

Linux Signals Fundamentals – Part I

by HIMANSHU ARORA on MARCH 5, 2012

What is a signal? Signals are software interrupts.

A robust program need to handle signals. This is because signals are a way to deliver asynchronous events to the application.

A user hitting ctrl+c, a process sending a signal to kill another process etc are all such cases where a process needs to do signal handling.

Linux Signals

In Linux, every signal has a name that begins with characters SIG. For example :

- A SIGINT signal that is generated when a user presses ctrl+c. This is the way to terminate programs from terminal.
- A SIGALRM is generated when the timer set by alarm function goes off.
- A SIGABRT signal is generated when a process calls the abort function.
- etc

When the signal occurs, the process has to tell the kernel what to do with it. There can be three options through which a signal can be disposed :

1. The signal can be ignored. By ignoring we mean that nothing will be done when signal occurs. Most of the signals can be ignored but signals generated by hardware exceptions like divide by zero, if ignored can have weird consequences. Also, a couple of signals like SIGKILL and SIGSTOP cannot be ignored.
2. The signal can be caught. When this option is chosen, then the process registers a function with kernel. This function is called by kernel when that signal occurs. If the signal is non fatal for the process then in that function the process can handle the signal properly or otherwise it can chose to terminate gracefully.
3. Let the default action apply. Every signal has a default action. This could be process terminate, ignore etc.

As we already stated that two signals SIGKILL and SIGSTOP cannot be ignored. This is because these two signals provide a way for root user or the kernel to kill or stop any process in any situation .The default action of these signals is to terminate the process. Neither these signals can be caught nor can be ignored.

What Happens at Program Start-up?

It all depends on the process that calls exec. When the process is started the status of all the signals is either ignore or default. Its the later option that is more likely to happen unless the process that calls exec is ignoring the signals.

It is the property of exec functions to change the action on any signal to be the default action. In simpler terms, if parent has a signal catching function that gets called on signal occurrence then if that parent execs a new child process, then this function has no meaning in the new process and hence the disposition of the same signal is set to the default in the new process.

Also, Since we usually have processes running in background so the shell just sets the quit signal disposition as ignored since we do not want the background processes to get terminated by a user pressing a ctrl+c key because that defeats the purpose of making a process run in background.

Why Signal Catching Functions should be Reentrant?

As we have already discussed that one of the option for signal disposition is to catch the signal. In the process code this is done by registering a function to kernel which the kernel calls when the signal occurs. One thing to be kept in mind is that the function that the process registers should be reentrant.

Before explaining the reason, lets first understand what are reentrant functions?

[RSS](#) | [Email](#) | [Twitter](#) | [Facebook](#)

ENHANCED BY 



EBOOKS

Free

[Linux 101 Hacks 2nd Edition eBook](#) - Practical Examples to Build a Strong Foundation in Linux

[Bash 101 Hacks eBook](#) - Take Control of Your Bash Command Line and Shell Scripting

[Sed and Awk 101 Hacks eBook](#) - Enhance Your UNIX / Linux Life with Sed and Awk

[Vim 101 Hacks eBook](#) - Practical Examples for Becoming Fast and Productive in Vim Editor

[Nagios Core 3 eBook](#) - Monitor Everything, Be Proactive, and Sleep Well



POPULAR POSTS

[15 Essential Accessories for Your Nikon or Canon DSLR Camera](#)

[12 Amazing and Essential Linux Books To Enrich Your Brain and Library](#)

[50 UNIX / Linux Sysadmin Tutorials](#)

[50 Most Frequently Used UNIX / Linux Commands \(With Examples\)](#)

[How To Be Productive and Get Things Done Using GTD](#)

[30 Things To Do When you are Bored and have a Computer](#)

[Linux Directory Structure \(File System Structure\) Explained with Examples](#)

[Linux Crontab: 15 Awesome Cron Job Examples](#)

[Get a Grip on the Grep! – 15 Practical Grep Command Examples](#)

[Unix LS Command: 15 Practical Examples](#)

[15 Examples To Master Linux Command Line History](#)

[Top 10 Open Source Bug Tracking System](#)

[Vi and Vim Macro Tutorial: How To Record and Play](#)

[Mommy, I found it! – 15 Practical Linux Find Command Examples](#)

[15 Awesome Gmail Tips and Tricks](#)

[15 Awesome Google Search Tips and Tricks](#)

[RAID 0, RAID 1, RAID 5, RAID 10 Explained with Diagrams](#)

[Can You Top This? 15 Practical Linux Top Command Examples](#)

[Top 5 Best System Monitoring Tools](#)

[Top 5 Best Linux OS Distributions](#)

[How To Monitor Remote Linux Host using Nagios 3.0](#)

A reentrant function is a function whose execution can be stopped in between due to any reason (like due to interrupt or signal) and then can be reentered again safely before its previous invocations complete the execution.

Now coming back to the issue, Suppose a function func() is registered for a call back on a signal occurrence. Now assume that this func() was already in execution when the signal occurred. Since this function is call back for this signal so the current execution on this signal will be stopped by the scheduler and this function will be called again (due to signal).

The problem can be if func() works on some global values or data structures that are left in inconsistent state when the execution of this function was stopped in middle then the second call to same function(due to signal) may cause some undesired results.

So we say that signal catching functions should be made reentrant.

Refer to our articles [send-signal-to-process](#) and [Linux fuser command](#) to see practical examples on how to send signals to a process.

Threads and Signals

We already saw in the one of previous sections that signal handling comes with its own bit of complexity (like using reentrant functions) . To add on to the complexity, we usually have multi threaded applications where signal handling becomes really complicated.


Every thread has its own private signal mask(a mask that defines which signals are deliverable) but the way signal disposition is done is shared by all the threads in the application. This means that a disposition for a particular signal set by a thread can easily be overruled by some other thread. In this case the disposition mechanism changes for all the threads.

For example, a thread A can choose to ignore a particular signal but a thread B in the same process can choose to catch the same signal by registering a callback function to the kernel. In this case the request made by thread A gets overruled by thread B's request.

Signals are delivered only to a single thread in any process. Apart from the the hardware exceptions or the timer expiry (which are delivered to thread which caused the event) all the signals are passed to the process arbitrarily.

To counter this shortcoming there are some posix APIs like pthread_sigmask() etc that can be used.

In the [next article \(part 2\)](#) of this series, we will discuss about how to catch signals in a process, and explain the practical aspect of signal handling using code snippets.

 Like 0

Add your comment

If you enjoyed this article, you might also like..

1. [50 Linux Sysadmin Tutorials](#)

2. [50 Most Frequently Used Linux Commands \(With Examples\)](#)

3. [Top 25 Best Linux Performance Monitoring and Debugging Tools](#)

4. [Mommy, I found it! – 15 Practical Linux Find Command Examples](#)


5. [Linux 101 Hacks 2nd Edition eBook **Free**](#)

▪ [Awk Introduction – 7 Awk Print Examples](#)

▪ [Advanced Sed Substitution Examples](#)

▪ [8 Essential Vim Editor Navigation Fundamentals](#)

▪ [25 Most Frequently Used Linux IPTables Rules Examples](#)

▪ [Turbocharge PuTTY with 12 Powerful Add-Ons](#)
- 
- [Awk Introduction Tutorial – 7 Awk Print Examples](#)
- [How to Backup Linux? 15 rsync Command Examples](#)
- [The Ultimate Wget Download Guide With 15 Awesome Examples](#)
- [Top 5 Best Linux Text Editors](#)
- [Packet Analyzer: 15 TCPDUMP Command Examples](#)
- [The Ultimate Bash Array Tutorial with 15 Examples](#)
- [3 Steps to Perform SSH Login Without Password Using ssh-keygen & ssh-copy-id](#)
- [Unix Sed Tutorial: Advanced Sed Substitution Examples](#)
- [UNIX / Linux: 10 Netstat Command Examples](#)
- [The Ultimate Guide for Creating Strong Passwords](#)
- [6 Steps to Secure Your Home Wireless Network](#)
- [Turbocharge PuTTY with 12 Powerful Add-Ons](#)
- ### CATEGORIES
- [Linux Tutorials](#)

[Vim Editor](#)

[Sed Scripting](#)

[Awk Scripting](#)

[Bash Shell Scripting](#)

[Nagios Monitoring](#)

[OpenSSH](#)

[IPTables Firewall](#)

[Apache Web Server](#)

[MySQL Database](#)

[Perl Programming](#)

[Google Tutorials](#)

[Ubuntu Tutorials](#)

[PostgreSQL DB](#)

[Hello World Examples](#)

[C Programming](#)

[C++ Programming](#)

[DELL Server Tutorials](#)

[Oracle Database](#)

[VMware Tutorials](#)

Comments on this entry are closed.

bob

March 5, 2012, 8:57 am

as usual, great article with very concise straight to the point explanation.

∞

Harry

March 5, 2012, 2:52 pm

waiting for part 2

∞

Himanshu Arora

March 5, 2012, 8:00 pm

@bob : Thanks

@Harry : Part-2 will be coming soon



∞

Ernest

March 6, 2012, 1:53 am

great info, can't wait for PART2

∞

Rahul Trivedi

March 9, 2012, 9:03 am

....Really Awesome explanantion

∞

Max

November 21, 2013, 3:18 am

Good job! It was a nice introduction to the subject! Thanks for clearing my confusion! 😊

∞

Next post: [Reverse Engineering Tools in Linux – strings, nm, ltrace, strace, LD_PRELOAD](#)

Previous post: [10 Practical Linux nm Command Examples](#)

ABOUT THE GEEK STUFF



My name is **Ramesh Natarajan**. I will be posting instruction guides, how-to, troubleshooting tips and tricks on Linux, database, hardware, security and web. My focus is to write articles that will either teach you or help you resolve a problem. Read more about [Ramesh Natarajan](#) and the blog.

CONTACT US

Email Me : Use this [Contact Form](#) to get in touch me with your comments, questions or suggestions about this site. You can also simply drop me a line to say hello!.

[Follow us on Twitter](#)

[Become a fan on Facebook](#)

SUPPORT US

Support this blog by purchasing one of my ebooks.

[Bash 101 Hacks eBook](#)

[Sed and Awk 101 Hacks eBook](#)

[Vim 101 Hacks eBook](#)

[Nagios Core 3 eBook](#)