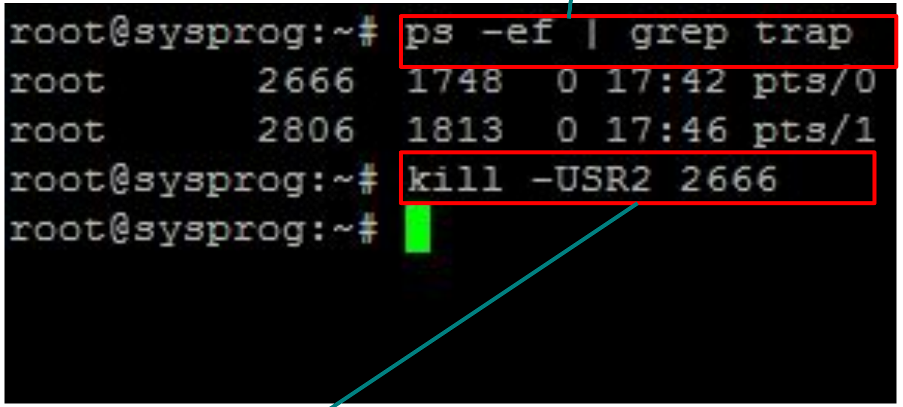
```
root@sysprog:~/signal# ./trap_1.sh
Running....
Running....
Running....
Running....
Running....
Running....
Running....
Running....
Running....
Running....
Running....
Running....
Running....
```

- Duplicate your session to root@localhost or start another session to your virtualbox

E.g Signal with trap (1)

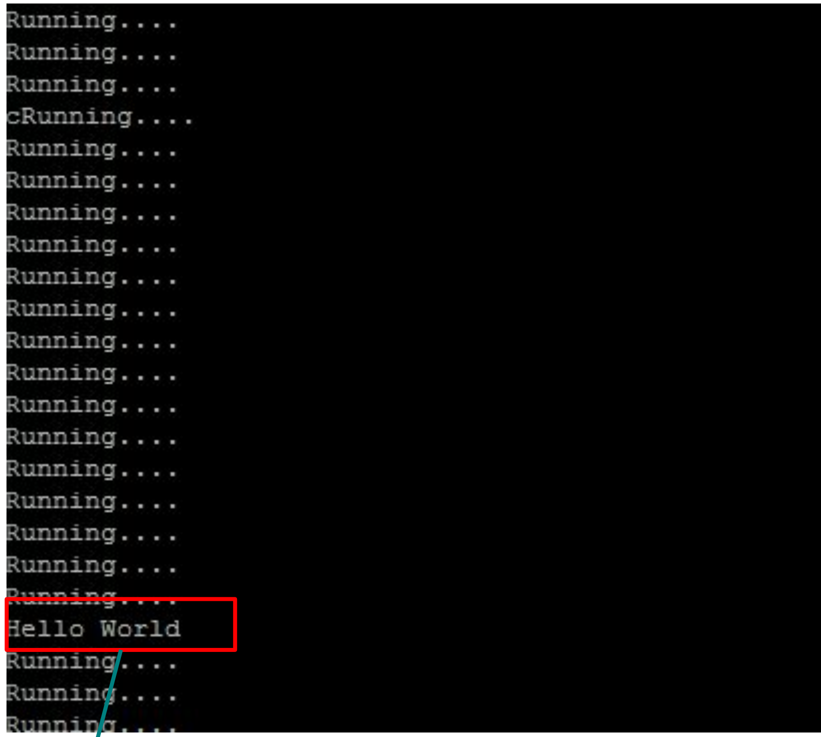
- Use the kill command to “kill” the process

Get the process ID



```
root@sysprog:~# ps -ef | grep trap
root        2666  1748  0 17:42 pts/0
root        2806  1813  0 17:46 pts/1
root@sysprog:~# kill -USR2 2666
root@sysprog:~# 
```

kill -[userdefined trap] [PID]



```
Running....
Running....
Running....
cRunning....
Running....
Running....
Running....
Running....
Running....
Running....
Running....
Running....
Running....
Running....
Running....
Running....
Running....
Running....
Running....
Hello World
Running....
Running....
Running....
```

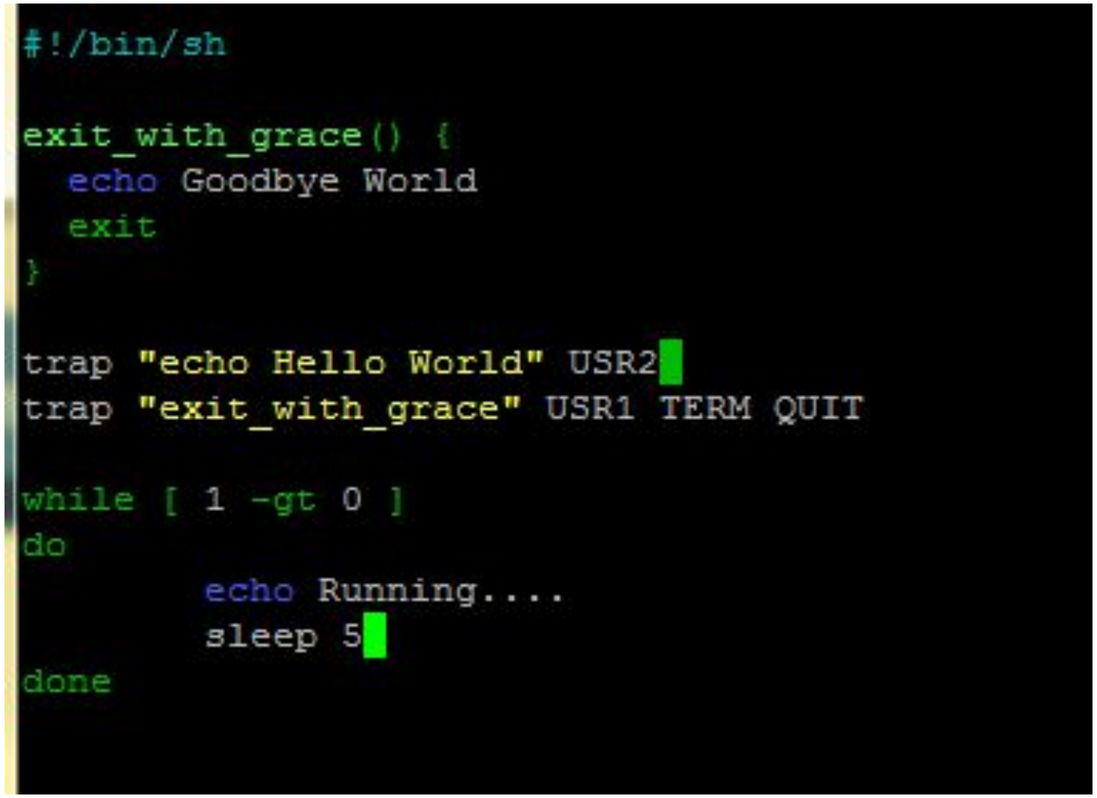
Signal Interrupt

E.g Signal with trap (1)

Did You Kill the process??

E.g Signal with trap (2)

Write this code

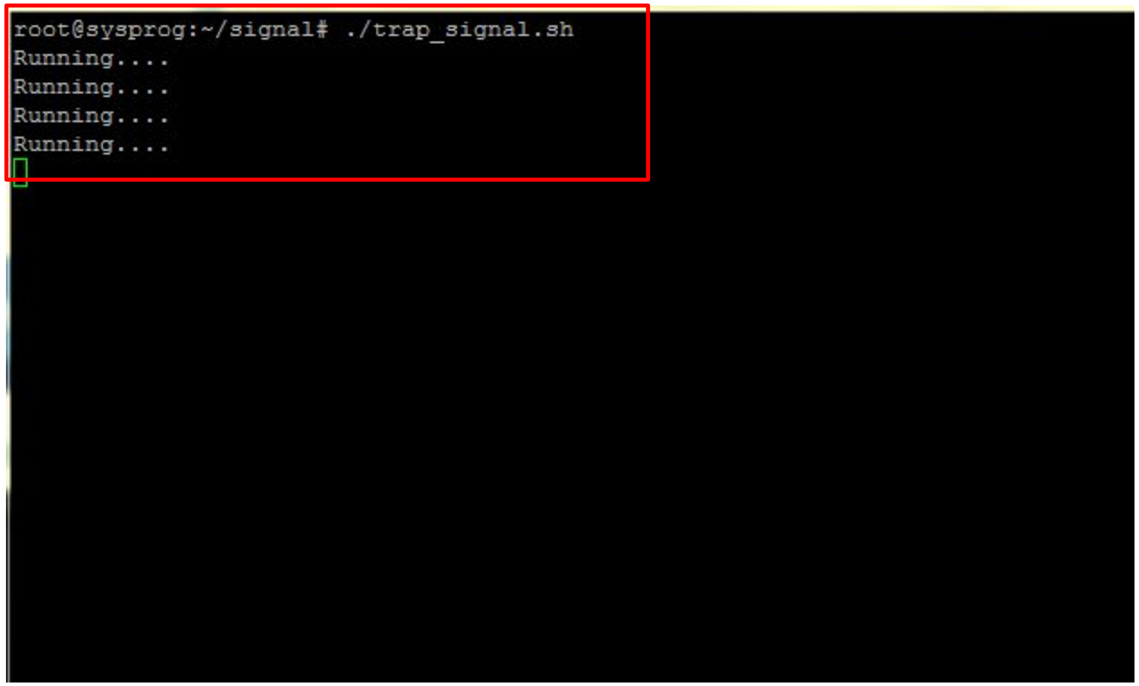


```
#!/bin/sh

exit_with_grace() {
  echo Goodbye World
  exit
}

trap "echo Hello World" USR2
trap "exit_with_grace" USR1 TERM QUIT

while [ 1 -gt 0 ]
do
        echo Running....
        sleep 5
done
```

- Execute the script (./trap_signal.sh)

E.g Signal with trap (2)

The program will loop forever…..

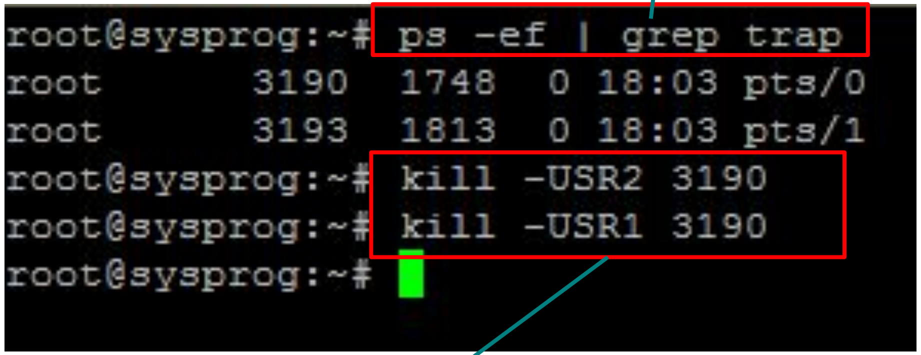


```
root@sysprog:~/signal# ./trap_signal.sh
Running....
Running....
Running....
Running....
```

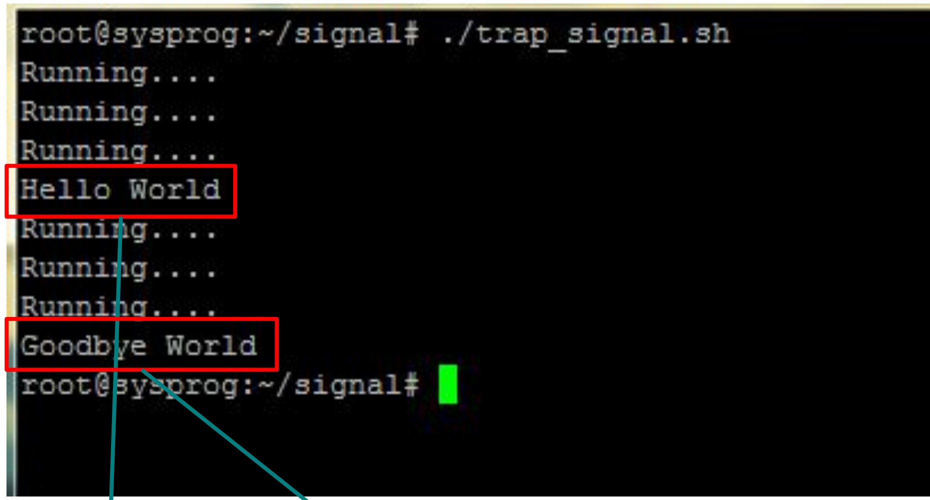
- Duplicate your session to root@localhost or start another session to your virtualbox

E.g Signal with trap (1)

- Use the kill command to “kill” the process



```
root@sysprog:~# ps -ef | grep trap
root      3190  1748  0 18:03 pts/0
root      3193  1813  0 18:03 pts/1
root@sysprog:~# kill -USR2 3190
root@sysprog:~# kill -USR1 3190
root@sysprog:~# 
```



```
root@sysprog:~/signal# ./trap_signal.sh
Running....
Running....
Running....
Hello World
Running....
Running....
Running....
Goodbye World
root@sysprog:~/signal# 
```

Get the process ID

kill -[userdefined trap] [PID]

Signal Interrupt [USR2]

Signal Interrupt [USR1]

Kill Dash Nine - by Monzy -

Ph.D. student at Stanford

http://www.monzy.com/intro/killdashnine_lyrics.html

PARENTAL
ADVISORY
EXPLICIT LYRICS

SOME RIGHTS RESERVED

Kill Dash Nine (Cont')

I guess I'll have to shut you down for good this time,
Already tried a `SIGQUIT`, so now it's KILL DASH 9.
You gotta learn when it's time for your thread to yield;
It shoulda slept; instead you stepped and now your fate is sealed.
I'll take your process off the run queue without even asking 'Cause my flow is like reentrant and preemptive multitasking.

…. ….. …..

And I've got your f****n `pid` and the bottom line
Is that you best not front or else it's KILL DASH NINE.

KILL DASH NINE, No more CPU time.
I run KILL DASH NINE, And your process is mine.
I run KILL DASH NINE, 'Cause it's MY time to shine
So don't step outta line or else it's KILL DASH NINE!

Kill Dash Nine (Cont')

What the lyric is all about ?

Exercise

Build 2 bash scripts which do these function:

Script #1

* Trap [some signal]
* Wait forever
* When the [some signal] happen, it prints data sent from Script #2 (using [temporary file])

Script #2

* Write some data to a [temporary file]
* Get PID of Script #1
* Send [some signal] to PID process Script #1

Get your PID (example)

```
#!/bin/sh

 get_pids(){
         echo Your PID is $$
 }
 trap "get_pids" USR1

 while [ 1 -gt 0 ]
 do
         echo Running.....
         sleep 2
 done
```

Read File and Initiate PID //script1.sh

```
#!/bin/sh

echo $$ > pid.file
read_files(){
        read -r file < /root/signal/file.read
        echo $file
}
trap "read_files" TERM USR1
while [ 1 -gt 0 ]
do
        echo Running.....
        sleep 2
done
```

Read PID from another Process and send signal to it //script2.sh

```
read -r pidfile < /root/signal/pid.file
echo -n "Enter some text > "
read text
echo $text > file.read

kill -USR1 $pidfile
```

QA