System Administration

Alessandro Barenghi

Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

barenghi - at - elet.polimi.it

April 16, 2013

Introduction

Why a system administration lesson?

- Strong binding between system architecture and network stack
- System administration and management skills are required to "survive" in this environment
- As a bonus, they come in handy in a lot of other contexts
- They are taken for granted in other courses

Chosen Platform

Why Linux?

- The chosen platform for the course is GNU/Linux
- Any recent (2010 onwards) Linux distribution is fine for practicing
- No restriction on the redistribution of tools/practicing material
- The notions easily generalise to affine Unices (f.i. MacOS X) with menial changes

Study methodology

The four letter creed

- “Ten minutes of direct practice are worth ten hours of study in system adminstration”
- Choose a distribution and install it in a realistic environment (at least a VirtualBox VM, although real Iron is better)
  - Debian/Ubuntu is an easy shot for beginners
  - Slackware is extremely clean as far as internal structure goes
  - Gentoo might not be for the faint of heart, but it’s very instructive
- Begin practicing soon, these notions take time to consolidate
- Linux is endowed with an outstanding manual available typing `man <command>`

Overview

What you should already know

- How to perform basic operations from the commandline (list files, change directory, copy files)
- Basic knowledge regarding the OS coming from system architecture and OS course (what is a process, how does an OS work)
- Basic knowledge of the underlying hardware, from the same course (how does a context change take place, how is a binary run)
- Sound knowledge of the C language fundamentals: the whole Linux kernel and commandline utilities are written in C

Overview

Lesson contents

- How to manage the multitasking environment in a Linux system
- How to examine what a program is employing as resources
- How to inspect a process running on the system
- How to manage a running system in times of trouble

Commandline interface

The shell

- We will be using a commandline interface to perform all our tasks as it is the simplest interface
- The commands we type in are tokenized (= split in strings, according to spaces) by the commandline interpreter, a.k.a. the shell
- The first token is the name of the program which should be executed, the others are passed as parameters
- The shell performs a `fork`, and its child `exec`s the program with the proper parameters

Under the hood

Process Tree Structure

- In a Linux system the processes are bound by a strict parent-son family relationship
- The boot process, after the kernel has bootstrapped the machine, yields the control to `init`[a]
- The `init` process generates all the other system process either directly (via `fork`, and `exec`s) or indirectly
- Every running process, except `init` has a father: it's the process which he was forked from
- Every process has a unique numeric identifier called Process ID (PID): on Linux it's represented as an 16 bit integer

[a] or SystemD, in case you are using it

Seeing processes

What is currently running?

- the first step in understanding what's going on in a system is looking at the processes running
- This can be done through the `ps` (process snapshot) command
- `ps` provides a list of the processes running, together with a couple of informations
- The output of the command can be redirected to a text file in the usual way (`ps > file.log`)
- A visual representation of the family tree of all processes can be obtained with `pstree`

Common ps options

Proper use of ps

- ps supports multiple syntaxes for the options, we will see the standardised one
- -e shows every process running
- -u <user> shows all the processes running as a certain user
- -Lf shows the number of threads of every process
- a shows the processes belonging to any user
- x allows to see processes which are not bound to a terminal

Interactive listing

A live view of the system

- `ps` provides a static list of the processes
- In a number of situations it is more helpful to see the evolution of the system state
- To this end, the `top` command provides a sequence of dynamic snapshots
- `htop` is a revised and enhanced version of top, still it is not the default tool
- Both tools periodically refresh the list of processes on screen, which can be sorted as you like

How do they work?

A(n old) system introspection filesystem

- All these tools have a common source for information : the `proc` filesystem
- It is a virtual filesystem which provides informations on all the processes running (and something more)
- It's existence is Linux specific, but other Unices provide equivalent mechanisms to access the same pieces of information
- When a program tries to list the contents of something in the `proc` filesystem, the OS generates these contents from scratch
- As the `proc` filesystem use is deprecated, these tools are moving to other ways to introspect the system

Process Inspection

Analyzing a live process

- We have seen how to obtain an overlook of the state of a system
- Up to now, the processes were (almost) black boxes
- Time to open the box and see what's inside
- This can be done via:
  - Debuggers (`gdb`)
  - Process tracers (`strace`, `lttng`)
  - File monitoring tools (`lsof`)

The GNU Debugger

- The GNU Debugger provides a plethora of functions to inspect the inner working of a program
- It acts through running the process under exam and tracing its behaviour via the `ptrace` system call
- It is able to alter the memory content of the program at the human debugger's will
- A detailed overview of the use will be presented in the next development tools lesson

Following the white rabbit : `strace`

- An alternative to per-instruction debugging is analysing the process at system call level
- Every process[1] needs to interact with the operating system
- It is possible to monitor the issuing and return values of every system call performed by a process
- Two tracing tools are available `strace` and `lttng`
- We will deal with `strace` as it is the most widespread one.

[1] or at least any process doing meaningful tasks

Following the white rabbit : `Strace`

- Follows the execution of a process and monitors syscalls, attaching to it via a `ptrace` call
- Offers a great way to see the big picture of a program behaviour
- `strace` by default prints out *all* the syscalls of a process
- Since they usually are a *TON* `-o <filename>` redirects to a file :)
- `-e=group` allows you to select only some syscalls relative to a peculiar function
  - `process`: syscalls concerning process management (e.g. fork)
  - `network`: syscalls concerning network (e.g. connect)
  - `file`: file read/write syscalls, fseek
  - `signal`: signal firing and masking calls

Following the white rabbit : Strace

- The `-p <PID>` options allows you to attach to a running process[2]
- The `-f` option enables the tracing of the child processes alongside the father
- The `-t` option prints out the system time at which the syscall has been run

[2] provided you have the permission to do so

An overlook on files

- One of the UNIX commandments states : "Under UNIX everything is a file"
- This means that the prime interface for data communication between kernelspace and userspace, and among processes are files
- This implies that all the physical devices are seen as a file by the programs in userspace
- Moreover, also sockets are seen as a peculiar type of file
- Although the syscall are often compatible, it is strongly advised not to mix them (e.g. use `write` instead of `send`) on a socket

An overlook on files

- A well designed file monitoring tool is a prime resource to understand what's happening
- The ultimate tool for file (i.e. mmapped devices, libraries, sockets and so on) monitoring is `lsof`
- The basic use just lists *all* the open files on a system
- Depending on the compile time options, `lsof` may list only the files of the processes owned by the user

Argh, too much info!

Ok, nice fireworks, but we'd like something more useful :

- the `-c <string>` option allows to list all the files opened by any command starting with `<string>`
- the `-c /<regex>/` option allows to list all the files opened by any command starting with `<regex>`
- the `+D` option allows to list all open files in a directory
- the `-u` option allows to list all open files of a certain user
- the options are usually combined with a logical *OR*
    - -a switches to *AND* combining

Not only files

Remember, "Under unix everything is a file":

- So we can also easily list open and listening sockets!
- the `-i @IP` option allows to list all the sockets open from-to a certain IP address
- the `-P` option prints numeric ports representations
- the `-p` option allows to list all open files from a precise PID
- the options may be reversed through prepending the usual caret symbol

Managing the running processes

- Up to now we have seen how to investigate the behaviour of a running system
- We did not interfere with it, we just observed what was going on
- This was done at system level (process tree examination) and at a finer grain (single process examination)
- We will now see how to manage the running processes

Signals

- The prime mechanism in a Unix system to communicate asynchronous information to a process are signals
- Signals can be though of as “software generated interrupts”
- Every process has a signal handlers table acting as the interrupt handler table
- The signal handler may choose to ignore the signal, do something or just fall back to the default action
- Usually the default action is the termination of the process

Signals

Here's a list of commonly used signals, together with the default behaviour:

- `SIGTERM` : terminates the process "gracefully" (file buffers are flushed and synchronized)
- `SIGSEGV` : terminates the process, issued upon a segmentation fault
- `SIGQUIT` : terminates the process dumping the memory segment into a `core` file
- `SIGKILL` : wipes instantly the process away from the system [unstoppable]
- `SIGSTOP` : sets the process in wait state [unstoppable]
- `SIGCONT` : resumes the execution of a process

The Unix flare gun : `kill`

- The commandline tool to send signals is aptly named ... `kill`
- Common syntax: `kill <signal> [options]`
- The signal to be sent can be specified either by its ID or its textual mnemonic
- The issued signals set flags in the fired signal table of the target process
- Since signals are resolved when a process is going to be run, `STOP` then shoot signals to die-hard processes
- Resume them with a `SIGCONT` and they'll be gone

Combining shell commands

- All the commands from the Unix shell follow the philosophy "do only one thing"
- By default they act on `stdin` and output the result on `stdout`
- You can chain commands through the use of the | character
- You can redirect the output of any command to a file using the > character
- An in-depth view on shell programming will be given further on in this course

Combined actions

- Due to a variety of reasons[3] a process may start spawning processes undefinitely (in jargon, a forkbomb takes place)
- Sending a `SIGTERM`/`SIGKILL` signal to each process by hand is annoying
- A combined action of `kill` and `lsof` makes an excellent forkbomb squad :
  - `lsof -t` outputs only the PIDs of the process owning the files (remember , libraries and mmaps are files :))
  - using a combination of shell expansion and `kill` allows you to wipe a clean slate of a lot of forkbombs

[3]Like, say, forgetting a `fork` call into a loop with a wrong termination condition

Eye of the beholder

- Watching over things is always important
- Sometimes it'd be useful to have a self refreshing command out of *any* command
- `watch` does exactly the tricks
- `-n <seconds>` specifies how often to refreshing
- `-d` highlights the changes from the last time (useful for waking you up)

Bottom line

- Managing the system will be important during this whole course
- A reasonable amount of skill in system management will save you way more time than the one you have invested in acquiring it
- When in doubt on something, do not fear to employ the system manual (available invoking `man <command>`)