MAGAZINE

BSD

FOR NOVICE AND ADVANCED USERS

EMBEDDED BSD

FREEBSD & ALIX

INSIDE

VOL.4 NO.5
ISSUE 05/2011(22)
1898-9144

10 Gig On Board

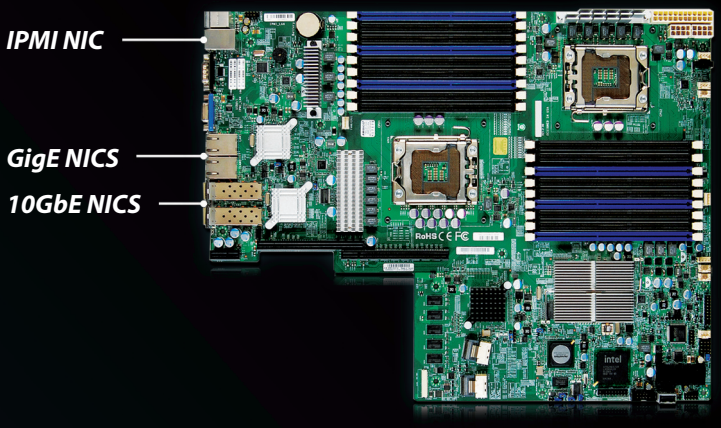
On-Board 10 Gigabit Ethernet Adapters leave your existing PCI-E slots available for other expansion devices.

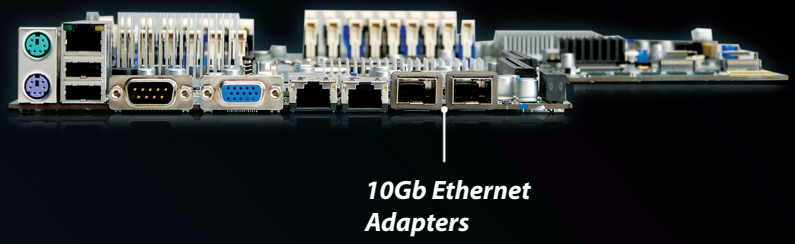
30% cost savings/port over equivalent Dual-Port 10 GB PCI Express add-on card solution

Blazing Fast, Embedded 10Gb Ethernet

10G Rackmount Servers in the iX-Neutron server line feature the Intel® Xeon® Processor 5600/5500 Series, and come with 10GbE networking integrated onto the motherboard. This eliminates the need to purchase an additional expansion card, and leaves the existing PCI-E slots available for other expansion devices, such as RAID controllers, video cards, and SAS controllers.

For more information on the iX-1204-10G, or to request a quote, visit: **http://www.iXsystems.com/neutron**





KEY FEATURES:

- Supports Dual 64-Bit Six-Core, Quad-Core or Dual-Core, Intel® Xeon® Processor 5600/5500 Series
- 1U Form Factor with 4 Hot-Swap SAS/SATA 3.5" Drive Bays
- Intel® 5520 chipset with QuickPath Interconnect (QPI)
- Up to 192GB DDR3 1333/1066/800 SDRAM ECC Registered Memory (18 DIMM Slots)
- 2 (x8) PCI-E 2.0 slots + 1 (x4) PCI-E 2.0 (in x8 slot -Low-Profile - 5.5" depth)
- Dual Port Intel® 82599EB 10 Gigabit SFP+ - Dual Port Intel® 82576 Gigabit Ethernet Controller
- Matrox G200eW Graphics
- Remote Management - IPMI 2.0 + IP-KVM with Dedicated LAN
- Slim DVD
- 700W/750W Redundant AC-DC 93%+ High-Efficiency Power Supply

Call iXsystems toll free or visit our website today!

1-855-GREP-4-IX | www.iXsystems.com

Powerful.
Intelligent.

Here it is!

The May issue of BSD magazine is out and full of new content. :)

We warm up with Michael Hernandez and his Introduction to Z Shell followed by the Developers Corner. There you will find Dru Lavigne's article about PC-BSD 9.0 Multiple Desktop Support, more DragonflyBSD news from Justin C. Sherrill (including information about new Dragonfly 2.10) and an article about „Evolution of an OpenBSD port" by Ian Darwin.
What will you find in this week How To's?
Same as the last year in May, this month's cover story is related to Embedded BSD. Bill Harris presents his work with using FreeBSD as the OS on Alix platform. Jared Barneck will show how to simplify application development on FreeBSD using Mono in the article of the same title.
Next you will find the sixth and unfortunately the last article from Rob Somerville's Drupal series followed by another Bill Harris How To: „Backups – Made Easy".
In the end of this section we will read how to fight DDoS attacks using PF from an article written by Matthieu Bouthors.
Then Darrel Levitch and James P. Howard II will show us some tricks and Sufyan bin Uzayr will „compare" BSD and GPL licences in the Let's Talk section.
Before we close the issue we will hear more about embedded software in Ryan Philips' „Allocating Dynamic Memory with Confidence" article.

I hope you will find all these articles informative and entertaining. Big thanks to all of our Authors, proofreaders and betatesters – their work is what makes this magazine better.

Thank you!
Zbigniew Puchciński
Editor in Chief
zbigniew.puchcinski@software.com.pl

MAGAZINE BSD

Editor in Chief:
Zbigniew Puchciński
zbigniew.puchcinski@software.com.pl

Contributing:
Michael Hernandez, Dru Lavigne, Justin C. Sherrill, Ian Darwin, Bill Harris, Jared Barneck, Rob Somerville, Matthieu Bouthors, James P. Howard II, Darrel Levitch, Sufyan bin Uzayr, Ryan Phillip

Proofreaders:
Corby Agid, Melanie Vonfange, Sander Reiche, Christopher J. Umina

Special Thanks:
Denise Ebery, Matt Olander

Art Director:
Ireneusz Pogroszewski

DTP:
Ireneusz Pogroszewski

Senior Consultant/Publisher:
Paweł Marciniak *pawel@software.com.pl*

CEO:
Ewa Dudzic
ewa.dudzic@software.com.pl

Production Director:
Andrzej Kuca
andrzej.kuca@software.com.pl

Executive Ad Consultant:
Karolina Lesińska
karolina.lesinska@bsdmag.org

Advertising Sales:
Zbigniew Puchciński
zbigniew.puchcinski@software.com.pl

Publisher :
Software Press Sp. z o.o. SK
ul. Bokserska 1, 02-682 Warszawa
Poland
worldwide publishing
tel: 1 917 338 36 31
www.bsdmag.org

Software Press Sp z o.o. 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*

The editors use automatic DTP system AUPUS

Mathematical formulas created by Design Science MathType™.

Get Started

Developers Corner

How To's

Tips & Tricks

Lets Talk

In business

Introduction to the Z Shell

In this modern age of computing, we are offered many choices with regard to how we might interact with our machines.

What you will learn…

- Some Unix and shell history.
- What the Z Shell (zsh) is.
- How to get started with zsh.

What you should know…

- A computer, preferably a BSD machine, but any will do.
- A means to install zsh on your local machine, or Internet access to sign up for a free shell account.

We could use a mouse to point at pictures and words in menus, or virtually press buttons on the screen. We could even use our fingertips on touch-sensitive screens to literally point at icons that represent what we want the computer to do. For those who cannot or prefer not to touch their machines, there is voice control available on many platforms, and there is ongoing research being conducted to implement methods of controlling computers with only our minds! It's astounding, really. These choices aside, there are those of us who prefer the tactile response and (usually) instant gratification provided by a command line interface, or CLI. You may already be familiar with the CLI. It's generally available on any BSD, including my favorite BSD derivative, Mac OS X. When using the CLI, you are presented with a shell prompt which is happily waiting there for your commands and willing to carry out your orders, but what is the shell, really? And what can it do for you?

This article will give you a brief introduction to shells in general, as well as introduce you to my favorite shell, the Z shell, more commonly known as zsh. For those of you who are new to the command line, all of the shell and Unix history might not make too much sense. Don't worry! The relevance of these now *ancient* shells and systems will become clearer soon enough. Understanding where you've come from can help you understand where you're going. If you are already well versed in Unix history, this entire article may be a rehash of things you already know. If that's the case, feel free to install zsh on your machine and get started: you won't need much help from me.

The term *shell* was introduced by Louis Pouzin. He created the first command-line interpreter, RUNCOM, as part of CTSS (*The Compatible Time Sharing System*) nearly a decade before Unix was created. Pouzin also worked on another time sharing system, named Multics (Multiplexed Information and Computing Service), and the term shell was coined to describe its command line interface. The first Unix shell was the Thompson shell, which was introduced in 1971 along with the first implementation of Unix, a replacement for Multics. According to Wikipedia, *The U in Unix is rumored to stand for uniplexed as opposed to the multiplexed of Multics, further underscoring the designers' rejections of Multics' complexity in favor of a more straightforward and workable approach for smaller computers*. Ken Thompson's shell (named `sh`) was quite limited by today's standards, but did include Input/Output redirection, among other basic features. People who worked with Unix had started to use it (and therefore Thompson's shell) for application development and scripting, and some began to find themselves constrained by its minimalism. In the mid 1970's, the Thompson shell was replaced by a shell written by Stephen Bourne and John Mashey, now known as the Bourne Shell. The Bourne Shell was a marked improvement from the original shell, because it added features which enabled it to be a fully programmable scripting language, as well as serve as an interface to the

users typing commands interactively at the terminal. Bourne also added environment variables to the shell, which meant that scripts could now have a context in which to run.

Later in 1970's another version of Unix came about, which originally shared the codebase of AT&T's Unix – the Berkeley Software Distribution, or BSD. Bill Joy, co-founder of Sun Microsystems and original author of the vi text editor, wrote the C shell (`csh`) which became the default shell for BSD Unix. Its shell scripting syntax was designed to be more like the C programming language with which Unix was written (and less like ALGOL, which is what the syntax of the Bourne shell was derived from.) The C shell also introduced features designed to improve the interactive experience, adding history (which allows users to recall and re-run previous commands quickly and easily), editing operations such as search & replace, aliases, job control and more. The C shell and its more modern incarnation, the Tenex C Shell (`tcsh`), are both considered harmful for shell scripting, however [1]. In the 1983, David Korn announced a new shell, which would become known as the Korn shell (`ksh`). Korn's shell included csh like interactive features, but retained the scripting language syntax of the Bourne shell. The Korn shell was designed not just to be a better shell, but to *enhance the UNIX tool kit by providing new and improved tools* [2]. In 1993, a more modern version was released (ksh93) which added more advanced programming features, such as associative arrays, while still retaining backwards compatibility with the Bourne shell. OpenBSD uses a version of the korn shell which is not the same as the original but is still `/bin/ksh`.

The Korn shell has served as an inspiration for other shells such as Bash (the Bourne Again SHell, which is the default on many systems, including most Linux distributions and Mac OS X), and Microsoft's Windows Power Shell (didn't expect a mention of Microsoft, did you?), as well as my favorite shell of all... the Z Shell.

The Z Shell was written in 1990 by Paul Falstad, a student at Princeton University. Paul had a professor named Zhong Shao, and decided that Shao's login name `zsh` would be a good name for a shell. Zsh is thought to be similar to the Korn shell, however it offers features from bash and tcsh as well. In fact users from just about any shell could find options in zsh to help them feel more at home. The Z Shell can emulate (basically *pretend to be*) the Bourne shell, as well as ksh. I find that zsh is at its best when not imitating other shells – it's a powerful shell packed with features. If you ask 10 zsh users why they use zsh, you may get 10 different answers. Some love zsh's extensive programmable command completion, some are impressed by the extended globbing (i.e., pattern matching. Glob is another bit of Unix history, dating back to the early 1970's), and some are intrigued by the zsh TCP/IP implementation and FTP client.

Conferences

BSDCan 2011

May 13-14
Ottawa, Canada

http://www.bsdcan.org/2011/

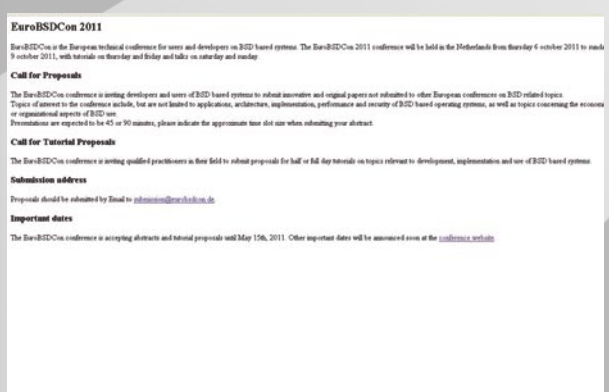
Open Source Business Conference

May 16-17
San Francisco, USA

http://www.eiseverywhere.com/

EuroBSDCon 2011

October 6-9
Netherlands



http://2011.eurobsdcon.org/CfP.html

That there are so many reasons to love zsh is not surprising, in fact zsh contains enough features to warrant its manual page to be split into over 15 parts. The Z Shell is definitely not designed according to any minimalist philosophy! It's got everything a shell user could want, and what is not part of the shell proper is (or can be) added as a contributed module.

I'm assuming that since you're reading BSD Magazine (thanks, by the way) that you have access to a BSD system. Zsh is open source and is available for all of the popular open source BSDs (FreeBSD, OpenBSD, NetBSD, etc.) and is shipped with Mac OS X as well. If for some reason you do not have a BSD available to use, I recommend a shell from SDF [3]. SDF, the Super Dimension Fortress, has been around since 1987 and provides free shell accounts as well as a host of other services. Their mission is *... to provide remotely accessible computing facilities for the advancement of public education, cultural enrichment, scientific research and recreation. Members can interact electronically with each other regardless of their location using passive or interactive forums. Further purposes include the recreational exchange of information concerning the Liberal and Fine Arts*. Their shells run on *a network of 8 64bit enterprise class servers running NetBSD*, and zsh is available there.

If possible, the best way to get started with the Z Shell is to install the latest development version on your local machine. As of this writing, the most recent version is 4.3.11 (with 4.3.12 soon to come). While the 4.2.x branch is still listed as the most recent stable branch, many (if not most) of us are using the 4.3.x development branch and find it stable enough for daily use. I'm going to leave the installation of zsh up to you, as it is available on many different systems and each has its own method of package installation (ports, packages, etc.) If you have a recent Apple computer, zsh will already be installed and located at `/bin/zsh`. I recommend you install the latest zsh via fink or macports, you'll find that it is most likely a more recent version than the one installed already. Once you have zsh installed, you can open your favorite terminal emulator (`xterm`, `Terminal.app`) or you can log into SDF with putty if you happen to be on Windows. I recommend you start by entering the following into your terminal window:

```
echo $SHELL
```

and press enter or return. You should see something like `/bin/ksh` or perhaps `/usr/pkg/bin/bash`. This command *echoes* or displays the contents of the environment variable named SHELL for you on the screen. It most likely will not display zsh, because zsh is very rarely the default shell on a system. Unless you have a shell setup that you are absolutely married to, I recommend changing your default shell to zsh immediately. You may need to know the full path to zsh, which you can find by checking the contents of `/etc/shells`. You can view the contents by entering:

```
cat /etc/shells
```

Once you know the full path to zsh, you can begin the (possibly life changing!) experience of changing your default shell to zsh by entering the following:

```
chsh
```

and pressing return. You will be asked for the shell you want to be your new default, and you will be asked to enter your password. Now I recommend you get acquainted with the zsh manual. The manual for zsh is quite large. As I said above, it's broken up into over 15 parts. You can see a list of these parts by simply entering:

```
man zsh
```

I recommend that you begin with the zsh roadmap that is shipped with recent versions of the shell. You can access it by entering:

```
man zshroadmap
```

As this is an article for BSD Magazine, and not a book, I'm going to stop here for now. There is so much I'd like to share about zsh and shells in general, but I'll have to save that for future articles. If you're too excited to wait for my next BSD Magazine submission, you can find a wealth of zsh information on the web (you can start with the official zsh page [4], and there is a great book available by Oliver Kiddle, Jerry Peek, and Peter Stephenson named *From Bash to Z Shell* [5], which offers the most comprehensive zsh coverage available in print form today. It is a few years old, but it well worth picking up, especially if you're just starting on this `$PATH`.

On the 'Net

[1] *http://www.faqs.org/faqs/unix-faq/shell/csh-whynot/*
[2] *http://slashdot.org/story/01/02/06/2030205/David-Korn-Tells-All*
[3] *http://sdf.lonestar.org/*
[4] *http://zsh.sourceforge.net/*
[5] *http://www.bash2zsh.com/*

MICHAEL HERNANDEZ

Mike is an IT consultant and web programmer. He lives in Brooklyn, New York, and he and his wife are celebrating their one year anniversary on February 14th. He also loves electronic dance music and commuting on his fixed gear bike, appropriately named Constance.

Supporting Multiple Desktops in PC-BSD 9.0

Beginning with version 9.0, PC-BSD will allow the selection of multiple desktops during installation. This article describes what changes were needed to allow for multiple desktop support and how you can help the PC-BSD project in this endeavour.

When the PC-BSD project was started in 2005, its goal was to provide an easy-to-use desktop experience. KDE was chosen as the default desktop as it was well known, easy to learn, and provided a suite of useful applications. The PC-BSD project also created a suite of custom graphical utilities to address missing functionality not provided by KDE–these PC-BSD utilities understand BSD device names and were integrated into KDE's menus. This made for a seamless user experience but did cause some confusion as to which functionality was provided by KDE and which was provided by PC-BSD.

In addition to KDE, Fluxbox was installed for users with older hardware or who preferred a lighter weight desktop environment. Over time, PBIs for GNOME, XFCE, and Enlightenment were created so that users could install these alternate desktops using PC-BSD's Software Manager.

As the PC-BSD userbase grew, it became obvious that many users did not like KDE and preferred other desktop environments, such as GNOME, or preferred a light-weight window manager other than Fluxbox. Further, installing an alternate desktop as a PBI was not ideal as it did not integrate with the PC-BSD utilities, making for a sub-optimal user experience. It became clear that the advantages of providing one supported desktop were being outweighed by the disadvantages of being forced to use a desktop one did not enjoy using.

Making the Necessary Changes

In order to integrate with multiple desktop environments, the PC-BSD utilities had to be de-coupled from KDE. This required a complete overhaul of nearly all of the PC-BSD tool-chain and the PBI format itself. The configuration tools have since been converted into pure shell or QT4, and are window-manager independent, helping to provide a consistent user experience regardless of the desktop being used. The PBI format has also been re-written with 100% command-line functionality in shell and can even run on native FreeBSD without an installed desktop.

Next, a Control Panel was created. The Control Panel will automatically hook into any of the desktop environments chosen during the installation. This means that users can easily find the graphical PC-BSD utilities which are used to manage their system and that those utilities will be available, regardless of the desktop the user has logged into. Figure 1 shows a screenshot of the Control Panel as it appears today. Additional utilities may be added to the Control Panel by the time PC-BSD 9.0 is released later this year.

pc-sysinstall, the installation utility used by PC-BSD, was also modified to allow for the selection of desktops and other system packages during installation. Figure 2 shows a screenshot of the installer's Desktop Selection screen.

Supported Desktops

One of the criteria in determining which desktops to include in the installer was XDG-compliance. XDG (*http://en.wikipedia.org/wiki/Xdg*) is an inter-operability standard for desktop environments that run on top of the Xorg window system. XDG-compliance allows for tight integration, making it possible to include the same default wallpapers, desktop icons, menu entries, etc. across multiple desktops.

The PC-BSD 9.0 installer allows you to select from the following XDG-compliant desktop environments. Most of these environments allow you to select which components (e.g. accessibility, development, games, etc.) to install with the base desktop. After installation, one can install/uninstall desktop components using *Control Panel -> System Manager -> System Packages*.

KDE4

KDE (*http://www.kde.org*) provides a complete desktop environment that includes hundreds of applications. It supports desktop effects, scalable graphics, easy access to network resources, localized menus, accessibility features, and a fully customizable environment. It provides a netbook desktop theme (available in *System Settings->Workspace Appearance->Desktop Theme*) to provide a *lighter* version suited to netbook hardware. It also has a large selection of themes, screensavers, and utilities created by the community and available from *http://kde-apps.org/*.

Figure 1. *PC-BSD Control Panel*

GNOME2

GNOME version 2 (*http://www.gnome.org*) also provides a complete desktop environment that includes 100s of applications. It supports desktop effects, localized menus, accessibility features, and a customizable environment. It is lighter weight than KDE4, making it suitable for netbooks.

Note

GNOME3 is currently being ported to FreeBSD. If the port is mature in time for the release of PC-BSD 9.0, it will be included as a desktop option.

LXDE

The *Lightweight X11 Desktop Environment* (*http://lxde.org*) is a fast and energy-saving desktop environment. LXDE provides multi-language support, standard keyboard short cuts and tabbed file browsing while using less CPU and less RAM than other desktop environments. LXDE will be the default desktop on the CD and live version of PC-BSD 9.0.

XFCE4

XFCE (*http://xfce.org*) is a lightweight desktop environment that aims to be fast and low on system resources, while still being visually appealing and user friendly. XFCE uses modular components that are packaged separately,

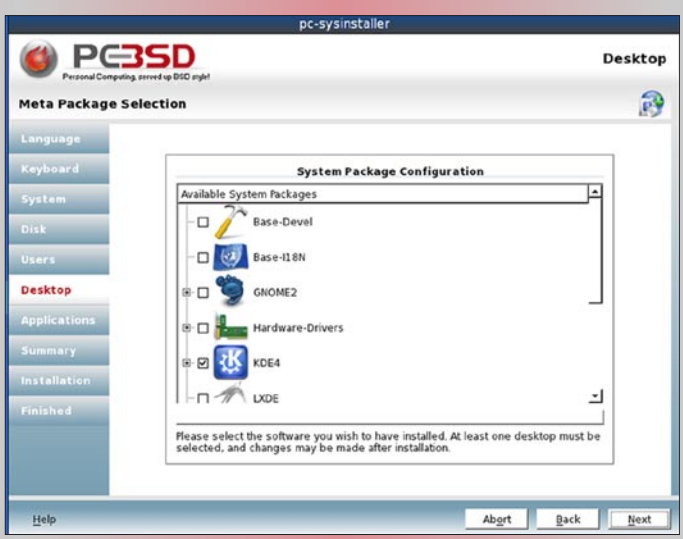


Figure 2. *Desktop Selection Screen of PC-BSD 9.0 Installer*

Resources

PC-BSD Forums: *http://forums.pcbsd.org*
PC-BSD Mailing Lists: *http://lists.pcbsd.org*
#pcbsd on IRC Freenode

allowing you to install the packages you wish in order to create the optimal personal working environment. You can find the modules that have been ported to FreeBSD/PC-BSD by searching for *xfce* at freshports.org.

Unsupported Desktops

The unsupported desktops category includes window managers that are typically used by power users. These are light weight environments that may require the user to start applications from the command line or modify configuration files in order to customize the desktop. These desktops are not XDG-compliant, meaning that they do not pre-load the PC-BSD desktop icons or menu items. However, they will include the PC-BSD wallpaper and pointers to Control Panel and AppCafe (the PC-BSD 9.0 application installer).

The following unsupported desktops are available for selection during and after the installation of PC-BSD 9.0:

Awesome

Awesome (*http://awesome.naquadah.org/*) is a highly configurable, framework window manager. It is extremely fast, small, dynamic and heavily extensible using the Lua programming language. A well documented API is used to configure and define the behaviour of the window manager. No mouse is required as everything can be performed with the keyboard.

IceWM

The goal of IceWM (*http://www.icewm.org/*) is speed, simplicity, and not getting in the user's way. IceWM can be configured from plain text files and has an optional, built-in taskbar with menu. It has been localized and additional themes are available from *http://box-look.org/*.

Window Maker

Window Maker (*http://windowmaker.org/*) includes a graphical tool called Wprefs which can be used to configure the desktop. By default, there is no taskbar and applications are accessed by right-clicking the desktop. Window Maker provides a number of dockable applications known as dockapps. Many dockapps are available in the FreeBSD ports/packages collections and you can find these by doing a *Short description* search for *windowmaker* at *freshports.org*.

Working with FreeBSD Porters

The desktops that are used by PC-BSD are made available thanks to the hard work of many FreeBSD port committers who *port* the source code so that it installs and works on FreeBSD/PC-BSD systems. The larger desktop projects have porting teams: KDE (*http://freebsd.kde.org/*) and GNOME (*http://www.freebsd.org/gnome/index.html*). The other desktops have one or two individuals who are responsible for maintaining the port of the desktop.

iXsystems, the corporate sponsor of the PC-BSD project, has donated several build environments to assist the FreeBSD desktop porters in their work. These build environments are for the KDE, GNOME, and Xorg porting teams, allowing the porters to use speedy hardware to collaboratively build and test their ports. The build environments run tinderbox (*http://tinderbox.marcuscom.com/*), a set of scripts for creating binary packages for multiple platforms and architectures, and for testing new ports, port upgrades, dependencies and packing lists.

Providing the build environments not only helps the porters, it also helps the PC-BSD community as new desktop changes are incorporated into testing snapshots. This allows testers to try out and provide feedback on the changes. The PC-BSD forums includes a Testing category (*http://forums.pcbsd.org/forumdisplay.php?f=64*) where users can provide feedback on their particular desktop. Ports committers subscribe to their desktop's forum and can respond to user feedback.

How You Can Help

Going from one supported desktop to many supported desktops is a major change for PC-BSD and we expect to find many usability bugs in this process. For this reason, 9.0 will have a testing period of over 6 months with bi-weekly testing snapshots. Snapshots are announced on the PC-BSD blog (*http://blog.pcbsd.org*) as they are released and users are encouraged to try a snapshot and provide feedback on the PC-BSD testing mailing list (*http://lists.pcbsd.org/mailman/listinfo/testing*). Since these are testing snapshots, we recommend that you install them in a virtual environment such as VMware or VirtualBox or on a test system that is separate from your main computer.

We need as many people as possible to try different installation scenarios (selecting a single or multiple desktops) and to poke about and try to use the various menus that come with the desktop. Finding and reporting error messages, missing applications, broken links, and other unexpected behaviour during the testing period means that they can be fixed before PC-BSD 9.0 is released, which in turn maximizes the user experience for everyone.

The PC-BSD Handbook is also being updated in preparation for the 9.0 release. The Handbook is a collaborative effort that happens on the PC-BSD wiki (*http://wiki.pcbsd.org/index.php/PC-BSD_9_Handbook*). Users are encouraged to read the existing Handbook entries for their favourite desktop environment(s) and to add information that would be useful to users new to that desktop environment. Any changes to the wiki are sent to PC-BSD community members who volunteer as editors. This means that you don't have to be a great writer or a native English speaker to contribute documentation–the editors review your changes and can edit them for grammar and readability.

Conclusion

PC-BSD has a vibrant community that is responsive to user feedback. Many of the changes that are being made for PC-BSD 9.0 are in response to user requests for changes in the default desktop. Readers are encouraged to participate on the forums, mailing lists, and IRC channel so that others can benefit from their PC-BSD experience.

DRU LAVIGNE

Dru Lavigne is author of BSD Hacks, The Best of FreeBSD Basics, and The Definitive Guide to PC-BSD. As Director of Community Development for the PC-BSD Project, she leads the documentation team, assists new users, helps to find and fix bugs, and reaches out to the community to discover their needs. She is the former Managing Editor of the Open Source Business Resource, a free monthly publication covering open source and the commercialization of open source assets. She is founder and current Chair of the BSD Certification Group Inc., a non-profit organization with a mission to create the standard for certifying BSD system administrators, and serves on the Board of the FreeBSD Foundation.

DragonFly News

Google Summer of Code progress

DragonFly has 6 slots for Google Summer of Code, with projects like kevent fixing, mirroring for the device mapper, new disk scheduling, a port of PUFFS, and more approved. We received far more interesting projects than we had available slots, which is both good and bad. Work will continue through the summer.

Pkgsrc progress

The first quarterly release of pkgsrc for 2011, pkgsrc-2011Q1, was tagged at the start of April. It includes updates to GNOME and KDE, along with many other updates. KDE 4.4 should be able to build without modification, or close to it, on DragonFly.

This release has close to 11,000 packages; packages for DragonFly have been built, but only for the next release. That leads naturally to the next topic:

The DragonFly 2.10 release and performance

The next release of DragonFly will be happening as this issue of BSD Magazine goes to press, so to speak. This release has removed almost all the old giant locking mechanisms in the system, except for one large one for the Virtual Memory system. DragonFly has switched mostly to token use, and is one of the few non-academic operating systems to use a primary sychronization mechanism that is not a blocking mutex.

There has been a definite speed upgrade to the 2.10 release, which can be seen in these series of graphs. Jan Lentfer ran a pgbench test on a 1GB database using a DragonFly 2.8 system, then ran it again on the same system upgraded to a recent DragonFly 2.9. The benchmarking system had 2G of RAM so the database activity was entirely within system memory.

I built a graph showing the difference in sysbench results between DragonFly 2.6, 2.8, and 2.10 (which had been tagged but not released at the time I made the graph.)

Jan Lentfer graphed the performance difference using PostgreSQL. This was with a 5.6G database on a system with 2G of RAM, and an atom 330 processor. This test also measures I/O speed since the database size was almost 3 times available RAM.

DragonFly has swapcache, the ability to cache disk information to a faster disk device. This is most useful if you have a SSD added to the system; swapcache will put the disk cache information for all your attached drives onto that fast device, making all disk accesses for cached info as fast as your fastest device. This is especially useful when the amount of data in use is greater than available memory.

If the previous graph wasn't already good news, Matthew Dillon made some changes to how DragonFly

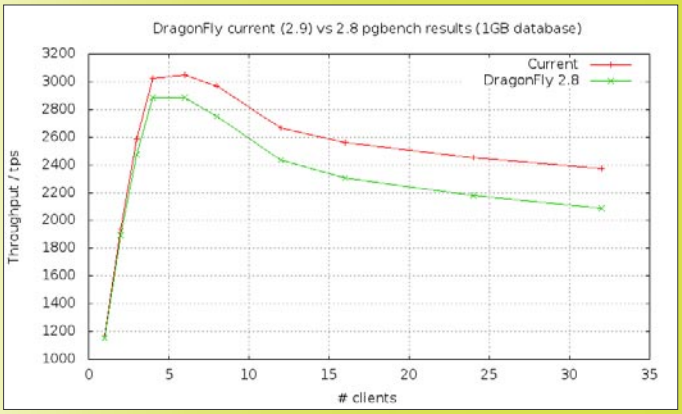


Figure 1. *pgbench improvements from DragonFly 2.8 to 2.10, in-*

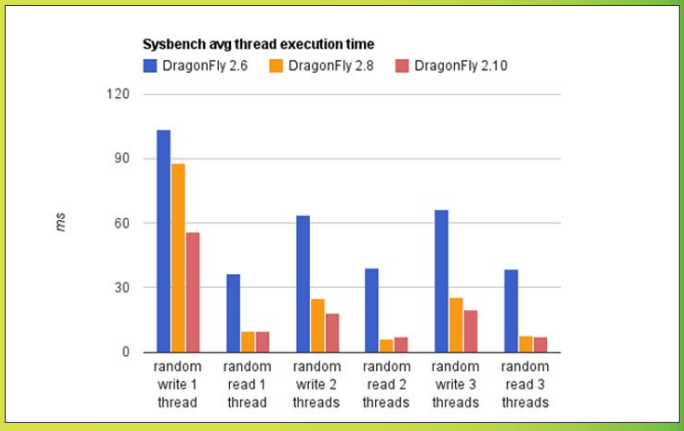


Figure 2. *sysbench results for DragonFly 2.6, 2.8, and 2.10 (shorter bars are better)*

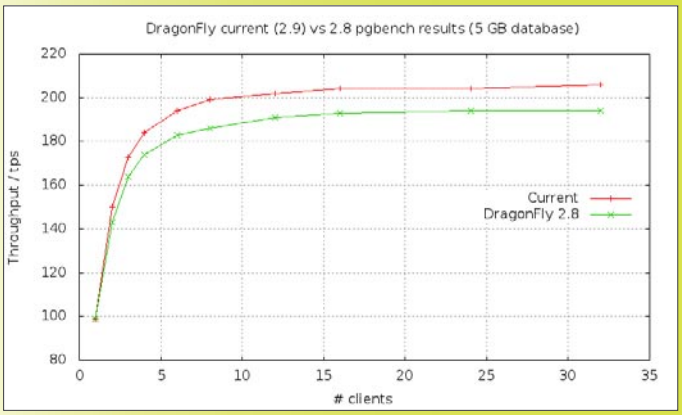


Figure 3. *pgbench improvements from DragonFly 2.8 to 2.10, database larger than RAM*

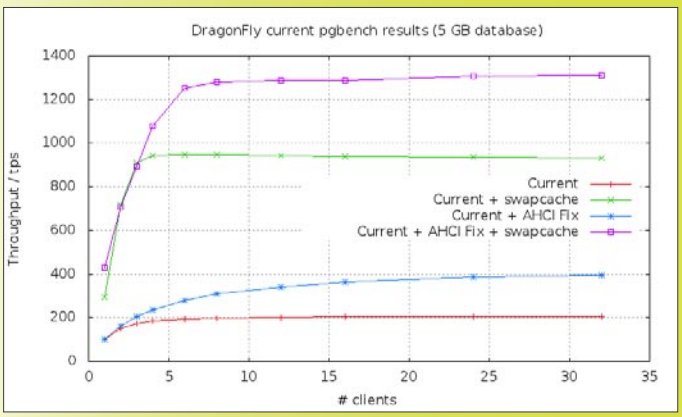


Figure 4. *pgbench improvements again from DragonFly 2.8 to 2.10, plus swapcache*

handles AHCI, utilizing all 32 tags, which made an even larger difference in performance, with fantastic results. Note that this is an expansion of the previous graph.

This upcoming release looks to have some excellent speed improvements. It will also include the recent deduplication work for Hammer, meaning full disk deduplication happens as a batch process overnight, and a *live* cache for regular disk activity, also called *fast cp*. Initial reports have it working well on hundreds of gigabytes of data with little memory usage; developer Venkatesh Srinivas reported success with only 256M of RAM available.

Graphs compiled with assistance from Jan Lentfer and Alex Hornung. Details on testing hardware are available at *http://www.shiningsilence.com/dbsdlog/2011/04/12/7586.html*.

JUSTIN C. SHERRILL

Justin Sherrill has been publishing the DragonFly BSD Digest since 2004, and is responsible for several other parts of DragonFly that aren't made out of code. He lives in the northeast United States and works over a thousand feet underground.

Evolution of an OpenBSD Port

In this article I'll talk about the evolution of the OpenBSD port of radicale (http://www.radicale.org/), a nice small, simple CALDAV-based calendar server written in Python by Guillaume Ayoub.

It's not a presentation about the evolution of the OpenBSD ports mechanism itself (see References for a paper on this), nor is it a how-to on porting software (again, see the References). But just a look at how and why certain changes were made to the port, as part of a study of the overall operation of ports development on OpenBSD at this time.

I am listed as maintainer of a couple of dozen OpenBSD ports. To be listed as maintainer means you have to be aware when the *upstream* author/maintainer issues updates, so the OpenBSD users can get them (assuming they are good; you also have to test). It also means you have to keep the port up-to-date when the ports mechanism changes, deal with dependencies on other ports, and so on. In other words, maintain the port!

Some of my ports are fairly active, like JOSM, a Java editor for *OpenStreetMap.org* maps, which is updated constantly in Subversion, and releases are declared stable every month or so. At the other extreme, some of my ports are for software from the beginning of Unix's open source days that are no longer maintained by their originators, like *spiff*, a slower but more thorough variation on Unix diff(1). Spiff was written in the mid-1980's by Daniel Nachbar at Bellcore (Bell Communications Research, an AT&T spinoff that went through many corporate changes over the years). I have lost touch with Mr. Nachbar and, as far as I know, no maintenance has been done on the software in years, perhaps decades. But it still works, so we keep it in-tree. When software stops working, isn't maintained *upstream* and can't be fixed by the ports maintainer, then we remove the port altogether.

I start many ports, and some of them make it into the tree and some do not. OpenBSD's ports mechanism understands this, and allows you to create your own hierarchy under `/usr/ports/mystuff` (I use a lot of shell variables as path shortcuts, so I call this `$mp`, for *my ports*). Under `$mp` you create a subset of the full hierarchy, e.g., for radicale, which is considered a productivity application, `$mp/productivity/radicale` becomes the staging directory for the port. Once it is completed and OK'd by another ports developer, it is imported into CVS and then checked out in its official location, which would be `/usr/ports/productivity/radicale`. Except I never got that far. Sergey Bronnikov submitted another port for radicale, and his was more complete than mine. I realized it was time to `rm -r $mp/productivity/radicale`, and start contributing to the new improved version.

The new port was picked up by Stuart Henderson, one of our more active ports maintainers, who commented on some improvements that would need to be made before the port could be imported into CVS. He actually provided most of the improvements (he's not just talk!), and so I tried building and installing his version.

Before too long I had a working set of calendars. I had previously used KDE's korganizer, so I had several calendar files, which I simply copied to `/var/db/radicale/calendars/ian/` and changed their ownership and group

(each port gets its own distinct userid and group, a standard *privilege separation* mechanism. After restarting the daemon (`/etc/rc.d/radicale` restart), I installed the Lightning add-on to Mozilla Thunderbird (there's a port for that) and instantly had working calendars. Good start, I thought.

At around this point Stuart imported this version of the radicale port into our tree.

Then I tried engaging the authentication mechanism. No dice – it all worked except for the part where Lightning should ask for a password, which never happened: I could still get at my calendars. After pondering the configuration file and the basic documentation on the web site, I contacted the upstream author. He was unable to replicate this problem. But I had copied Stuart on the mail I sent upstream, and he apparently pondered better than I. He had patched the source to use `${SYSCONFDIR}`, but not run the substitute command. He committed a fix to the Python source that made it work, and also put a comment in the configuration file showing how to use the `htpasswd` command. I updated to his version of the port and tried again. Now I could only access my calendars with a password – definitely an improvement from a security point of view, but not the final story, yet.

Then I tried engaging the encryption (SSL). Since I already had a web server certificate, I simply pointed the Radicale configuration file (`/etc/radicale/config`) at the files in `/etc/ssl` and `/etc/ssl/private`. No dice. Thought about this some more. Finally ran the server from a console and saw the stack trace: permission denied on `/etc/ssl/private/server.key`. One good head-bonk later, and a few keystrokes to copy the files and chown them, and I had working encryption. But our goal is not just to get a port working for ourselves, but to make it easy for end users. I wrote a draft README file and sent it to Stuart, and we discussed it by email. Should we put a note in the MESSAGE file which is displayed when the port is installed? Or in the pkg-readme which is longer but not displayed automatically? Finally we agreed to put a note in the README file telling the user how to edit the *config* file to enable encryption, reminding people about this issue. I had overlooked the issue because many of the ports do their own privsep: they start as root just to open such files, then setuid/setgid to the respective userid. But most Python-based ports do not work this way, including radicale. So the comment in the readme file got committed (as revision 1.3), hoping to make it easier for anyone else to get it right first time.

One limitation is that Radicale itself does not (yet) offer per-user ACLs (access control lists), so you can't offer some but not all of your online calendars to another user on the system. Radicale's web site claims it does not intend to be a 100% implementation of the CALDAV spec, so this feature may or may not be added someday. However, for a small, mutually-trusting user community with `personal = False` it offers a good shared calendar mechanism. For a large, untrusted community you should install it with `personal = True` to limit each user to their own calendars.

Of course, I did all this work on my *test server* a.k.a. my laptop. Since so much polishing had been done at this stage, installing and getting it running on the real server consisted only in the following:

1. sudo `pkg_add -v` radicale
2. sudo vi `/etc/radicale/config #` to enable ssl and authentication, as described
3. Install a password for each user with htpasswd
4. sudo `/etc/rc.d/radicale start #` to start the server
5. Connect and enjoy!

Radicale is a nice, simple server for CALDAV calendar clients. It thus offers a good-sized sample of what is involved in preparing and *tuning up* a port/package to make it easy for end-user installation. It is now available for use as described here on OpenBSD -current, and will be in the next stable release.

References

- Evolution of OpenBSD Ports: interview in 10 Years of PkgSrc, *http://www.netbsd.org/gallery/10years.html#espie*
- Ian Darwin's maintainer list at OpenPorts, *http://openports.se/search.php?stype=maintainer&so=ian%40openbsd* (and similarly for any other maintainer, just change the email address)
- Spiff technical paper at *ftp://ftp.cyberway.com.sg/pub/funet/unix/security/docs/usenix/usenix/summer88/spiff.ps.gz*
- Radicale website: *http://www.radicale.gz*
- OpenBSD porting guide: *http://www.openbsd.org/porting.html*

IAN DARWIN

Ian Darwin is an OpenBSD committer who lives in the country well north of Toronto, Canada. He runs *NIX on just about all his computers; he once said that his only Windows looked out over the hillsides where he lives.

BSDCAN 2011

THE BSD EVENT OF 2011
http://www.bsdcan.org/

Ottawa, Canada

BSDCan 2011 – The event to be at this year

BSDCAN 2011

There's only one major BSD Event in North America in 2011: BSDCan

WHERE

Ottawa, Canada

WHEN

Early May 2011, with two days of tutorials before the conference. Exact dates to follow.

WHO

Contributors, developers, and users

VENUE

University of Ottawa
http://www.uottawa.ca/

AT FEES YOU CAN AFFORD

We plan to keep costs to a minimum. As such, the conference will be held at University of Ottawa and accommodation is available within the University residences. Hotels are also within close walking distance of the conference venue.

WHAT DOES IT COST?

Type	CAD
Regular	$195
Corporate	587
Additional Corporate	$120
Student	57
Tutorial	57

Comfortable accommodation is available on campus at very reasonable rates. See our website for details.

Take the BSDA Certification exam.
For details see
http://bsdcertification.org/

SCHEDULE OVERVIEW

Wednesday

4:00 pm	Sign-in desk opens at a local pub. Get your registration pack and have a drink.
8:00 pm	Sign-in desk closes.

Thursday

9:30 am	Opening words
11:00 am	First set of talks
12:00 pm	lunch
1:00 pm	Second set of talks
2:00 pm	break
2:30 pm	Third set of talks
3:30 pm	break
4:00 pm	Fourth set of talks
5:00 pm	Key Signing Party

Friday

10:00 am	First set of talks
11:00 am	break
11:30 am	Second set of talks
12:30 pm	lunch
1:30 pm	Third set of talks
2:30 pm	break
3:00 pm	Fourth Set of Talks
4:00 pm	Fifth Set of Talks
5:00 pm	Closing words

Sat

8:30 am	Breakfast
9:30-4:00	Tourist fun

TALKS FROM 2010

Please see the website for complete details.

- ClangBSD - Replacing gcc with clang
- Consideration for the BSD Professional Exam
- Security Implications of the Internet Protocol version 6 (IPv6)
- Puffy At Work -- Getting Code Right And Secure, The OpenBSD Way
- Everything you need to know about cryptography in 1 hour
- Networking from the Bottom Up: IPv6
- Porting dummynet to Linux and Windows
- Journaled Soft-Updates
- Porting hwpmc to non x86 platforms
- Maintaining a Customized FreeBSD Distribution
- Debuggers - Architecture and Implementation
- pfSense 2.0
- Networking from the Bottom Up: Ipv6
- The New VVorld
- A new packet scheduling architecture for FreeBSD

SOCIAL ACTIVITIES

It's not all work. Social activities play a major role in project development.

Wednesday

4:00 pm Drinks + registration at a local pub

Thursday

4:30 pm BOFs
7:00 pm Gathering at local eateries for dinner

Friday

4:30 pm Key signing party
7:00 pm Gathering at local pubs for drinks

Saturday

8:30 am Breakfast

afterwards: various tourist-type things

To stay informed, please join our announcement mailing list. Details at http://www.bsdcan.org/

2010 PLATINUM SPONSOR

OUR 2010 GOLD SPONSORS

FreeBSD & Alix

A pint sized install of an Enterprise OS

The embedded device or Single Board Computer (SBC) market has for the most part, been dominated by variety of Linux derivatives.

What you will learn…

- Can FreeBSD be used successfully in the embedded device market?
- Which type of embedded devices work best for FreeBSD
- How to install FreeBSD 8.2 on x86 based embedded SBC
- Benefits and limitations of Embedded platforms
- How to determine which applications are best suited for embedded platforms

What you should know…

- How to configure FreeBSD networking from CLI prompt
- Configuration of applications
- Edit system files with vi or other text editors
- Setup a serial console and use a terminal emulator (minicom)
- Install packages from the CLI prompt

Today, some version of Linux can be found running on everything from toasters, to cameras. Much of Linux success in this area began with OpenWRT and the early LinkSys WRT54G routers. As long as it had some RAM, and some flash for storage, a Linux operating system could usually be found on it. Usually, this was a stripped down kernel, a feature limited tool set, due to the constraints of storage and memory found on the router like platforms.

FreeBSD has always had an excellent reputation as a server grade operating system. It is not as widely known as an OS for embedded devices.. The reasons are many, mostly having to do with kernel support for the various hardware platforms and processors. Today, one will still find FreeBSD primarily targeting Intel/AMD x86/x64 based processors, along with a few ARM based platforms.

The embedded hardware market has been undergoing dynamic changes over the last few years, with SOC

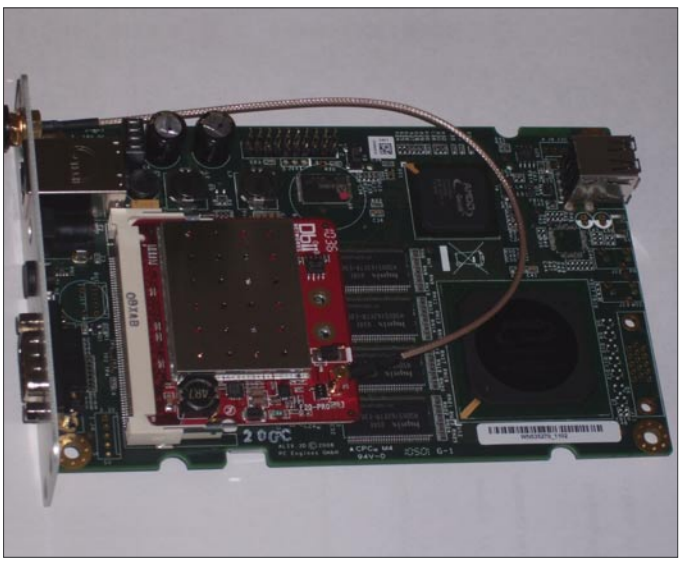


Figure 1. *Alix 3 CPU side with mini-pci B/G radio*

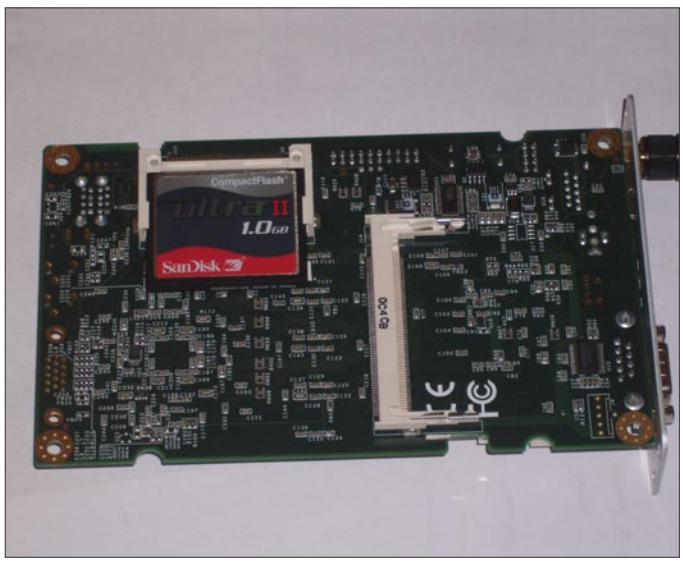


Figure 2. *Alix 3 SBC CF and Mini-PCI slot*

(*System On Chip*) and ultra low-power processors. An example is the AMD Geode LX800. A fully Intel x86, compatible processor, running at 500 Mhz, drawing less than 1 watt of power. Alix has taken this processor and combined it with a small 100x160 mm board to form a quite powerful *single board computer* (SBC). The combined power diet for this tiny board is less than 5 watts.

How viable is FreeBSD today, on such devices? What applications could be used on low power devices like the Alix and others? That's is what we will look at next.

Getting Started

To prepare for this article, I contacted the good folks at *TitanWireless, LLC*, in Austin Texas, who shipped me an Alix 3 board, equipped, according to my wishlist. The unit came configured with:

- AMD Geode LX800 Processor, 500 Mhz
- 256 Megs of RAM
- CF Flash (1 gig in test unit)
- LAN – 1
- DBII F20-PRO B/G miniPCI radio card
- Aluminum case, antenna and power supply

The Alix 3 series SBC was chosen because of it's wide range of configuration options, availability and cost. It can run nearly any modern x86 operating system, (given memory and disk limits). The company and their suppliers provide excellent support as well. Their support of FreeBSD and Linux was of primary consideration.

My first concern was getting the latest production release of FreeBSD (8.2-RELEASE) onto the flash card and able to boot. This turned out to be a straight forward and simple process, once I had scrounged up a serial cable to use for the console.

Install FreeBSD to Flash

There are several ways, (PXE, USB-CDROM, etc) but simply using a common multi-format flash card USB reader, and a host PC or server running FreeBSD worked very well. Following the example documentation (for the most part) from this site, *http://www.freebsdonline.com/content/view/589/506/* was great, with only a couple changes. Pay attention to the device names (you don't want to re-install over any existing hard drives).

Suggestion: Install the *Minimal Install* as space is a premium on flash storage.

When the Alix boots, pressing [s] during the memory test will let you change the serial port speed to 9600 (do this!). It defaults at first to 38,400 baud.

FreeBSD 8.2 changes the serial port names, so where they mention editing `/etc/ttys` and changing `ttyd0`, find the line that starts with `ttyu0` and make it look like this in step 3:

```
ttyu0 „/usr/libexec/getty std.9600” vt102 on secure
```

That's it. FreeBSD is installed, and it should boot right up to a login prompt.

Normal configuration such as network interfaces, and such can be done via `/etc/rc.conf`. The LAN interface is `vr0` and the Atheros wireless device is available on `ath0`. My `/etc/rc.conf` looked like:

```
hostname=”alix3”
sshd_enable=”YES”
sendmail_enable=”NONE”
ifconfig_vr0=”inet 192.168.1.32 netmask 255.255.255.0
                    broadcast 192.168.1.255”
defaultrouter=”192.168.1.1”
```

The first steps, just like building any server, is to create a regular user, setup networking, with DNS. Then install a few tools to make like life nicer..

```
# pkg_add -r bash screen
```

Installation of applications is as simple as running the appropriate `pkg_add -r` …. Keep a close eye on remaining storage space.. eg;

```
# pkg_add -r apache22
 # pkg_add -r lighttpd, etc.
```

This left the system with around 500 megabytes of flash storage (out of 1 Gig) useable and approximately 220

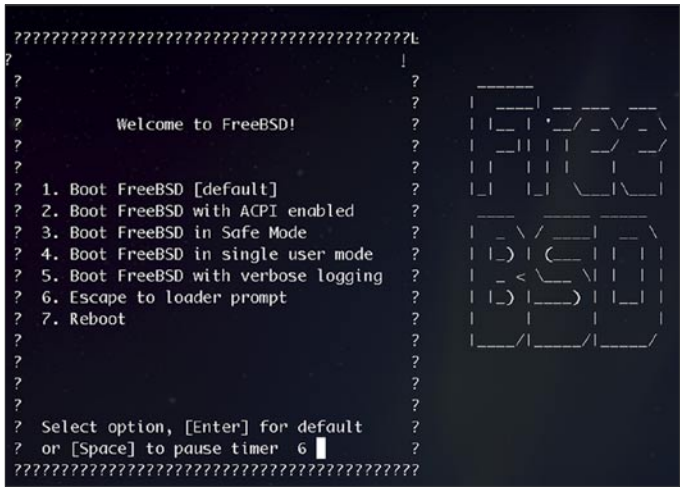


Figure 3. *Alix FreeBSD Boot screen over serial console*

megabytes of available memory for applications. Fifty megabytes where allocated for swap space. It bears noting, that this is a complete (albeit minimal) FreeBSD operating installation.

Compact flash has become relatively inexpensive, as SD Flash is replacing it as the favorite for digital camera storage. It's possible to find CF (4gig/$20, 8gig/$40), which makes other applications viable on the Alix.

Applications

The biggest issue to determining what applications *fit* well on an embedded device is available resources, whether they be memory, cpu, or storage (flash). Evaluating what applications would benefit the most from an embedded device platform may be simplified by understanding what the benefits and limitations that such platforms entail.

Benefits

Low power requirements (could run off a UPS for days)
Very fast boot times (Alix boots FreeBSD in 43 seconds)
Good network I/O
Durable for unfriendly environments
Wireless and network connectivity

Limitations

Memory (256 megs or less usually)
Storage (some CF cards are up to 8/16 gig)
Flash i/o performance and limited write cycles
Expansion limited to Mini-PCI slots

So given the limitations, what are applications that we can put on Embedded FreeBSD? Several are obvious, like a router or firewall deployments, (see pFsense) but there are a few others you might not think of right off:

- Fast booting and quickly available DNS/DHCP servers (very good)
- Asterisk PBX (perfect for SOHO with sip-providers instead of PSTN lines)
- Radius Servers
- Web server for small sites with mostly static content (kiosks, portals, web-app front-ends)
- Network monitoring systems
- Wireless Access Points or wireless bridges
- IRC or shoutcast streaming servers

These applications work surprising well, as they need minimal cpu, a lot of network i/o and minimal disk activity.

On the 'Net

- Titan Wireless, LLC, *http://www.titanwirelessonline.com*
- FreeBSD *http://www.freebsd.org*
- FreeBSD Online *http://www.freebsdonline.com*
- Alix *http://pcengines.ch*

FreeBSD allows you to mount certain filesystems read-only, which if you plan carefully, you can build a server with a filesystem that is nearly indestructible. Mounting / and `/usr` read-only and then making `/var` a ramdisk is a favorite trick of mine. One can always plug in external storage (the Alix has 2 x USB 2.0 ports) and place data on thumb or external hard drives.

Avoid Disk or CPU intensive Programs

Applications that have a lot of disk i/o will not bode well due to the bandwidth and cycle limitations of flash memory. Likewise, programs that consistantely load the processor will cause higher power consumption, and higher latency.

Summary

FreeBSD is well suited for x86-based embedded platforms. FreeBSD 8.2 really shines on these small machines as an application platform. FreeBSD is well known for it's reliability and performance on large footprint servers. That same reliability is enhanced on embedded platforms. Installation is simple straight-forward, and the hardware for systems like the Alix platform are well-supported by FreeBSD.

Replacing PC's or servers that draw hundreds of watt/hours of power, with small embedded devices could easily *green* up an office or data center, without sacrificing reliability or performance.

Embedded x86 based hardware offers a great opportunity for full-power operating systems such as FreeBSD to explore avenues previously limited to custom linux distributions. Replacing just one full size server with an embedded server will produce a return on investment in only months, from the power savings alone.

BILL HARRIS

has been installing and managing a variety of Unix Operating Systems for the last 25 years in the North Texas area. He has worked on everything from Radio Shack(c) Xenix, DEC Ultrix, Digital Unix, FreeBSD and Linux.


Looking for help, tip or advice?
Want to share your knowledge with others?
BSD MAGAZINE
Give us your opinion about the magazine's content
and help us create the most useful source for you!

Mono

(C# and the .NET Framework) on FreeBSD

The .NET Framework and the C# language have simplified the software development process in many ways.

What you will learn…

- How to install Mono
- What is and how to use portshaker
- The basics of the Mono components
- How to compile a simple application using Mono
- How to install and the basics of using the MonoDevelop IDE

What you should know…

- How to install and use the FreeBSD ports tree as it will be used to install Mono, Portshaker, and MonoDevelop
- Development basics

A lot of the complexities of other languages, such as memory management, are solved automatically using C# and the .NET Framework. Because of this, rapid application development is simplified. What's better, is that all of this is available today on FreeBSD using Mono.

What is Mono?

The home page of the Mono Project, *http://www.mono-project.com*, describes Mono as follows:

Mono is a software platform designed to allow developers to easily create cross platform applications. Sponsored by Novell (http://www.novell.com/), Mono is an open source implementation of Microsoft's .NET Framework based on the ECMA (*http://www.mono-project.com/ECMA*) standards for C# (*http://www.mono-project.com/CSharp_Compiler*) and the Common Language Runtime (*http://www.mono-project.com/Mono:Runtime*).

There is more information on the What is Mono page here: *http://mono-project.com/What_is_Mono*. This article will cover the following topics.

- Installing Mono
- Installing and using portshaker
- Mono components
- Compiling Hello World in Mono
- MonoDevelop IDE
- The BSD# Project

Installing Mono

Mono is available as a port on FreeBSD. The following steps will guide you through installing it.

Step 1. Download FreeBSD ports

For those new to FreeBSD, the ports tree is a list of applications that can be automatically downloaded, compiled, and installed. Running the following command as root will download and install the ports tree.

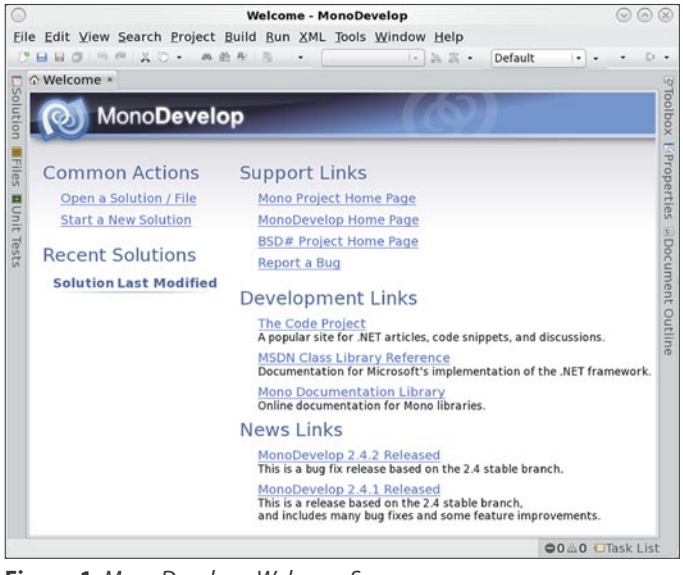


Figure 1. *MonoDevelop - Welcome Screen*

```
# portsnap fetch extract
```

Step 2. Installing and using portshaker

The default ports tree does not include all the ports that use Mono. Also some ports based on Mono that the default ports tree include are no longer maintained there. Instead, many Mono ports are only added to the ports tree using a tool called portshaker. The long-term supported version of Mono is 2.6.7 is in the regular ports tree. However, the latest version is 2.10.1. The portshaker utility will merge into ports latest version of Mono. If you prefer to use the long-term supported version, skip this step. Installing portshaker.

```
# cd /usr/ports/ports-mgmt/portshaker
```

Once portshaker is installed, running it as root will update the mono port and merge Mono ports into the ports tree.

```
# portshaker
```

The mono port is updated and other mono ports are merged into the ports tree.

Step 3. Installing the latest version of Mono from Ports

Once the ports tree is installed, Mono can be installed by running the following as root.

```
# cd /usr/ports/lang/mono
# make install
```

This will download and install mono.

Mono Components

Mono is composed of the following components:

- Mono compiler
- Mono runtime
- Base class library
- Other libraries

Mono compiler

Like many program languages, Mono code is compiled. Mono, like .NET Framework on Windows, has different versions. Each version has a separate compiler.

`mcs`	The depricated compiler for .NET 1.1
`gmcs`	The compiler for .NET 2.0
`dmcs`	The compiler for .NET 4.0

Use the appropriate compiler for your needs. Any new application should use the latest compiler but there may be business reasons for using a previous compiler such as .NET 2.0.

Mono runtime

The mono runtime could almost be compared to running a shell or perl script, as mono applications are launched by first calling mono:

```
# /usr/local/bin/mono /usr/local/path/to/someapp.exe
```

The mono binary is the runtime that implements the CLI (*Common Language Infrastructure*). It includes a JIT (*Just-in-time*) compiler, an AOT (*Ahead-of-Time compiler*). It also handles memory management for you and has an excellent garbage collector to clean up memory.

Base class library

Mono is first and foremost a framework and as all frameworks it has a very large class library. Most importantly, this class library is compatible with the Microsoft .NET class library, so applications written in Mono should also run on Windows platform. This allows enterprise solutions to be developed once and to use them on any of their operating systems.

Compiling "Hello World" in Mono

As mentioned previously, one of the compilers for Mono is gmcs. It is simple to compile C# code from the command line. Create a new file called *hw.cs*.

Note

Class object files in C# end with a .cs extension. Unlike C++, which has a .cpp file and a separate .h file, classes are contained in one file. Using your favorite text editor, add this text to the new file:

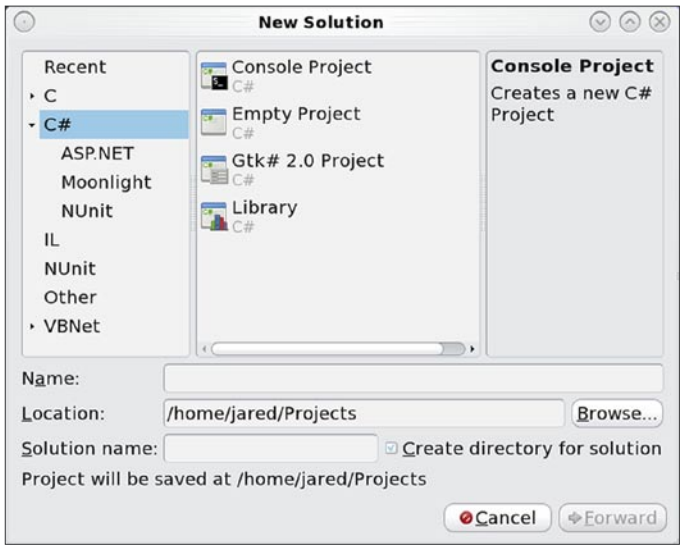


Figure 2. *MonoDevelop - New Solution*

```
/*
 * hw.cs
 */
using System;
namespace HelloWorld
{
     class HelloWorld
    {
        static void Main(string[] args)
        {
            System.Console.WriteLine(„Hello World”);
        }
    }
}
```

Save the file. Compile the code to create an hw.exe program.

```
# gmcs hw.cs
```

Now use the mono runtime to run the file. You have now compiled your first .NET application using Mono.

MonoDevelop IDE

There is an IDE for Mono named MonoDevelop. MonoDevelop is described as follows on its home page: *http://www.monodevelop.com*. MonoDevelop is an IDE primarily designed for C# and other .NET languages. MonoDevelop enables developers to quickly write desktop and ASP.NET Web applications on Linux, Windows and Mac OSX. MonoDevelop makes it easy for developers to port .NET applications created with Visual Studio to Linux and to maintain a single code base for all platforms. MonoDevelop has a lot of the features of modern IDEs, such as code completion, auto-format and more. The current release in ports is 2.4.2

Installing MonoDevelop

MonoDevelop can be installed from ports as follows.

```
# cd /usr/ports/devel/monodevelop
# make install
```

Note

While Mono itself is not yet available for PC-BSD as a PBI, MonoDevelop 2.4.2 is available as a PBI which makes it available to install from the PC-BSD Software Manager. *http://www.pbidir.com/bt/pbi/415/monodevelop*

Note

MonoDevelop 2.6 is in beta, though there is not yet a port for it. Some of the exciting new features include support for .NET 4.0 projects and improved GIT support. To get a more complete list of features being added in 2.6, see the following link. *http://monodevelop.com/index.php?title=Download/What's_new_in_MonoDevelop_2.6.*

Running MonoDevelop

A MonoDevelop shortcut is installed under the Development section of the KDE menu. However, you can simply run monodevelop from a shell. If you are coming to mono from Visual Studio, you are going to feel right at home as MonoDevelop has a feel very similar to Visual Studio. The Welcome page is quite friendly and provides links to help get you started.

Creating a new Solution

In .NET there are both Projects and Solutions. First you create a Solution. A single default project is added to the Solution by default and more Projects can be added to a Solution. Creating a Solution is simple. There is a link on the Welcome page or you can go to File | New | Solution or for those who prefer keyboard shortcuts, you can press *Ctrl + Shift + N*. There are multiple solution types you can choose from. You can develop in C, C#, VBScript, and other languages.

The BSD# Project

There is a project that exists to port Mono, MonoDevelop, and Mono-based applications. The web site home page, *http://code.google.com/p/bsd-sharp/*, describes the project as follows.

The BSD# Project is devoted to porting and maintaining the Mono .NET framework and applications for FreeBSD.

The repository currently contains FreeBSD ports for the framework, libraries and third parties applications released which are not yet in the main FreeBSD ports tree, with the intent that they will be integrated once they are ready.

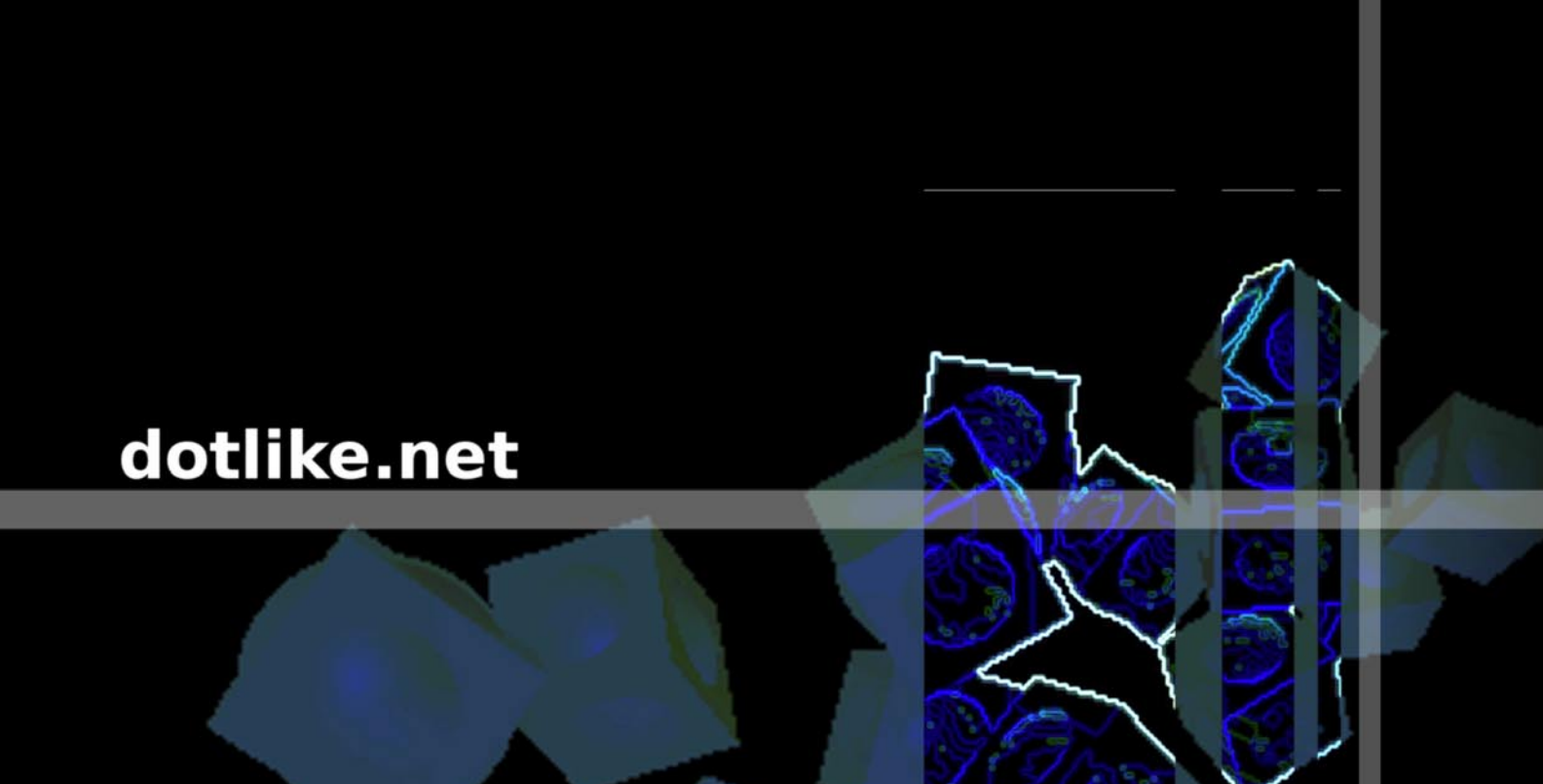
The project aims to act as a central testing point for porting new releases, for introducing new applications, and for testing framework wide changes that will affect all applications that rely on Mono, before they reach the FreeBSD ports tree.

Some information can also be found at *http://www.mono-project.com/Mono:FreeBSD*. While the project is successful, more contributors are needed as there is plenty of work to do. If you have time and the desire, consider joining the project. You can start by becoming a member of the *mono@freebsd.org* mailing list.

JARED BARNECK

Jared Barneck has been a FreeBSD enthusiast for over ten years. He works as a C# developer for LANDesk Software. He maintains a blog at www.rhyous.com where he shares his FreeBSD and C# knowledge with all of us.

dotlike.net



Linux
Netzwerk
Sicherheit
Programmierung

Drupal on FreeBSD

Part 6

In this the last article in the series on the Drupal Content Management System, the author looks back at what has been covered in the previous 5 articles and shares his real world experience with Drupal.

What you will learn…

- How to integrate PHP / JS and write a basic drupal module

What you should know…

- Basic BSD / PHP skills and how to install / administer Drupal CMS (Parts 1, 2, 3, 4 & 5)

One of the reasons the author is so encouraged by software licensed under the *General Public License* (GPL) and the BSD Licence is there are no limits to how far applications can be extended or modified to meet the business requirements of the moment. With the current belt-tightening amongst organisations, a powerful case can be made for non-proprietary software especially where the IT department is the focus for innovation and cost savings. This especially extends to the area of the World Wide Web, where commercial and government organisations are being forced to adopt a professional web presence but at the same time struggle with software that just doesn't quite "fit". Proprietary software often does not achieve 100% of the customer requirements, additional modules or modification are often required for specialist needs and this once again raises a historical problem IT managers and developers struggle with. Do you purchase an off-the-shelf product and live with the functional deficiencies, write the application from scratch or use a BSD/Open Source solution? Each approach has its strengths and weaknesses, but in the authors experience, the latter solution is becoming accepted even in organisations that were strongly committed to Closed Source as recently as a few years ago.

Getting down to fundamentals, software such as FreeBSD and Drupal allow the System Architect to be creative and flexible and meet more of the customer requirements either in a shorter period of time or cheaper than proprietary solutions. Recently the author was asked to publish a number of YouTube videos on his employers website, but due to the structure of the CMS application a vendor supplied module was required, at the cost of approximately $1000. The author could have written a module in-house to add this functionality, but time was committed to other projects. In the end, the idea was abandoned and much was made of the irony that an application that was meant to help the organisation communicate effectively via the Internet was in fact hindering the process. I don't believe for a moment that the original developers of the application were so short-sighted as to ignore the importance of extensibility, but it is disappointing when the financial imperative to make a profit limits creativity. Without straying too far into the quagmire of licensing ethics, it is safe to say Closed Source software (especially vertical market applications) will allow the developer to be as creative as the budget or the API allows. GPL and Open Source software by being intrinsically open, does not suffer from this. The opposite indeed could be argued – Drupal warns against *Hacking Core* and the temptation has to be resisted to perform a quick and dirty fix that has long term implications.

One of the old sayings is that WWW doesn't stand for World Wide Web it stands for Wild Wild West. Indeed, the

Internet is a challenging and often hostile environment, yet the similarity between the two is striking. Like the Wild West, frontiers are continually being moved forward, competition is rife, and it is essential that if an organisation is to achieve and maintain a lead that innovation and creativity are built into the process. The more constraints that are placed upon the developer or architect, the less successful the project will be in the medium and long term as there will be conflict between the rapidly changing needs of the Internet community (The Customer) and any limited functionality of the software. This also the raises the issue about the *The medium is the message* – again there is tension between those that wish to maintain the status quo of the old way and those that wish to expand horizons – a good example being Mark Zuckerberg (Facebook) and Jack Dorsey (Twitter).

By design, FreeBSD and Drupal inherently lean towards creative solutions, and in the authors experience they have delivered every time.

Case Study

An urgent request for an online calendar based booking system with reporting was given to me to develop, to replace the current telephone/paper based system. There was no available budget. The pre-requisites were user friendliness, security, speed of development, robustness, workflow and mission criticality. All individual daily bookings were to be displayed on a large wall mounted LCD monitor for customers and the department to view on a daily basis. The system would be accessed by customers 24/7, and any failure in the system would have a major impact on the organisations reputation. The most important factor was that the interface had to be foolproof, and it was essential that under no circumstances could double-bookings take place.

Three options were considered, use the current CMS product and write a custom module and mini-site, write a custom application from scratch using PHP or Perl, or use Drupal. The first option was dismissed as the current CMS is under review and is due to be upgraded or replaced in the near future. While I was confident I could write a suitable application in Perl or PHP, I was concerned that I would not have sufficient time to test the system properly for any security flaws or bugs as it was essential that no customer could have access to any other customer's history. Drupal was finally settled on, as I could quickly put together a prototype, gather together a willing base of customers for user acceptance testing as the system was being developed, and also have the ability to add further functionality at a later date (e.g. XML integration with our Financial Management Systems). The system was developed and rolled out over a period of weeks which included commissioning the server, configuring email and backups, testing, firewall configuration, consulting with users/training and adding extra functionality. If my time was committed 100% to the project this would have taken considerably less, but I had to fit this in around other commitments. The system was developed using Drupal 6.2 as some of the modules were not available for 7.0.

Key functionality

The system had to support:

- Calendar based unique bookings
- Reports for customers and the administrator for historical and daily bookings
- A unique slot system based on 45 minute periods (but not necessarily consecutive)
- Bookings could only be made for Monday to Friday
- Online registration and password changes
- Email alerts to the office administrator to approve accounts and bookings
- Email alerts to customers when booking is approved
- Display daily bookings on public monitor
- A *Chinese Wall* that allows customers to quickly identify unavailable slots but not view their competitors bookings
- Automatic backups onto a Windows 2003 server

Apart from the standard core Drupal modules, the following were installed to add functionality (Table 1).

Method

The server was commissioned with a fixed IP address and the AMP stack was installed and configured. Server patched and upgraded to the latest versions.

Email was configured using Postfix, and as the server was being initially tested on our Intranet rather than being publicly available via our DMZ, some clever configuration was required of our Astaro firewall to allow outgoing mail out to the real world while sending email internally via Microsoft Exchange. I used the transport rules in Postfix to accomplish this, but this was the first major challenge as our System Administrator had our internal mail-server very secured and unfortunately we couldn't pass both the external and internal email via the Exchange box. Webmin was also installed to allow easy maintenance as well as SSH, but these were secured and configured so that access was only available from inside our network. As this was a time/date based application, it was important that the server was configured to pull the correct time and date from our in-house NTP server.

It was decided to use Drupal 6.2 as a number of the modules I wanted to use were not available under 7.0.

Hopefully in time this will improve, but for the immediate future 6.2 will continue to be supported so this was not an issue. 6.2 was duly installed, and the additional modules listed in Table 1 were uploaded and configured. Apache had to be configured to support smart URLS, and as I installed Drupal in a subdirectory off the webserver root, this new directory was made the root directory when pages were requested.

The next step was to automate backups, and a bash script was written to archive the MySQL database, the `/www` tree and important files from the system tree (`/etc`, `/logs` custom scripts etc.) and this was added as a cron job to be run daily. This allowed ad-hoc snapshots to be taken during development as required. An additional script was added to MD5 sum the archives, and copy these via CIFS to a Windows 2003 server share. The Drupal cron script was added to cron to be fetched via Wget and executed every 15 minutes, which would check for Drupal updates etc.

Development

Development of the application was straightforward. Additional content types were created for each type of booking, and pages were used to communicate to the users website status, FAQ section, terms of use etc. As each booking type had different fields, extensive use of CCK and FormFilter was used to add or remove fields as required on each booking page. While it is possible to use CSS and the template API to obfuscate fields, these could still be accessed so it was decided to remove any potential risk at source and not publish fields that potentially would pose a security risk. A good example is when a user adds a booking, Drupal provides a check box for Published. While it is easy to define in the content type for this check-box to be disabled, if it was still displayed to the user they could inadvertently enable it thereby allowing their booking to be crawled by robots or viewed by another user if they were to discover the URL. By using FormFilter and disabling Publish altogether, the only options available to the user were Save or Edit the booking. Considerable care was taken to ensure fields were validated appropriately, and most of this was easily achievable via CCK. The only custom validation was the bookings validation module, which will be covered later. A custom dropdown was used for the slot times as these were not consecutive and were every 45 minutes. The Drupal date module only supports 15 minute increments.

Table 1. *Additional modules for booking system*

Module	Provides
Administration Menu	Friendly Admin UI
CCK	Allow development of custom content fields
Calendar	Displays views in Calendar
Date	Add date field to CCK
Formfilter	Hides fields from customer (e.g. publish)
Sections	Themable sections
Skinr	Skins / themes Drupal output
Token	Provides hooks to node name in distributed email
Rules	Provides email based workflow
Captcha	Stops robots from subscribing
External Links	Adds icon to links external for site
Jquery	Required for CCK date popup
Views	Reporting and calender etc.
Custom Module	Slot verification

After the different content types were added, the next step was to integrate data gathered from the forms with the Calender and Views. Two calenders were designed, one for the Administrator that allowed them to view all bookings by all users with all details available, and a cut down version for the end users that only showed wether a slot was booked or available. Multiple reports were designed using views, and again the same method was used so that the Admin could have access to all the relevant information, and the end users only could access their particular history.

Workflow was the next challenge, and this was accomplished by using the Rules and Tokens module and every time a booking was added or approved, an email was fired off to all involved with the relevant status and booking reference.

The next stage was to build any ad-hoc pages, FAQ's, etc. as a precursor to building the menus and blocks. Once these we all in place, a block was created for the Administrator and a separate block for the users, each containing links to the relevant content, forms, reports etc.

Security and user accounts were configured next, and additional fields were added to the login profile for each user. A checkbox was added that would not accept the registration unless the user had read the terms and conditions. Groups were added for Administrators and Users, and access to the relevant content, modules and functionality were defined and checked. Blocks and menus were then configured as appropriate, allowing different levels of security.

A dedicated module was written to perform validation of the bookings. Originally, the unique field module was used but as the project progressed more and more validation

was required as additional functionality was demanded (e.g. the booking system was extended to cover an additional 6 locations rather than just the one in the original project specification). This was achieved using Netbeans and Xdebug, as some of the logic was quite complex. The Devel module was also temporarily installed, to allow evaluation of variables exposed via the API's.

A special view was created to display the daily booking as a report. Using the combination of Views slideshow, each booking was cross faded every 10 seconds or so. A custom template was added with a high-quality royalty free image as a backdrop which would display alongside the bookings via a unique URL. As a separate PC would be used in the reception area to display this, a dedicated Drupal account was created with a special login that would automatically display that page on login. Google Chrome was used as the browser in full screen mode and with a black background the display looks striking on a 22 inch LCD wall mounted flat panel.

Finally the Zen theme was uploaded and the CSS configured and written. Extensive use of Firefox and Firebug was used at this stage, and the *.tpl* and *.css* files were modified to give the site a professional look and feel while maintaining accessibility and cross browser standards.

From a system administrator angle, backups were tested and a complete restore was checked using a Virtual Machine. Provided a donor server is available with the the OS preconfigured, with the current dataset I can have the system restored in under 5 minutes. The system was then tested using various tools including Zenmap, Siege and Wget to check for unwanted open ports, stability, and data being exposed that shouldn't be.

Testing and project creep

Initially the system was tested in house by 4 members of staff and this ironed out most of the initial bugs and logic problems. Remote access via our secure portal was given to a few external customers to test, and while the initial version followed the original specification many good suggestions were submitted by those who would use the system on a daily basis. While this added to the burden of project creep, Drupal proved to be powerful and versatile and all the customers suggestions were incorporated by the go-live deadline.

Go Live

The server was relocated to the datacentre and reconfigured to reflect the move into our DMZ. Various Postfix and network settings were reconfigured to reflect the move, and the external Astaro firewall configured to allow access. Web services were then available publicly, and admin functionality such as SSH and Webmin were only allowed internally. Site was launched on time, at the total cost of a second-hand HP server and my development time.

Lessons learned

As usual, it is important to get a decent functional specification at the outset. Unfortunately, with the best will in the world project creep is always a reality. My biggest mistake was not anticipating the additional site data which required moving from a Drupal module to a custom module written in-house. In hindsight, it would have been preferable to have used that method at the beginning but as the original plan was not as wide reaching, this is understandable.

If you are going to write a custom module, unless it is only a few lines of code, a debugger is essential. Xdebug and Netbeans is an excellent combination for this, but should only be used in a test environment as it exposes critical information (e.g. login names).

We did experience one issue with the calendar and earlier versions in Internet Explorer. Access via our secure portal would not allow the user to populate the date field calender dropdown, but switching to Google Chrome or Firefox solved this problem. This was a strange problem, as using the same version of IE on our local network was OK.

Final Outcome

Since go live, the system has required only minor tweaks – one new customer thought the email alert was too brief so further details was added. Some changes have been made to the reports to give different views of the data. One pleasant surprise we discovered as we moved from a paper based booking system to online is that a particular 80 year old customer had to buy a PC to go online as they hadn't used the internet before. They are now successfully using the system on a regular basis.

ROB SOMERVILLE

Rob Somerville has been passionately involved with technology both as an amateur and professional since childhood. A passionate convert to *BSD, he stubbornly refuses to shave off his beard under any circumstances. Fortunately, his wife understands him (she was working as a System/36 operator when they first met). The technological passions of their daughter and numerous pets are still to be revealed.

Backups – Made Easy

A fast solution to a real problem

When you have to do a major Operating System or Application upgrade, this script and server with big disks, will get the job done.

What you will learn…

- How to do server to server backups, easily and simple across a LAN
- How ftp and dump, combined, can form a powerful backup solution

What you should know…

- How to edit simple shell scripts
- Create/modify user accounts with adduser
- How to enable ftp services (ftpd)
- Edit system files with vi or other text editors

Backups are usually the lowest priority of a new server install, but invariably, soon become an issue when a application or system patch needs to be applied.

Where is the tape drive? Do I have enough optical media?

Sometimes, just a quick snapshot is that it needs, as a fall back in case an upgrade goes awry.

Other times, you want a daily or weekly off-disk dump of a critical server. If you have another Unix/Linux server on the network with plenty of storage, the simple shell script below will do the job nicely.

The beauty of this script is needs no additional packages, as it uses only commands included with a minimal install of FreeBSD, and few services other than ftp and lots of storage on the target server.

The script, obviously, should not be used on an insecure or network with limited bandwidth. What you will need:

- Destination Server on the same LAN subnet providing ftp services, and plenty of storage. System can run any modern Unix/Linux OS.
- Good LAN connectivity between the source and destination servers (preferably 100Mb/s or better and on the same physical subnet)
- root shell access to the source server

This process will refer to *source* and target server. The source server is your FreeBSD machine you wish to backup.

The *target* or destination server is the machine on the LAN with plenty of free disk space.

Target Server

Create an account on called *backup* with a known password and a home directory on a filesystem with plenty of free space. Some systems may already have a backup account that simply needs a password set to be usable (eg: Ubuntu).

Check the system to make sure it has a functioning ftp server. The steps to configure the ftp service varies between operating systems.

Make a directory called *dumpfiles* in the backup user's home directory and make sure it's owned by the backup account, eg:

```
# mkdir /var/backups/dumpfiles
# chown backup /var/backup/dumpfiles
```

It bears noting once again, make sure that the filesystem where *backup*'s home directory is located, has enough free space to hold your planned dumps of your target machine. FreeBSD's adduser will let you specify the home directory when you create the account.

Listing 1. *run_backup.sh // Bourne Shell Script*

```
#!/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/sbin:/usr/local/bin:/root/bin
export PATH
#     Simple and insecure script to do a fast dump of
#     specified filesystems to a remote server with storage
#
echo "BackMeUp"
echo "version .5"
umask 066
HOST='/bin/hostname'
BACKUPHOST="targetserver.acme.org"
BACKUPUSR=backup
BACKUPPWD=backmeup
BACKUPLOG="/tmp/backup.log"
DUMPDIR="/var/backups/dumpfiles"
DUMPFS="/ /usr"
DUMPDATE='date'
X=1

cat >> .netrc <<-EOF
machine $BACKUPHOST
       login $BACKUPUSR
       password $BACKUPPWD
       macdef init
       !rm .netrc
       pass off
       xferbuf 4000
       cd $DUMPDIR
       mkdir $HOST
       cd $HOST
       binary
EOF
for fs in $DUMPFS
do
       echo "put \"|dump 0aLf - $fs \" $HOST.$X.dump" >>.netrc
       X='expr $X + $X'
done
echo "quit" >>.netrc
echo "" >>.netrc

echo "$DUMPDATE: Backup started " >>$BACKUPLOG
ftp $BACKUPHOST
echo "$DUMPDATE: Backup Completed " >>$BACKUPLOG
```

Source Server

You should first test connectivity between your source and target server, by using ftp from the command line and your target server's backup account credentials. Test transferring a file to make sure permissions are correct on the destination account, eg:

```
# cd /usr/share/misc
# ftp target_server
connected to target_server
220 target_server FTP server
Name: backup
Password: backmeup          // or whatever you chose
ftp> cd dumpfiles
ftp> put birthtoken         // ( a known text file on BSD
```

If the login and file transferred successfully, then your are ready to test the script below. The shell script is meant to be run under *root* privileges, which is a requirement of *dump*.

The script utilizes ftp's .netrc command file capabilities, which do create some potential security concerns, and should be used only on a secure network. The script attempts to minimize exposure of usernames and passwords as much as possible, but you should understand these concerns.

- It sets the umask for the user to read/write for the user only (no other access).
- The script generates a *.netrc* file in the user (root) home directory with the necessary commands to execute dump for each filesystem specified, over a ftp pipe.
- ftp is called with the target host, which reads in the just created *.netrc* file.
- the .netrc is deleted from the local disk.
- the target host's *.netrc* commands are read/processed along with the *init* macro.
- A directory is created on the remote host based on the source's hostname.
- the actual *dump* commands are processed for each of the filesystems listed in DUMPFS variable.

The script should be edited to properly reflect your values in the variables below:

BACKUPHOST is the target server.
BACKUPUSR is the account we created or are using on the target server.
BACKUPPWD is the password for the account above.

On the 'Net

- FreeBSD *http://www.freebsd.org*

DUMPDIR is the location of the dumpfiles directory in the backup users home directory.
DUMPFS is the list of filesystems on our source server we want to backup/dump.

Save the following as `run_backup.sh` (see Listing 1).

Some notes:

- Script is tested against FreeBSD 8 on source, FreeBSD and Ubuntu on target.
- Change permissions on the `run_backup.sh` by using `chmod 500 ./run_backup.sh` to limit read/execute to root only.
- Move run_backup.sh to `/root/bin` and limit access to `~root/bin`.
- The value used in the *xferbuf* command can be increased substantially if your servers are on a gigabit LAN. The value supplied works well for 100BaseT.
- Other options for dump can tune the process for performance, like block size.
- Each filesystem will be stored in a dump file with a number representing it's position in the DUMPFS variable .. `ie; / is 1, /usr is 2`, under a directory named by as the hostname of your source server.
- The dump is a level 0, which includes all files on the given filesystem.
- it is possible to do later restores over an ftp pipe, using: `ftp> get server_name.0.dump "|restore -ivf -"`
- The script could be placed in root's cron to be run regularly, and redirect stdout.
- The script could easily be enhanced to support incremental support as well.
- Any unix/linux or even a Mac OSX laptop can act as a remote target for your backup script.

BILL HARRIS

has been installing and managing a variety of Unix Operating Systems for the last 25 years in the North Texas area. He has worked on everything from Radio Shack(c) Xenix, DEC Ultrix, Digital Unix, FreeBSD and Linux.

Say Hello to Red Team Testing!

Security Art

Security Art's Red Team service operates on all fronts on behalf of the organization, evaluating all information security layers for possible vulnerabilities.

Only Red Team testing provides you with live feedback on the true level of your organizational security.

Thinking creatively! That's our approach to your test.

Security Art's Red-Team methodology consists of:

1. Information and intelligence gathering
2. Threat modeling
3. Vulnerability assessment
4. Exploitation
5. Risk analysis and quantification of threats to monetary values
6. Reporting

Ready to see actual benefits from your next security review?

info@security-art.com

Or call US Toll free:
1 800 300 3909
UK Toll free:
0 808 101 2722

www.security-art.com

Fighting DDoS Attacks with PF

For a long time, Denial of Service attacks were disregarded, as they were considered to be the work of script kiddies.

What you will learn…

- how to make advanced PF configurations against specifics threats
- how to use third party diagnostic tools with PF

What you should know…

- how UDP/TCP connections work
- how UDP/TCP connections are working

Things have changed, these attacks are now massively distributed in order to be more efficient and have serious goals. Anonymous and related groups use these attacks to share political messages, mafia from around the world use these attacks to blackmail shopping websites. So, network administrators need to be prepared to react as efficiently as possible to properly mitigate these attacks. In this article, we will explore some simple, but effective strategies you can use to mitigate these attacks using *Packet Filter* (PF). Attacks that saturate the incoming bandwidth are out of the scope of this article, since these attacks cannot be stopped by PF and need to be fought at the ISP level. Instead, we will focus on attacks that saturate other network resources.

Getting the Diagnostic

In order to efficiently fight a threat, one should have the most accurate information. The following commands will help us gathering this information.

A good start would be to look at the different counters:

```
pfctl -s info
```

Some values, like the total number of states in the state table have a hard limit. PF does call this a hard limit but nothing is hard-coded in the source code, the default limits may be modified in pf.conf as we will see during the article.

The following command will show us these limits:

```
pfctl -s memory
```

Since one of the most important tables is the state table, we might want to take a look at it with the following `verbose` command, which prints the full content of the table:

```
pfctl -s state
```

We can filter the IP addresses establishing incoming connections by the number of established connections:

```
pfctl -s state | cut -d' ' -f 3 | cut -d: -f 1 | sort
                | uniq -c | sort -n
```

Information at the PF level is not sufficient, during the attack, you should also gather raw packets with tcpdump. In order to quickly analyse these captures, Tshark (available without installing the wireshark port) will help us. For instance, to display only TCP Syn packets (high rates can be observed during a SYN Flood attack), we will use the following command:

```
tshark -R „tcp.flags.syn==1 && tcp.flags.ack==0" -r/
                path/to/capture.cap
```

The following command will prove helpful when trying to dissect HTTP packets:

```
tshark -z „proto,colinfo,http.content_length,
http.content_length” -z „proto,colinfo,http.content_type,
http.content_type” -R „http.response and http.content_type
contains image” -r /path/to/capture.cap
```

By adapting the above commands to better fit the particularities of each situation, it is possible to identify the type of DDoS attack that PF is facing.

Spoofing 'n' Flooding

The good news is that the attacks involving source address spoofing must remain very simple, since the attacker is not receiving response, he is not able to establish a complete session. The bad news is that most of the time, these attacks create an half-open connection, which consumes resources while waiting for the establishment of the complete connection. This, never happens in these attacks. The worst news is that the packets sent during these attacks are indistinguishable from legitimate packets, so they can't simply be blocked.

In order to fight these threats, you first need to block as many UDP protocols as possible. For instance, SNMP v3 is still UDP based but, should be accessed from only a few specific IP addresses. Since there is no handshake mechanism in UDP, each spoofed packet seems to be a legitimate one.

In the TCP world, every connection must have performed the three-way handshake in order to be usable. So the only possible attack (without revealing the offending IP address) is SYN Flood. Adding synproxy state to all pass rules should solve the problem in most cases, but it does not work in bridged mode which may be problematic for many. Below is an example of a rule rule involving synproxy:

```
pass in quick on $ext_if proto tcp to ($ext_if) port
                80 flags S/SA keep state
```

Since each SYN request is generating an half-open connection, the state table reaches capacity very quickly and PF starts to block incoming requests. The default limits are very conservative and *BSD with 1GB+ RAM available can perform very well with millions of entry in the state table. Modifying these limits can be done simply by adding a limit directive in the PF configuration:

```
set limit { states xxxxxx, src-nodes xxxxxx, frags
                xxxxxx, table-entries xxxxxx}
```

The states value defines the maximum number of simultaneous states, this is definitively the first value to raise for fighting an attack. Default value is 10000, but, 100000 is a good start. Using only this setting would show you some other limits like src-nodes value if you use sticky-address or source-track options. The most important parameter is table-entries, this is the global limit for all the tables used by PF. This parameter should always be more than twice the states parameter value, especially if you use NAT in your rules. The frags parameter is the maximum number of packets buffered for scrubbing. Be careful with this value. If you are facing attacks involving extremely high numbers of large UDP packets, it may be more efficient to disable scrubbing.

Increasing the limits is a good approach but is not sufficient for high-rate attacks. At this point you will have to act more aggressively on removing remove old entries from the state table. We will need to ask PF to purge more often (every 2 seconds instead of the default 10 seconds) with the following rules:

```
set optimization aggressive
set timeout interval 2
```

These optimizations may also be used against UDP based attacks since PF also uses some mimics states in order to have a record of sessions during UDP transactions. In some cases, these counter-measures will not be sufficient and the only remaining solution is to force PF to become stateless, using the no state option.

```
pass in quick on $ext_if proto tcp to ($ext_if) port
                80 no state
```

Not spoofing but still flooding

Attack on services without spoofing is generally used against TCP based protocols and let us know the real IP addresses of the attackers. So we can use PF to individually blacklist each IP involved in the attack. Since PF is not able to inspect pieces of data contained in packets, the only way to detect these attacks is by identifying IP addresses that make a lot of requests during a small amount of time. We will be using max-src-conn-rate and max-src-conn options to add the offending IP addresses to a table referenced by the overload keywords. Last but not least, we will be able to flush every connection already established by the offending IP addresses with the flush option. So if we want to limit to 75 total connections per host and only 10 new connections per 5 seconds period, we will put these lines on our ruleset:

```
table <blacklisted_hosts> persist
block in quick from <blacklisted_hosts>
pass in quick on $ext_if proto tcp to $web_server port
80 flags S/SA keep state (max-src-conn 75, max-src-conn-rate
10/5, overload <blacklisted_hosts> flush)
```

This approach is very efficient but may not help if each host is performing a reasonable level of connections. In this case, we will have to isolate the offending IP addresses by inspecting pieces of data contained in packets.

Depending of on the refinement of the attack, the attack could be operating at OSI level 4 (e.g. sending non-HTTP traffic to port 80) or level 7 (e.g. sending valid HTTP requests). In each case, the most important is to detect a pattern in the packet content. After having identified a pattern, we will have to inspect each incoming packet in order to identify offending IP addresses.

This task could be done by snort but this is a little bit over-sized for what we need to do. In our case ngrep should be enough. ngrep is easy to install (no configuration needed) and will inspect packets looking for a particular regular expression. Like tcpdump, ngrep can also use BPF filters. Like grep, ngrep can be inverted in order to show only non-matching packets with the -v option.

For instance, if we want to log non-HTTP traffic incoming on TCP port 80, we will use the following command-line:

```
ngrep -q -d em0 -p  -v '^GET .* HTTP/1.[01]' port 80
                >> offending_packets.log
```

Having the offending packets heading in a log file is a good start but this is still not enough for PF. If you are really sure of your pattern, you could use the following set of commands in order to add the identified IP addresses to the blacklist:

```
grep „^T .*:.* -> .*:80" offending_packets.log | sed -e
„s/^T //" | sed -e „s/:.*$//" | sort | uniq | xargs pfctl
                -t blacklisted_hosts -T add
```

This is a basic approach that works but may really be risky and does not help us to unban IP addresses. In order to have more control on the log file analysis We will use fail2ban.

The fail2ban configuration is really straightforward, we will see it in action by following our previous example. Since the default configuration of fail2ban does not support PF, we need to add a least these configuration parameters to `/usr/local/etc/fail2ban/action.d/pf`:

```
[Definition]

actionstart =
actionstop =
actioncheck =
actionban = pfctl -t blacklisted_host -T add  <ip>
actionunban = pfctl -t blacklisted_host -T delete `pfctl
-t blacklisted_host -T show 2>/dev/null | grep <ip>`

[Init]

port = http
localhost = 127.0.0.1
```

We will also have to configure a filter for our custom log file in `/usr/local/etc/fail2ban/filter.d/ddos`:

```
failregex = T <HOST>:.* -> .*:.*
```

And finally configure the use of these components on `/usr/local/etc/fail2ban/jail.conf`. We will ban IP addresses that appear in the log file 20 times or more:

```
[ddos-pf]

enabled  = true
filter   = ddos
action   = pf-allports[name=http, protocol=tcp]
logpath  = /path/to/offending_packets.log
maxretry = 20
bantime = 172800
```

Fail2ban will do the work for us but it is always a good idea to look over its shoulder at the blacklist:

```
pfctl -t blacklisted_host -T show
```

Conclusion

We have seen that a simple *BSD box with PF and few other tools are pretty efficient when fighting DDoS attacks. As always, the key points are to use the right tool at the right moment and to know the capabilities and limitations of each tool that we may have to use.

MATTHIEU BOUTHORS

Matthieu Bouthors is a French *NIX enthusiast since a decade. Working for a French hosting company, he aims find open source solutions meeting the high-level requirements of its customers.

Data CENTER

FOR IT PROFESSIONALS

MAGAZINE

Want to have all the issues of Data Center magazine?
Need to keep up with the latest IT news?
Think you've got what it takes to cooperate with our team?

Check out our website and subscribe to Data Center magazine's newsletter!

Visit: *http://datacentermag.com/newsletter/*

The MacOS X Command Line

My wife thinks I bought my Mac laptop to use as a status symbol. But every hacker knows I bought it because I wanted a decent Unix laptop.

What you will learn…

- Apple-specific command line tools
- Opening most files
- Working with the clipboard
- Taking screen shots

What you should know…

- The Unix Command line
- How to get around MacOS X

The fact it was based on BSD was even better. MacOS X features a command line interface that is as authentic as any Unix interface because BSD runs at the core of MacOS X. But Apple has provided a number of command line tools to enhance the experience and this article outlines the author's favorites.

open

MacOS X provides a command line tool to open applications and files. MacOS X applications are actually collections of files residing within one directory with a name ending in .app. I usually use open at the command line to start most applications, leaving the Dock clear of applications not running:

```
howardjp@thermopylae:~$ open /Applications/Safari.app
```

is enough to start Safari and if the browser is already running, it will open a new window. The `open` command also works on individual files and will open the file in its associated application. For instance, running open on a PDF will open the file in Preview. And running `open` on a normal directory (as opposed to an application package) will open the directory in Finder. The `open` command provides a number of useful options. The option `-t` treats the file, regardless of type, as a text file and opens it in the default text editor. A related option, `-e` simplifies the process and opens the file in TextEdit, the native text editor provided with MacOS X. Also related is `-f`, which reads from the standard input and passes the input to the default text editor.

It is also possible to override the default application with other types of files using the option `-a`. But it is important to remember the full path to the application must be given:

```
open -a /Applications/Adobe\ Reader\ 9/Adobe\
Reader.app/foo.pdf
```

This form is quite cumbersome, but it may be appropriate in some circumstances. One last option worth mentioning is `-R` which find the references file in Finder, instead of opening the file itself. Finally, the `open` also supports URLs:

```
open http://www.jameshoward.us
```

will open my website directly in the default browser.

pbcopy and pbpaste

The Unix command line has historically interacted poorly with the numerous graphical interfaces that have been stacked upon it. One key area lacking support is the clipboard. MacOS X brings two utilities to close that gap, `pbcopy` and `pbpaste`. These commands together provide complete access to the MacOS X clipboard (which Apple calls the pasteboard, explaining the names of these two commands). The first of

the two, pbcopy, takes its input from the standard input and adds it to the system clipboard. The command only accepts one option, -pboard, which accepts one of four suboptions, *general'*, *ruler*, *find*, and *font*, all of which are different system clipboards available on MacOS X. The general pasteboard is the main system clipboard and the others are for special use. The pbpaste pulls data from the clipboard and prints it to the standard output. Like pbcopy, pbpaste accepts the option \opt{pboard} to determine which pasteboard to acquire data from. The pbpaste command adds a second option, -Prefer which takes three possible options *txt*, *rtf*, and *ps*. These options direct pbpaste to looks for a certain type of formatted information on the pasteboard. The *txt* flag suggests standard text data. The *rtf* and *ps* suggest Rich Text Format and PostScript, respectively. Despite this option, it is not possible to direct the exact output pbpaste prints. This option only tells pbpaste what type of information to return first. These two commands offer the MacOS X command line warrior a simple and fairly complete set of tools for working with and manipulating the Mac OS X pasteboards.

Screencapture

Another command line gem in MacOS X is a screen capturing program called screencapture. This command line application accepts a handful of options making the tool quite powerful. The program requires a single command line option, a file name to store the screen capture in. Without any other options, this will copy the full screen to the named file, which is stored in PNG format by default. The file format can be changed with the option -t which accepts *pdf*, *jpg*, and *tiff* as acceptable formats. The manual page suggests other formats are permissible. Experimentally, *gif* works and *ps* does not. The option -w instructs screencapture to only capture a single window and highlights the current window. Moving the mouse will allow the user to select a different window for capture. The -o option forces screencapture to ignore the shadow when capturing a single window. Like other screen capture utilities, screencapture allows the user to select a delay before taking the image with the T option, which accepts a number as the number of seconds to wait. The screencapture command provides other useful options. When the screen is captured with this utility, it triggers a sound like camera shutter opening and closing to signal the capture has been taken. This can be disabled, probably for nefarious purposes, using the -x option. Also when using the option -P, the utility will automatically open the saved image file in the Preview.app application. The screencapture command provides other options for controlling how a window can be selected and also for opening the screen capture in a new Mail.app message.

binhex and macbinary

If you have been a Macintosh user since before MacOS X, then you may have a collection of files stored in some of Apple's unique formats, such as BinHex or MacBinary. Apple has provided a command line tool for creating and converting these file formats. Prior to adopting the Unix-like structure of MacOS X, Apple used a proprietary disk format called HFS (an extended version called HFS Plus was also available). This disk format broke files into multiple components called forks. There were normally two forks with the first being traditional data. The second, called the *resource fork* included metadata applicable to the file, such as associated applications or icons. To simplify transfer of these files, the MacBinary format was created, that combined the forks of a file into a single package suitable for transport. They typically had a file name ending in .bin or *.macbin*. Apple provides macbinary for working with these types of files. The macbinary command takes a subcommand as its first option. Available subcommands are *encode*, which creates a new MacBinary file, *decode* which unpackages an existing MacBinary file, and *probe* which attempts to determine if the files listed are MacBinary files. Similar to MacBinary is that the BinHex format packages the different forks of an HFS-based file into one file, but also makes that file 7-bit clean for transferring over ASCII connections, such as email. This is similar to the use of uuencode on the Unix platform. These files typically had the extension *.hqx*. Apple also provides binhex to work with these files and it takes the same options as macbinary. Both commands take several options, but the most useful is -c which makes the two commands read from the standard input for *decode* and write to the standard output for *encode*.

Other Tools

The traditional Unix command uname is available for interested users, but Apple has provided a second command for MacOS X specific information. That command, sw_vers, will provide the product name (distinguishing between MacOS X and MacOS X Server), the operating system version, and the build number. In addition, there are a collection of utilities for accessing XCode, the native IDE for MacOS X, package building, and other developer tools. These were not including in this overview due to their technical nature, but they are useful to understand Apple has considered the needs of programmers when deviating from common practice in the Unix world.

JAMES P. HOWARD, II

The author is a senior analyst in Washington, DC, in the United States where he focuses on statistical and mathematical systems. He can be reached at jh@jameshoward.us or via Twitter @howardjp.

Implementing OpenSMTPD

An Independent Reference Document

OpenSMTPD is one of the mail servers included with OpenBSD. Configuring OpenSMTPD is more readily understood and comparatively less complex than configuring Sendmail.

What you will learn…

- How to prepare a gateway for network mail
- How to configure OpenSMTPD
- How to configure mail filtering

What you should know…

- Basics of OpenBSD
- General local-area network concepts

This document describes running an instance of the mail transfer agent OpenSMTPD which is included as a component of OpenBSD systems and can run as an alternative to the Sendmail internetworking Simple Mail service.

To implement this example, a working installation of OpenBSD is required.

Read the manual pages to make configurations specific to your network (see Listing 1).

This document does not describe certificate creation, the concept has been simplified to a case of sending e-mail relayed through a *centralized* server. The centralized server requires only a user and a password; e.g., a service provider which filters by network and the like.

The network topology used in this example is a network gateway that has two static addresses configured at a local ethernet interface and a local mail server with example.org as the domain. Eventually we will migrate to IPv6.

This example mail system configuration has alpine as the message user agent and procmail is the mail delivery agent. Bogofilter and spamd are implemented to prune e-mail. Procmail and bogofilter are presented with very simple configurations- the purpose of the basic configuration examples is that they run alright. If you want to, create recipes to enhance your specific network.

The gateway will be restarted following it's configuration- run `pfctl -nf /etc/pf.conf` before rebooting.

Gateway Configuration

Spamd is a component of OpenBSD, use the supplied scripts to make it run at startup.

Listing 1. *Pertinent OpenBSD Manual Pages*

```
$ man 1 bogofilter
$ man 1 procmail
$ man 5 hostname.if
$ man 5 mailer.conf
$ man 5 pf.conf
$ man 8 newaliases
$ man 8 newsyslog
$ man 1 pkg_add
$ man 8 rc.conf
$ man 8 smtpctl
$ man 8 smtpd
$ man 5 smtpd.conf
$ man 8 spamd
$ man 5 spamd.conf
$ man 8 spamd-setup
$ man 8 spamdb
$ man 8 spamlogd
$ man 5 syslog.conf
```

```
# vi /etc/rc.conf.local
spamd_flags="-v -5 -G 10:4:864 -l 192.0.2.13"
```

Spamd is served with alias.

```
# vi /etc/hostname.internal_interface
inet alias 192.0.2.13 subnet_mask broadcast_address
```

Synproxy state, configurable at packet filter, can prevent SYN-flood attacks – here are the pertinent `pf.conf` lines: see Listing 1

Configure spamd.conf per your preferences. Examples are provided in the sample configuration file in the `/etc/mail` directory. Next, make spamd run from crontab:

```
# crontab -e -u root
31    0-31/4  *     *     *     /usr/libexec/spamd-setup
```

Set a spamtrap for mail arriving to anything other than *example.org*.

```
# vi /etc/mail/spamd.alloweddomains
example.org
```

The log files can be modified for easy reading.

```
# touch /var/log/spamd
# touch /var/log/spamlogd
# vi /etc/syslog.conf
!spamd
daemon.err;daemon.warn;daemon.info       /var/log/spamd
!spamlogd
daemon.debug                             /var/log/spamlogd
```

Now you can check for interesting entries in your log files (see Listing 3).

Consider changing `newsyslog.conf` per your environment. Let us reboot the gateway.

```
# shutdown -h now
```

Mail Server Configuration

Add the packages alpine, procmail, and bogofilter to the mail server. Bogofilter for this example only succeeds or exits. Procmail can be specific to your user account:

```
$ cat .procmailrc
:0fw
| bogofilter
#!-- where the mail is going, per procmail:
#:0
#* ^TO_.*
#/var/mail/your_user_account
#your_user_account
```

Before the change from sendmail to smtpd, ensure the mail queue is empty.

```
# sendmail -bp
```

Stop sendmail.

```
# pkill sendmail
```

Make changes to the mail wrapper.

```
# vi /etc/mailer.conf
    sendmail        /usr/sbin/smtpctl
    send-mail       /usr/sbin/smtpctl
    makemap         /usr/libexec/smtpd/makemap
    newaliases      /usr/libexec/smtpd/makemap
# vi /etc/rc.conf.local
    sendmail_flags=NO
    smtpd_flags=
```

Edit `/etc/mail/aliases` and run the newaliases command.

Maps can be named freely, here `bigD` is used. So edit `/etc/mail/bigD`; e.g.,

```
your_alias: your_user_account
```

Listing 2. *Packet Filter Modification*

```
mail_server = "192.0.2.9"
spamd_proxy = "192.0.2.13"
table <spamd-white> persist
table <spamd-greytrap> persist
match in on egress inet proto tcp from !<spamd-white>
                    to \
    $mail_server port 25 rdr-to $spamd_proxy port spamd
pass out quick on egress inet proto tcp to any port
                    smtp modulate
    state
pass in log on egress inet proto tcp from any to
                    $spamd_proxy port
    spamd modulate state

pass in log on egress inet proto tcp from <spamd-white>
                    to \
    $mail_server port smtp synproxy state
```

Listing 3. *Log File Samples*

```
$ cat /var/log/spamlogd
Apr 18 10:13:46 gate spamlogd[15106]: outbound [centralServer.com]
Apr 20 15:40:17 gate spamlogd[3531]: inbound 80.13.214.150
Apr 23 12:55:21 gate spamlogd[3531]: inbound 175.180.69.7
$ cat /var/log/spamd
Apr 17 23:54:45 gate spamd[23239]: listening for incoming connections.
Apr 17 23:54:45 gate spamd[17488]: got suffix example.org
Apr 20 15:40:17 gate spamd[23239]: 80.13.214.150: connected (1/0)
Apr 20 15:40:30 gate spamd[23239]: (GREY) 80.13.214.150: \
   <hotmabox@yahoo.com>  -> <hotmabox@yahoo.com>
Apr 20 15:40:30 gate spamd[23239]: 80.13.214.150: disconnected after \
   13 seconds.
Apr 23 12:55:21 gate spamd[23239]: 175.180.69.7: connected (1/0)
Apr 23 12:55:33 gate spamd[23239]: (GREY) 175.180.69.7: <gigi.wu@gmail.com> \
    -> <vbibiorm@gmail.com>
Apr 23 12:55:33 gate spamd[23239]: 175.180.69.7: disconnected after \
   12 seconds.
```

Implement the map in your files.

```
# makemap -t aliases -o /etc/mail/bigD.db /etc/mail/bigD
# chmod 640 /etc/mail/bigD.db /etc/mail/bigD
# chgrp _smtpd /etc/mail/bigD.db /etc/mail/bigD
```

Configure OpenSMTPD.

```
# vi /etc/mail/smtpd.conf
mail_if = "your_networkCard"
listen on $mail_if
map "aliases" { source db "/etc/mail/aliases.db" }
map "bigD" { source db "/etc/mail/bigD.db" }
accept for local alias aliases deliver to mbox
accept from all for domain "example.org" alias "bigD" \
   deliver to mda "procmail -f -"
accept for all relay via "smtp.centralServer.com"
```

Check smtpd.conf for validity.

```
# smtpd -n
```

Start the daemon.

```
# smtpd
```

To stop, and start anew:

```
# pgrep smtpd
# pkill smtpd
# pgrep smtpd
```

If things are convenient, consider a restart of the mail server. Check the result.

```
# cat /var/log/maillog
```

or:

```
# gzcat /var/log/maillog.0.gz | grep smtpd
```

and:

```
# smtpctl sh s
```

Invoke the telnet command to ensure that connections are successful.

```
$ telnet
telnet> open mail.example.org 25
telnet> close
```

Have fun! For topics not covered here or if this implementation was not clear to you, perhaps the OpenBSD *misc* list has some helpful archives.

DARREL LEVITCH

Darrel resides in Lyndon, Kentucky, USA and designs networks. He enjoys modifying existing infrastructure with features found in Berkeley Software Distributions and installed on commodity hardware or in virtual environments.

Hakin9

PRACTICAL PROTECTION

IT SECURITY MAGAZINE

License Wars!

When I sat down to brainstorm on this month's article, I decided to write about something out of the ordinary. Obviously, the topic had to be related to BSD, yet, I was determined to touch upon something that is a bit above than just being 'geeky'. Why? Simply to make BSD fanatics proud, and at the same time show non-BSD fans how great the world of BSD is!

What you will learn…

- Why BSD License beats GPL, in simple terms!

What you should know…

- Details of BSD License and GPL

Last week I was talking to my Linux-user friends, and gradually the conversation shifted to GPL v/s BSD licensing. The conversation was a short one indeed, and I suppose BSD users might be interested in what I had put forth.

What makes an open source project a success story? Of course, it has to fulfil a need, but apart from that, it must attract users. Secondly, it should have the ability to attract and retain developers as well. You might wonder: users, yes they are needed. But why should it attract developers? Naturally, the more developers a project has, the more work gets done! Yet, that shouldn't concern us. The point to be noted is how those developers' contributions are treated. As a general rule, most open source projects let developers retain the copyright to their respective works. Simply put, this means that if someone wishes to purchase the copyrights of a given project, then all the concerned developers will have to agree. There are no alternatives in such a scenario – even if one of the developers refuses, the copyright purchase cannot be completed.

Seems simple enough so far? Well, visualize a fairly large open source project, that has many people who have made (or are making) substantial contributions in one or the other. Needless to add, acquiring copyrights of such a project will be next to impossible. Need examples? Good ol' Linux. Correction, it's GNU/Linux. The copyrights are distributed among so many people that even Linux developers have lost count.

Still not convinced? Try MySQL, a product wherein the creators did not allow contributions unless the copyrights were assigned to them. The bottom-line read that a MySQL contribution was a mere question of the number of dollars needed to swallow it up. Again, just another drawback of the GPL.

So let us get back to the question of what makes an open source project a success story? True, your next door economist will reply that your clients must be satisfied with your work, because if you have one satisfied customer, you're likely to have more. As they say, if you cannot sell it, you can't give it for free either as no one would take it!

But economics lessons apart, we, as open source enthusiasts, know that an open source project is as good as its license. Unless the license offers the users full freedom, the chances of success are limited. Once again, if you need some examples, allow me to cite PostgreSQL and Python – both licensed under the BSD license. And both of them are going strong.

So coming back to the question, what makes an open source project a success story? Truth is, it's the proper licensing. Unless you give your users the freedom they are seeking, you will not be able to make a mark, at least as an open source entity.

To finish up, let us look at Django, a Python-based Web framework, first released in mid-2005. It was developed as a framework for a news site for the LJ World Journal by Adrian Holovaty and Jacob Kaplan-Moss, and others. But its popularity soared exponentially once they open sourced it. How? It was given a BSD license (don't blink, you read that right). Currently, Django has its code in an open repository, backed by a publicly accessible wiki, separate mailing lists for users and developers and an IRC Channel.

And so, Django today is not only the most popular Python based Web framework but also one of the most well known adherents of BSD license. The success of Django shows that the BSD license is the way to go!

With that said, allow me to sum it up for you. The bottom-line is that the BSD-style licence, though not free from its share of technicalities, is yet to encounter a *violation*, while GPL has had numerous *violations* thus far. To put it the other way, using a permissive BSD licence, you can ensure you get a tension-free sleep.

The reason for the same is that BSD licence is practically public domain, and it does not speak in extinct languages. Here, FREE means FREE-as-in-real-life, period.

Now, before taking leave, let me disclaim. I admit I am not a lawyer. But the above description definitely is not technical (and it wasn't meant to be technical, either). While I believe most of us would agree that the BSD licence eats GPL for lunch, there might be a few who'd think otherwise. If you're one among them, feel free to share your views with us!

SUFYAN BIN UZAYR

Sufyan is a 20-year old freelance writer, graphic artist, programmer and photographer based in India. He writes for several print magazines as well as technology blogs. He is also the Founder and Editor-in-Chief at http://www.bravenewworld.co.nr He can be reached at http://www.sufyan.co.nr

Allocating Dynamic Memory with Confidence

Embedded software applications face many challenges that are not present on desktop computers.

What you will learn…

- How to analyze worst-case memory footprint of an application
- Characteristics to look for in a memory allocator
- How to benefit from dynamic allocation without risking out-of-memory errors

What you should know…

- Basic C / C++ skills and how to allocate / free memory

A device with a dedicated function is expected to perform that function consistently, no matter how complex the task is at the software level.

Users will put up with occasional slowdowns and crashes on a desktop computer, but devices are held to a higher standard, especially when they are part of a mission-critical system. Memory allocation is an important factor for providing the necessary performance and reliability on an embedded device.

On a general-purpose computer, well-designed applications allocate memory on-demand, so that each application only uses as much memory as it needs at any given time. If an application needs a large amount of memory, the user is expected to stop using other applications until it is finished.

Embedded devices are typically designed to perform a fixed set of tasks. The user may not even realize that there are anything like applications running on a device. Devices that do not support demand-paging will simply fail when memory is full. Even an unexpected drop in performance can be frustrating, and in some cases dangerous. For that reason, well-designed applications on embedded systems often preallocate memory so that performance is consistent and failure is prevented.

However, for complex applications it is not always possible to predict all memory requirements in advance. Instead, the application can be analyzed to determine its worst-case memory consumption and allocate a buffer of that size in advance. Such analysis can be difficult, especially when starting from scratch.

Storing, organizing, and sharing data makes up a large part of the memory requirements for an application. A device application can use an embedded database library to manage memory more effectively, by both imposing bounds on memory usage and analyzing worst-case behavior in a consistent way. The database library can handle all the details of reading, writing, indexing, and locking data within a predictable footprint, so that the application's own memory requirements are greatly reduced.

Designing for Predictable Memory Usage

Reliable embedded devices depend on predictable behavior. For memory allocation, this requires knowing how much memory an application will need in the worst case, and then finding ways to reduce that amount. To do this, an application developer needs to follow a good memory allocation strategy, measure memory consumption under a variety of representative configurations, and analyze the results.

Total memory consumption includes not only the memory requested by the application, but also the overhead of the dynamic memory allocator itself. Some

allocators are more susceptible to fragmentation than others, so it is important to know what kind of allocator the application is using. Most operating systems use a general-purpose allocator that performs well on average, but that may badly fragment memory at unexpected times. On such platforms, a bounded allocator can be used in each application to limit allocation overhead.

Memory Allocation Strategy

A useful strategy to avoid memory fragmentation is two-phase allocation. Under this strategy, large and long-term object are allocated first so that they are guaranteed a place in memory. Small and short-lived objects are allocated in the second phase because they are less likely to fail even if memory is fragmented. In this way, there is little risk that an allocation will fail merely because no contiguous region of memory is large enough.

Both the application code itself and any libraries that allocate memory should apply this strategy. Otherwise, the worst-case behavior of the application cannot be predicted accurately. Even a bounded allocator cannot provide any guarantees if an embedded library only imposes soft limits on its allocation behavior.

Statistics Collection and Analysis

When measuring memory allocation behavior, the most important statistics to collect are the largest amount of memory allocated at any one time and the size of the single largest allocation, including allocator overhead. Other statistics may also be valuable for certain memory allocators.

The amount of memory used by an application usually depends on how it is configured and how it is used. Statistics should be collected for several different configurations that represent all of the extreme memory use cases. The application should also be divided into discrete operations that can be tested individually, so that results can be calculated without simulating all possible combinations.

By knowing an application's total memory consumption, it is possible to allocate a large enough memory pool when an application is started to satisfy all allocation requests for the life of the application. Provided that operations run sequentially, one by one, the memory consumption is defined as the largest consumption of any individual operation. If operations could overlap, the maximum memory consumption is defined as a sum of all the operations that could be run concurrently.

Managing Memory Effectively with ITTIA DB

ITTIA DB SQL is an embedded database library that is specifically designed for devices and embedded systems. For example, memory allocation in ITTIA DB SQL follows the two-phase principle, so that memory requirements are consistent and predictable.

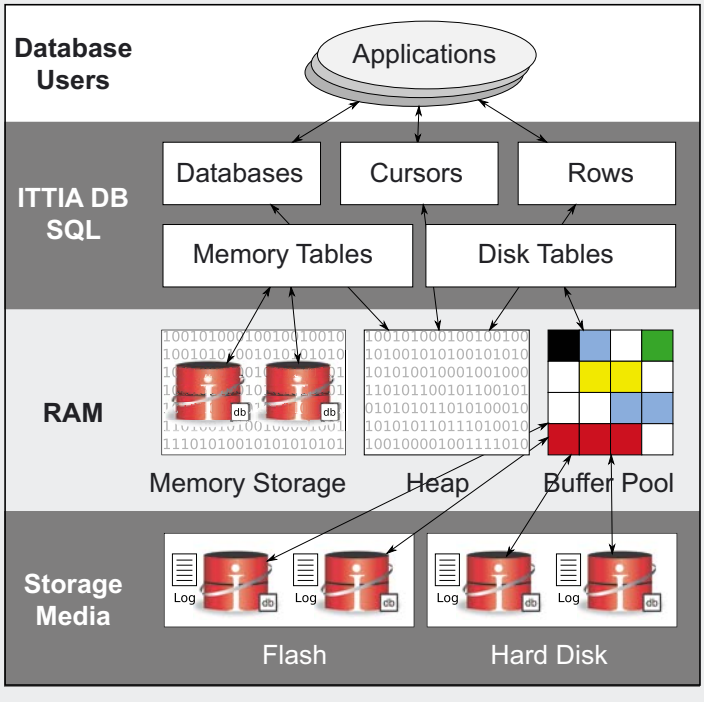


Figure 1. *Memory model for ITTIA DB embedded database*

Table 1. *Actual memory consumption and worst-case estimate calculations*

Workload	Statistics	Static Overhead	Measurements	Weigh and Sync	Total
100	Actual	186kB	5kB	2kB	193kB
	Estimate	579kB	5kB	2kB	586kB
1,000	Actual	186kB	5kB	2kB	193kB
	Estimate	579kB	5kB	2kB	586kB
100,000	Actual	186kB	5kB	2kB	193kB
	Estimate	579kB	5kB	2kB	586kB

ITTIA DB SQL also includes a built-in allocator that can be enabled to restrict all database allocations to preallocated segments of memory. The built-in memory allocator has proven limits on memory fragmentation overhead, and provides statistics so that worst-case behavior can be measured for each database-driven application.

Other statistics can also be collected, such as the number of database resource handles opened by the application and the number of locks used to provide safe, efficient shared access. These provide additional insight into application behavior, which can be used to reduce the memory footprint.

Use Case: Weigh Station

At a weigh station, trucks are moved onto a large scale and the measurement is collected, stored, and later transferred to a back-end system. A device is used to read sensor data from the scale and associate weights with trucks. Trucks can be grouped together into a train, so that data is not sent to the back-end system until an entire train is complete.

In this scenario, an embedded database can be used to log sensor readings continuously in one thread while trucks are identified and synchronized with the back-end system in another thread. The application code only needs to operate on one truck and one sensor reading at a time, so dynamic memory allocation can be avoided everywhere except in the database itself. In this way, analyzing the dynamic memory consumption of the database is sufficient to determine the requirements of the entire application.

To determine the amount of memory used by the database, consumption is measured sequentially for three separate operations: opening database connections, capture of scale measurements, and truck data entry and transfer. The memory consumption for the application is the total for these three operations, since the measurement and truck threads can be run concurrently.

When the simulation is run under various workloads, memory consumption is stable no matter how many trucks are weighed or measurements captured. The largest allocations are performed during start-up by opening the database connections and cursors. The measurement and weigh/sync threads contribute very little to the memory footprint. Statistics for both actual memory usage and the upper bound on estimated memory consumption are captured from the built-in memory allocator in ITTIA DB SQL (Table 1).

Conclusion

Memory allocation behavior can have a significant impact on the performance and reliability of an embedded device. Extreme measures such allocating all memory statically at compile-time are extremely restrictive, and not necessary if developers are willing to apply some analysis. For software libraries where the worst-case behavior is not clearly defined, applications can run out of memory unexpectedly even with a bounded memory allocator. An embedded database that provides robust memory management features, like ITTIA DB SQL, can be used to limit and analyze the most dynamic allocations in a device application.

RYAN PHILLIPS

Ryan Phillips is a Lead Engineer at ITTIA with special focus in embedded systems and database technologies. He has over a decade of software development experience.

In the next issue:

- **BSD Certification**
- **FreeBSD and LDAP**
- **and Other !**

Next issue is coming in June!

Enterprise Open Source Storage:

FreeNAS with ZFS has arrived...

Reliable Scalable Affordable

1-855-GREP-4-IX | http://www.iXsystems.com/FreeNAS

iX systems®