MAGAZINE

BSD

FOR NOVICE AND ADVANCED USERS

Adobe® PDF Magazine Version

BSD'S AND SOLARIS

INSIDE

VOL.4 NO.1
ISSUE 01/2011(18)
1898-9144

iX-Green Neutron 10G Rackmount Servers: Energy Efficient, Next-Generation 10 Gigabit Performance

› INNOVATIVE TECHNOLOGY › HIGH DRIVE DENSITY › LOW POWER ENVELOPE

The new iX2216-10G 2U Rackmount server features built-in Intel® 10 Gigabit Ethernet Adapters, resulting in increased performance in an energy-efficient design.

iXsystems announces a new 2U Rackmount Server that is equipped with energy-efficient, next-generation 10 Gigabit controllers. 10 Gigabit Ethernet provides unparalleled throughput for virtualized servers and unified networks, and has emerged into the enterprise and service provider space as a viable, low-cost networking alternative to Fibre Channel.

The iX2216-10G features the latest Intel® processors based on the 32nm and 45nm next-generation microarchitecture, which represents the next step in intelligent performance, automated energy efficiency, and flexible virtualization.

The iX2216-10G also features dual on-board Intel® 82599EB 10 Gigabit SFP+ Ports, dual on-board Intel® 82576 Gigabit Ports, dual Intel® Xeon® Processors 5600/5500 series, and 18 DIMM slots supporting up to 192GB of DDR3 ECC Registered memory. This system would be ideal for HPC, Data Center, Virtualization, Clustering, and Cloud Computing applications.

iXsystems is dedicated to bringing the energy saving advantages of 2.5" disk drives and high-efficiency power supply technology to our customers. With the iX2216-10G, you can have up to 16 x 2.5" high-rotation SAS or SATA drives. The redundant AC-DC high-efficiency 920W, 94%+ certified power supplies, will also help lower your total cost of ownership.

For more information on the **iX-2216-10G**, or to request a quote, visit:

http://www.iXsystems.com/GN10G

Ideal for Slot-Constrained Environments

10G Rackmount Servers in the iX-Green Neutron server line 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.

The iX2216-10G features high drive density with sixteen 2.5" SAS or SATA drive bays, while still leaving sufficient capacity within the chassis to integrate DVD-ROMs or other optical devices.

- With the latest multi-core 10 GbE server platforms, customers can now realize up to 2.5 times the bandwidth supported by earlier generation platforms.

- 10GbE is mainstream: More cost effective than Fibre Channel and Infiniband, faster than Fibre Channel.

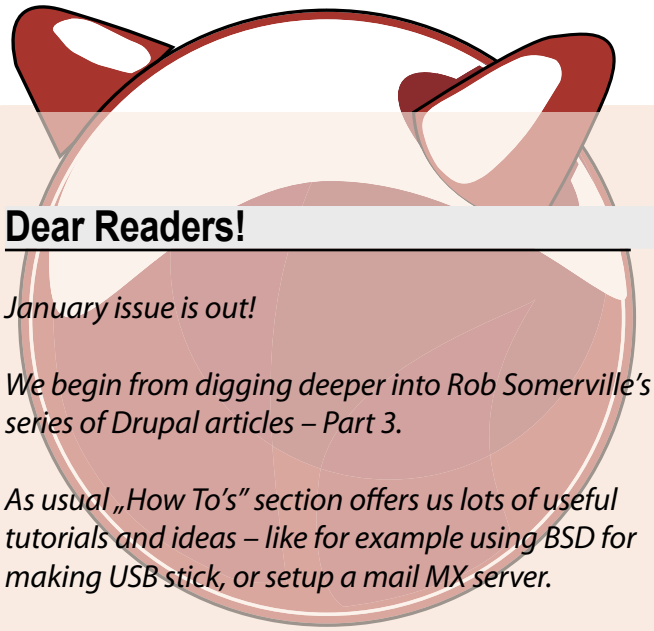
- 50% lower per-port cost compared to industry standard 10GbE add-on card.

Key features:

- Supports Dual 64-Bit Six-Core, Quad-Core or Dual-Core, Intel® Xeon® Processor 5600/5500 Series
- 2U Form Factor with 16 Hot-Swap SAS/SATA 2.5" 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 CD-ROM / DVD-ROM and/or 3.5" DVD-RW Drive
- 920W high-efficiency (94%+) AC-DC Redundant power supplies with PMBus and I2C

Call iXsystems toll free or visit our website today!

+1-800-820-BSDi | www.iXsystems.com



Dear Readers!

January issue is out!

We begin from digging deeper into Rob Somerville's series of Drupal articles – Part 3.

As usual „How To's" section offers us lots of useful tutorials and ideas – like for example using BSD for making USB stick, or setup a mail MX server.

In „Let's Talk" we can find the main article written by Petr Topiarz, and after that get a bit less serious with Sufyan :).

I believe you will find this issue a promising start into 2011. Please mail us with your feedback – we're always interested in your opinions regarding the magazine.

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:
Anton Borisov, Joshua Ebarvia, Diego Montalvo, Francisco Reyes, Eric Schnoebelen, Juraj Sipos, Rob Somerville, Petr Topiarz, Sufyan bin Uzayr, Girish Venkatachalam

Art Director:
Ireneusz Pogroszewski

DTP:
Ireneusz Pogroszewski

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

National Sales Manager:
Ewa Łozowicka
ewa.lozowicka@software.com.pl

Marketing Director:
Ewa Łozowicka
ewa.lozowicka@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

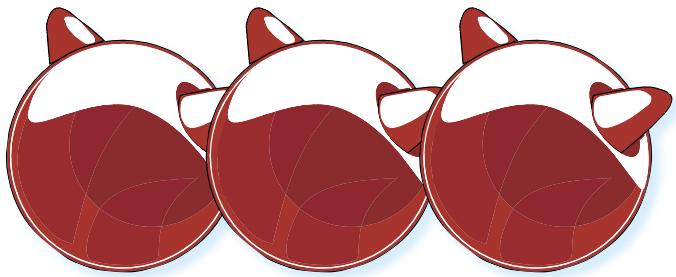
Software Press Sp z o.o. SK is looking for partners from all over the world. If you are interested in cooperation with us, please contact us via e-mail: *editors@bsdmag.org*

The editors use automatic DTP system AUPOS

Mathematical formulas created by Design Science MathType™.

Get Started

How To's



Let's Talk

Drupal on FreeBSD

Part 3

Continuing the series on the Drupal Content Management System, we will look at creating a store front for our new website using CCK and Views.

What you will learn…

- How to expand Drupal with custom content and views

What you should know…

- Basic BSD system admin skills and how to install/administer Drupal CMS (Parts 1 & 2)

Traditionally with standard HTML pages, content was written either in an editor or generated by a WYSIWYG (What You See Is What You Get) web design tool such as Dreamweaver. With the increased complexity and size of websites, managing sites using this technique rapidly became very problematic, and a number of technological advances were offered in response to this problem. From a technological perspective it is important to separate content from style, and this was achieved using CSS (*Cascading Style Sheets*) which is an extremely powerful way of allowing the the same content to be displayed in a variety of ways (See CSS Zengarden References). Compared to the overhead involved editing individual documents, this overcame the limitations of embedding styles within content, and allowed the web developer to re-skin a website by changing the centralised CSS or dynamically at the click of a button.

This did not address the problem of how to quickly modify content that is common across a website (for instance a common document footer that contains company name and contact details) or provide access

Figure 1. *Default Drupal Content types*

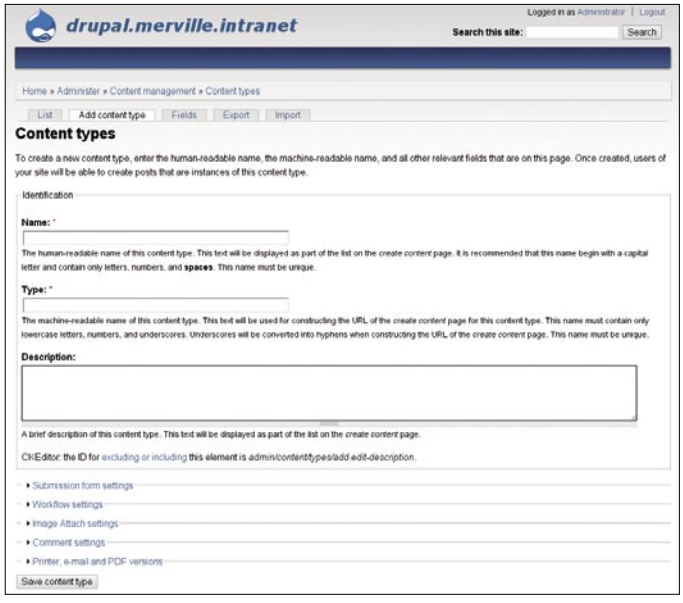


Figure 2. *Adding a new content type*

to disparate database content as the original HTML standard did not offer this. Server Side Includes and languages such as Perl, PHP etc. solved the problem, but development time and cost was required to truly make a website dynamic. It was not until the integration of the database, the web-server and the GUI for managing the content itself that the Content Management System really came into its own from the end user perspective. One major stumbling block in providing dynamic fresh websites was finally overcome – how to provide the functionality that allowed the user inexperienced in writing code to rapidly update content and style without technical knowledge.

However, early CMS's still had a major limitation – while generating an individual page is relatively trivial, generating custom content from a DB is not so straightforward. The opposite problem than the HTML

Figure 3. *The new Product content type*

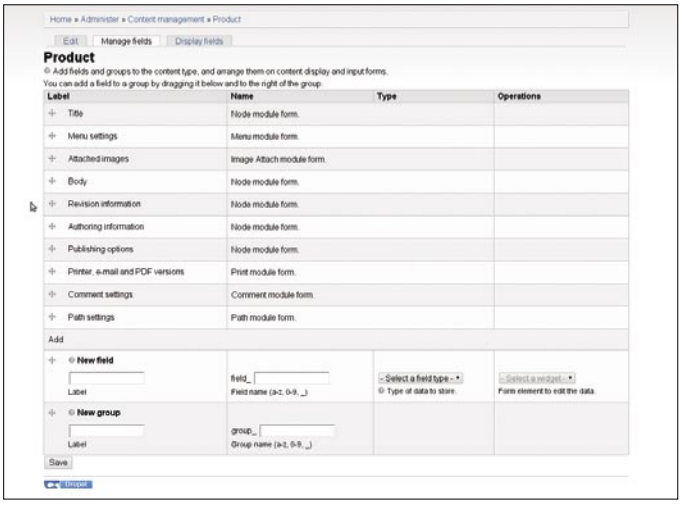


Figure 4. *Adding a custom field*

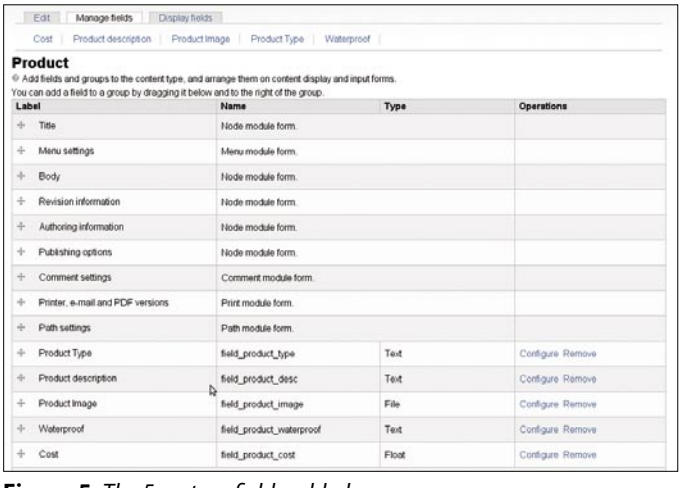


Figure 5. *The 5 custom fields added*

Table 1. *Default content types*

Image	Used for displaying images or Figures
Page	"Static" content that will not have any comments or postings, e.g. an about us page
Story	Ideal for informative content, e.g. press releases, news etc. that requires user feedback via comments.

Table 2. *Content Type definition*

Group	Field	Value
Identification	Name	Product
	Type	product
	Description	Custom content for our product line.
Submission Form Settings	Minimum number of Words	25
	Explanation or submission guidelines	Add new products using this template
Workflow Settings	Published	Disabled
	Promoted to Front Page	Disabled
	Create New Revision	Enabled
Comment Settings	Default Comment Settings	Disabled

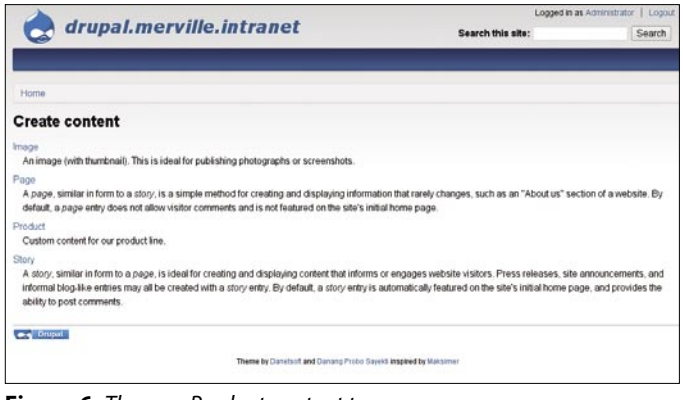


Figure 6. *The new Product content type*

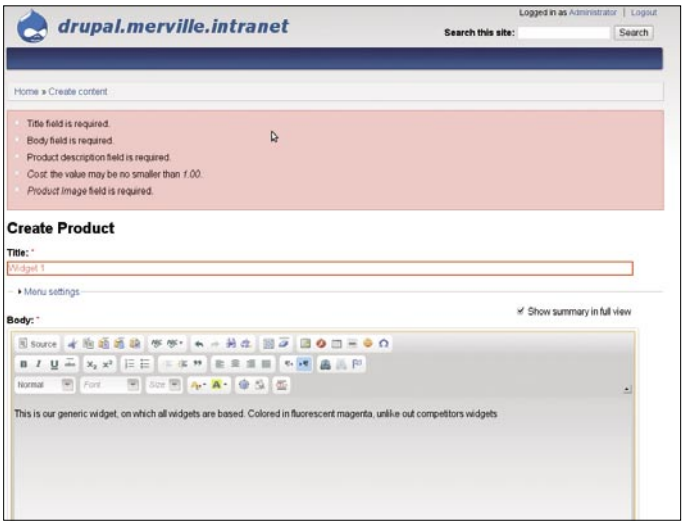


Figure 7. *Input validation failed*

style issue occurred – the database content is too loosely coupled. For instance, a product list on a widget website could have a page for each widget type, and a custom summary page listing each widget, description and cost etc. The end user would have to manage the content for each widget and any changes to the cost would have to be reflected in summary page as well. If they forgot to update the summary or widget page, the information would be inaccurate. The traditional approach to this would be for developer to write custom code to allow the user to input each widget attribute into a database, and write a custom page to dynamically pull this centrally stored data out. If in the future though, the user required an additional field (e.g. a waterproof widget) the developer would need to be consulted to make the modifications to both the template and the underlying code.

Drupal provides two very powerful modules, CCK (*Content Construction Kit*) and Views that allows the developer or webmaster to quickly generate custom content types for holding and displaying user defined database fields without

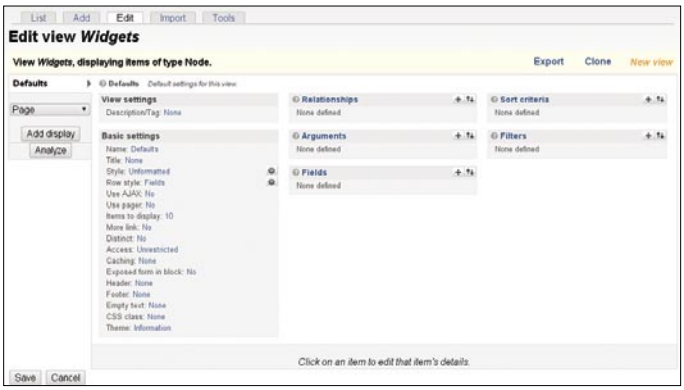


Figure 8. *Newly created Widgets view*

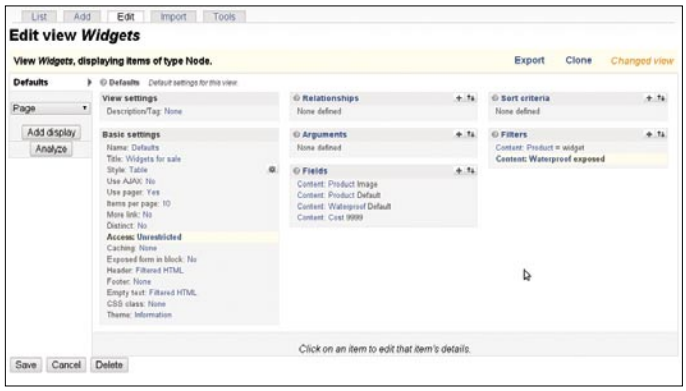


Figure 9. *Settings, Fields and Filters configured*

coding – such as our widget product line. In this tutorial, we will build a crude non-production storefront teaser for *Widgets.com*, displaying a detailed description page for each widget and a summary page with images, description, price etc. In reality, a production e-commerce system would utilise a product such as Ubercart and Drupal, but as the shopping cart output format is so ubiquitous this was used as an example.

Required modules

The CCK, Views and ImageField and FileField modules need to be installed and enabled.

Creating a new content type

Drupal by default provides the following content types (Figure 1).

We will create a new content type called Product. Once logged into Drupal as Administrator, *Navigate to Home* » *Administer* » *Content management* » *Content types* and click on Add Content Type (Figure 2) and complete the fields as detailed. Click on the Group hyper-link to expand the options as required.. Save, then click on the manage fields link and add the new fields saving each field in turn and adding additional values as desired. (Table 1-2 Figure 3-5).

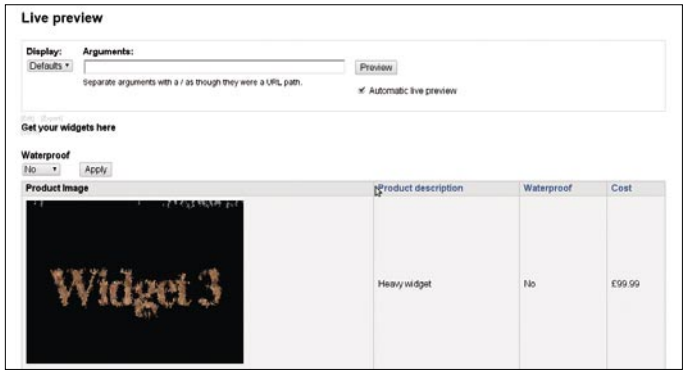


Figure 10. *Non-waterproof filter in default view preview*

Creating content

Once you have defined the custom field types, visit *Home » Create content* and add some new Product pages. If you omit vital information, or try to input incorrect information (e.g. a widget that has zero cost) you should get an error message (Figure 6-7).

Displaying the content using views

Visit *Home » Administer » Site building » Views* and create a new view with the parameters in Table 4 . (Figure 8). Click on Add or Update to commit each field change, save to commit the entire view. Sort the fields view order by clicking on the up and down arrow then click and drag each item to re-order. The automatic live preview should be enabled by default, so you should see your results up until you add the last filter (Figure 9-10). If the preview is not displayed automatically, click on Preview. The last filter (marked in red) prevents unpublished content being exposed to visitors, and as we have disabled publishing the product nodes by default no storefront items will be available until they are authorised. This would be required in in a production environment, an email could be sent to webmaster on a product node commit by an

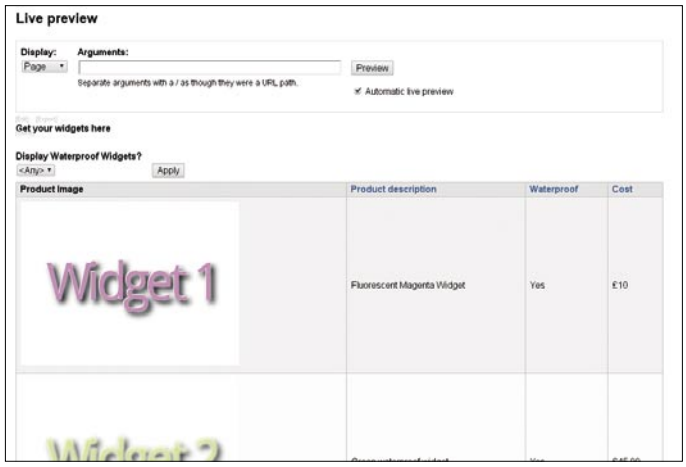


Figure 11. *Unfiltered page preview*

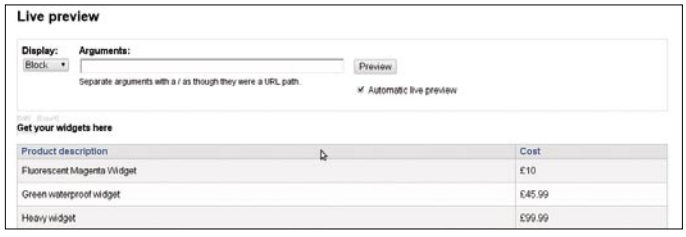


Figure 12. *Block preview*

Table 3. *Field definitions*

Label	Field Name	Field Type	Widget	Additional Values	
Product Type	product_type	Text	Select List	Help text	Demonstration product types for our storefront. As we are only selling widgets, we will only have one item here but this is included for future expansion.
				Required	Enabled
				Allowed Values List	widget|Widget
				Product Type	Widget – Note Field type needs to be saved first to display this value
Product description	product_desc	Text	Text field	Required	Enabled
Product Image	product_image	File	Image	Required	Enabled
				Minimum Resolution	320x240
Waterproof	product_ waterproof	Text	Checkboxes / radio buttons	Help text	Is this product waterproof?
				Allowed values list	Y|Yes N|No
				Default value	No
				Required	Enabled
Cost	product_cost	Float	Text field	Required	Enabled
				Minimum	1
				Maximum	99,99
				Prefix	£

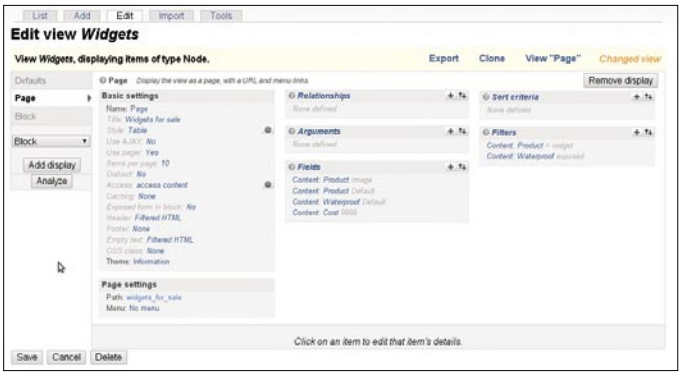


Figure 13. *Final Page view for Widgets*

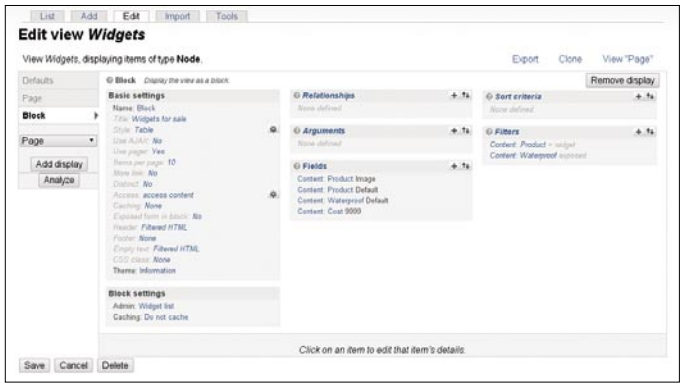


Figure 14. *Final Block view for Widgets*

end user to allow editorial control. For the purposes of the tutorial this filter can be omitted, but regardless of this the correct view permissions will need to be added for the following fields:

- `field_product_cost`
- `field_product_desc`
- `field_product_image`
- `field_product_type`
- `field_product_waterproof`

which is found under *Home* » *Administer* » *User management*. Please resist the temptation to give unauthorised users access to all views, this may mean fewer mouse clicks, but it would potentially create a

Table 4. *Views settings*

Views Parameters			
View name	Widgets		
View Description	All widgets available on widgets.com		
Group	**Field**	**Value**	**Additional Values**
Basic settings	Title	Widgets for sale	
	Style	Table	Page access options: access content
	Use pager	Full pager	
	Row Style	Field	
	Header	Get your widgets here	
	Empty text	Sorry – there are no items available to view.	
	Access	Access restrictions: Permission *Access	
Filters +	All	Content: Product Type (field_product_type)	Is equal to widget
	Groups:	Node Published	Enabled
Fields +	Content	Content: Cost (field_product_cost) Content: Product Image (field_product_image) Content: Product description (field_product_desc) Content: Waterproof (field_product_waterproof)	Enabled Enabled Enabled Enabled
	Content: Product Image	Format	Image
		Link this field to its node	Enabled
Style: table *	Field	Product description – Sortable Waterproof – Sortable Cost – Sortable	Enabled Enabled Enabled
Filters +	Groups:	Content: Waterproof (field_product_waterproof) – Allowed values	Expose Label: Display Waterproof Widgets?
	Groups:	Node Published	Enabled

security risk by overriding the more granular field level security.

We now have a default framework for our view, but this is not much use on its own so from it we will now create 2 displays, a Page and a Block. The Page view is what visitors will see as our storefront landing page, and the Block will display our widget descriptions and cost teaser. To do this:

- Add a Page and Block display.
- Add the path `widgets_for_sale` to the Page view.
- Override the default field views for block and exclude the following from being displayed:
  - Content: Product Image field
  - Content: Waterproof field
  - Content: Cost fields
- Add Widget List to the block under Block: Block admin description.
- Finally, under Fields Content: Product Default enable Output field as Link and change the path to `widgets_for_sale` for the Block view.

Figure 15. *Published product pages*

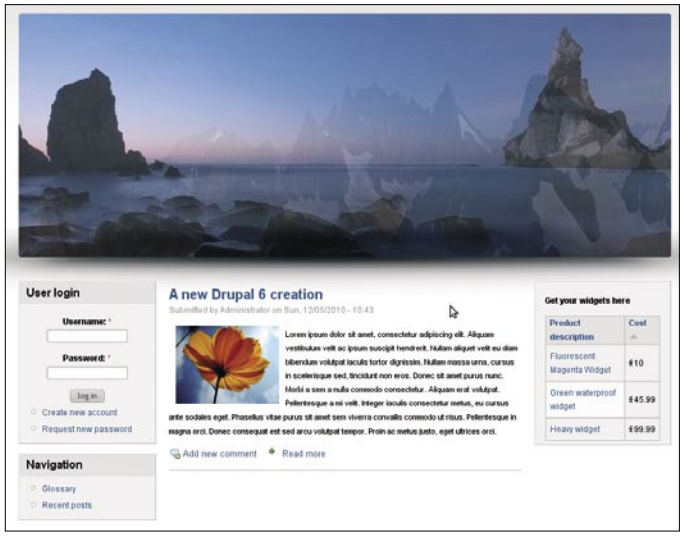


Figure 16. *Widget product teaser block*

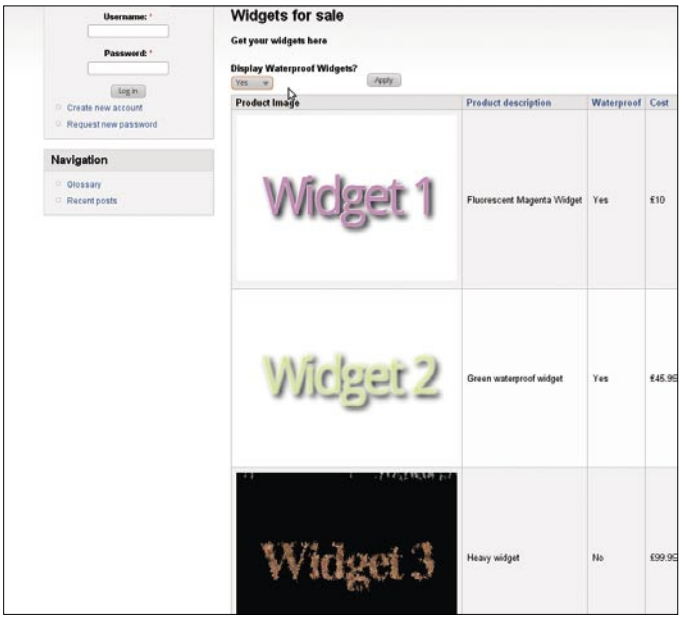


Figure 17. *Product landing page with filter and sort*

This will redirect the user to our storefront when they click on the product. The resulting previews and view settings can be seen here (Figure 11-14).

Adding the block to the front page

Now that we have a cut down list of our widgets, we need to display this to our visitors on the front page. To do this, navigate to *Home* » *Administer* » *Site building* and add Widget list to the right sidebar of the Danland theme. Change the block title to `<none>` and change Show on only the listed pages to `<front>`. Finally, navigate to *Home* » *Administer* » *Content management* and publish all Product pages (Figure 15).

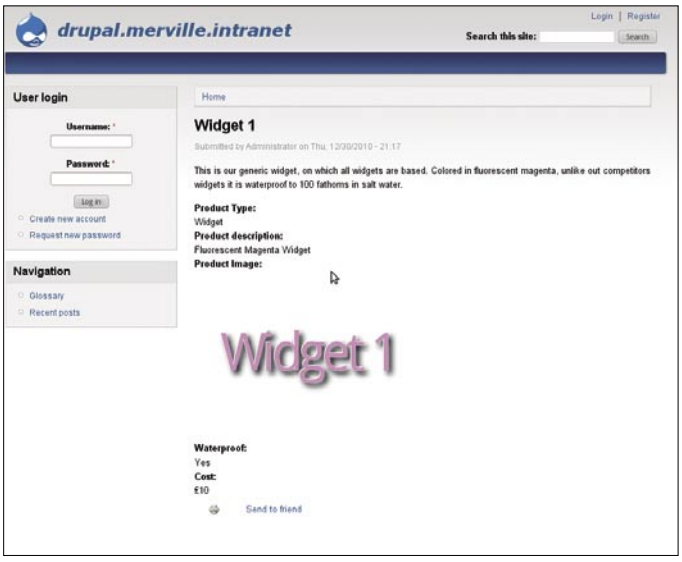


Figure 18. *Product page showing Widget 1 content*

On the 'Net

- CSS Zen Garden *http://csszengarden.com*
- W3C CSS pages *http://www.w3.org/Style/CSS/*
- Ubercart *http://www.ubercart.org*

We should now have the following functionality (Figure 16-18):

1. The ability to add a custom widget product with a mandatory image, with strong input validation, a minimum description length of 25 words and a custom field that defines if the Widget is waterproof with automatic smart URLs (e.g. *http://mysite/content/widget-1*), revisions that doesn't automatically publish and the ability to easily expand the product types
2. A custom cost field that accepts and displays sterling amounts between £1 and £99.99
3. A limitation on the size of product image uploaded
4. A Product teaser block on our front page linking directly to our basic storefront sortable by description and cost
5. A landing page accessible via the smart URL *http://mysite/widgets_for_sale* that allows the user to display all widgets, waterproof widgets or non-waterproof widgets and sort by description, functionality or cost that links through to the widget product pages via the image.

To Do

1. Modify the CSS for widgets page to display more cleanly
2. Add links to the Product description field on the widgets content page
3. Add thumbnails to the teaser

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.


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!

Email MX server in FreeBSD

Configuring FreeBSD as a mail MX server with Postfix

This is a tutorial on how to setup a mail MX server using Postfix.

What you will learn…

- What is a mail MX server
- The two most common types of MX servers
- How to setup a mail MX server

What you should know…

- Basic DNS concepts
- How to become root

What is a mail MX server and why would you want one?

On my article on the January 2010 edition I covered how to setup a mail machine capable of receiving mail and letting the user connect to get his mail. For a small mail setup that would be enough, however for a busy domain or a group of domains it may be better to separate the load across many machines for performance and redundancy.

The function of a mail MX server is to receive mail from outside sources and to forward the mail to the appropriate server. In addition to contributing to capacity having MX servers allow for more flexible scenarios such as the possibility to have the mail server, using POP or IMAP, to be closed off from the Internet to allow only internal users (although that is not a common setup).

Unless otherwise instructed do the work as user root.

We will need to use the port system. If you are new to it check chapter 4 (*http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/ports.html*) of the hankdbook.

A MX server is able to act as a backup or as the primary machine to collect mail for domains. In this article I will cover both.

Initial Notes

For this article we will be setting up a MX server for `<MyDomain>` and the server will have `<ExternalIP>` as it's external IP. Replace with your actual domain and with the actual machine's external IP.

Although a MX server could potentially connect to a database to read values, in this article I will discuss only how to configure the server by reading local files. The benefit of using a database to read the configurations would be the ability to control many machines from a single location; the primary drawbacks would be performance and the possibility of the database becoming a single point of failure.

This tutorial installs the postfix port in batch mode, however if you like you can remove the `BATCH=yes` and do the installs in interactive mode. Be aware that if you do the port install in interactive mode any pre-requisite ports will also be run in interactive mode.

Installing the Postfix port

```
#cd /usr/ports/mail/postfix
#make WITH_TLS=yes BATCH=yes install clean
#rehash
```

The TLS option is needed if you plan to support TLS in the future for higher level of security. Although not covered in this article, if you believe you may need to secure email transmissions in the future then it doesn't hurt to have the support already there. See the links section for more info on TLS and how to use it in Postfix.

After Postfix is installed we need to configure it and disable sendmail. To disable sendmail,the default MTA that comes with FreeBSD, and enable postfix at startup we need to edit `/etc/rc.conf` by adding:

```
sendmail_enable="NO"
sendmail_submit_enable="NO"
sendmail_outbound_enable="NO"
sendmail_msp_queue_enable="NO"
postfix_enable="YES"
```

Disable some sendmail specific daily maintenance by editing `/etc/periodic.conf` and placing the following

```
daily_clean_hoststat_enable="NO"
daily_status_mail_rejects_enable="NO"
daily_status_include_submit_mailq="NO"
daily_submit_queuerun="NO"
```

Configure Postfix as the system mailer. Edit `/etc/mail/mailer.conf` as follows:

```
#
# Execute the Postfix sendmail program, named /usr/local/
                    sbin/sendmail
#
sendmail        /usr/local/sbin/sendmail
send-mail       /usr/local/sbin/sendmail
mailq           /usr/local/sbin/sendmail
newaliases      /usr/local/sbin/sendmail
```

Backup MX

A backup MX machine is the simplest MX configuration; it uses DNS to know to what machine to forward mail to.

MX records have a value and a domain. The lower the value the higher the priority.

Using one of my domains as an example see Listing 1.

In the example above mail will first try to go to *mail.natserv.com*, then *backupmx.natserv.com* and finally *tarbaby.junkemailfilter.com*. The Backup MX in the above case is *backupmx.natserv.com*. If the primary mail server, *mail.natserv.com*, was ever down mail will go to the backup machine and remain there for a few days until the primary server came back up. The amount of time the backup holds the mail for the primary is configurable.

The entry tarbaby.junkemailfilter.com is related to anti-spam measures and I will explain in the section on spam.

The Postfix configuration file is `/usr/local/etc/postfix/main.cf`. Change the entire main.cf as follows see Listing 2.

The list of domains that you allow your backup server to relay for will be in the relay_domains file. Edit `/usr/local/etc/postfix/relay_domains` to look like

```
domain1
domain2
```

The vast majority of spam targets non existing users. In order to prevent the MX server forwarding emails to non existing users we put the list of email addresses in `/usr/local/etc/postfix/relay _ recipients`

Listing 1. *Listing MX DNS entries for a domain*

```
>dig mx natserv.com

; <<>> DiG 9.6.1-P2 <<>> mx natserv.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 34424
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;natserv.com.                   IN      MX

;; ANSWER SECTION:
natserv.com.            28800   IN      MX      0 mail.natserv.com.
natserv.com.            28800   IN      MX      5 backupmx.natserv.com.
natserv.com.            28800   IN      MX      10 tarbaby.junkemailfilter.com.
```

```
user1@domain.com x
user2@domain.com x
user3@domain.com x
user4@domain.com x
```

The value of the second parameter is not relevant; it can be anything.

Notice we listed the recipient list in the `main.cf` as hash: `/usr/local/etc/postfix/relay_recipients`. That means we are using a binary form of the file to speedup lookups. You need to run postmap against any file that is in hash form by calling postmap like

```
#postmap /usr/local/etc/postfix/relay_recipients
```

We did not do a hash for the list of domains because it is usually a much smaller list. The list of users can potentially have hundreds or thousands of entries. Also, different postfix lists can accept hash or a simple lists. Some accept one, but not the other; some accept both.

To finish our initial setup we create an alias file (needed by postfix, even if you don't plan to change it), stop sendmail and start postfix

```
#newaliases
#/etc/rc.d/sendmail stop
#rehash
#postfix start
```

To test postfix is running

```
telnet localhost 25
You should see a prompt
Trying 127.0.0.1...
Connected to < your host name >
Escape character is '^]'.
```

Use CTRL+], the type quit.

Edge MX

If the final destination mail server will not be listed in DNS then the MX servers need to have another way to indicate where to send the emails. One way to accomplish this is using a transport. In the main.cf we add:

```
#transport maps
transport_maps = hash:/etc/postfix/transport
```

In `/etc/postfix/transport` we add:

```
<domain> smtp:<final.server.com>
```

A more concrete example. If we removed entry 0 from the DNS MX entry list above we could have an entry in the transport file like:

```
natserv.com smtp:mail.natserv.com
```

Listing 2. *Postfix main.cf file*

```
relay_domains =/usr/local/etc/postfix/relay_domains
smtpd_recipient_restrictions =
  permit_mynetworks ,
  reject_unauth_destination,
  reject_unlisted_recipient,
  reject_rbl_client hostkarma.junkemailfilter.com=127.0.0.2,
  reject_rbl_client zen.spamhaus.org,
  reject_rbl_client dnsbl-1.uceprotect.net,
  reject_rbl_client ix.dnsbl.manitu.net,
  reject_rbl_client bl.spamcop.net,
  reject_rbl_client psbl.surriel.com,
  reject_rbl_client ubl.unsubscore.com

# You must specify your NAT/proxy external address.
proxy_interfaces =
relay_recipient_maps = hash:/usr/local/etc/postfix/relay_recipients
```

Note that I used a hash file so don't forget to do: `postmap /etc/postfix/transport`.

Any domain you list in the tranport file, needs to have the domain listed in `/usr/local/etc/postfix/relay_domains` and each user listed in `/usr/local/etc/postfix/relay_recipients`.

Anti-spam measures

As you may have noted, in the `smtpd_recipient_restrictions` of the `main.cf` we have lines like:

```
reject_rbl_client hostkarma.junkemailfilter.com=127.0.0.2,
reject_rbl_client zen.spamhaus.org,
reject_rbl_client dnsbl-1.uceprotect.net,
reject_rbl_client ix.dnsbl.manitu.net,
reject_rbl_client bl.spamcop.net,
reject_rbl_client psbl.surriel.com,
reject_rbl_client ubl.unsubscore.com
```

Those are RBLs (*Real Time Black lists*). Using RBLs is one of the simplest ways to reduce spam, however it is possible to block some valid emails so it is important to not use any RBLs that are overly aggressive. The list I have used has proven to have a good balance in my own usage.

Some RBLs have restrictions on commercial usage and large volume usage, so if you are going to use an RBL in a commercial or large volume setting you may need to check with the rules and regulations of the RBL.

In the DNS entry for natserv.com I have tarbaby.junk emailfilter.com in the lowest priority entry. This entry is an anti-spam measure run by project tarbaby(see links section). The MX entry will not accept any emails, but will look for signs that a smtp server is trying to send spam. The tarbaby server ends the connection by telling the sending server that it can not accept the mail. Any well behaved smtp server will try a higher priority MX server. Many spammers do not honor the MX records, so if spammer connects only to the tarbaby server you reduce your spam. Furthermore, even if a server honors MX, but is recognized by the tarbaby server as sending spam it gets added to the tarbaby RBL.

On the 'Net

- *http://www.postfix.org/ – Postfix web site*
- *http://www.networkworld.com/newsletters/gwm/0329gw1.html – How can TLS increase e-mail security?*
- *http://www.postfix.org/TLS_README.html – Postfix TLS support*
- *http://wiki.junkemailfilter.com/index.php/Project_tarbaby – Project TarBaby*

You may wonder why would a server connect to the lowest priority server in the MX order. This is something spammers do often because many mail setups have configured their MX servers in a less secure fashion than their primary servers so spammers sometimes are able to get more spam through by going to the lower priority MX servers. Whenever you setup MX servers you need to make sure your backup MX servers have the same, or higher, anti-spam measures than your primary mail server. As an example compare my RBL list for the domain natserv.com in the MX compared to the actual primary machine.

MX RBL list

```
reject_rbl_client hostkarma.junkemailfilter.com=127.0.0.2,
reject_rbl_client zen.spamhaus.org,
reject_rbl_client dnsbl-1.uceprotect.net,
reject_rbl_client ix.dnsbl.manitu.net,
reject_rbl_client bl.spamcop.net,
reject_rbl_client psbl.surriel.com,
reject_rbl_client ubl.unsubscore.com
```

Primary machine RBL list

```
reject_rbl_client hostkarma.junkemailfilter.com=127.0.0.2,
reject_rbl_client zen.spamhaus.org
```

Notice how I have significantly more RBL servers in the MX machine. My primary machine is very stable and rarely ever is down so rarely ever valid mail needs to go through it. I am able to have stronger anti-spam measures in the backup MX than in the primary server. The 2 RBLs I have listed in the primary have a good ratio of blocking spam, but not blocking valid emails. Some of the other RBLs I list in the MX are more aggressive and have a higher chance of blocking valid emails so I don't use them in the primary server.

FRANCISCO REYES

Francisco Reyes is a system administrator and DBA in NY City. He writes at http://adminlibre.com.

Installing NGINX and PHP 5.3.x

on FreeBSD 8.1

Have been using Apache as my default web server on FreeBSD servers since departing from IIS 4.0 and NT systems in 1999. Apache has always performed great on my installations and give the Apache Foundation great praise.

What you will learn…
- How to configure NGINX and PHP 5.3.x on FreeBSD

What you should know…
- How to install ports and edit configuration files

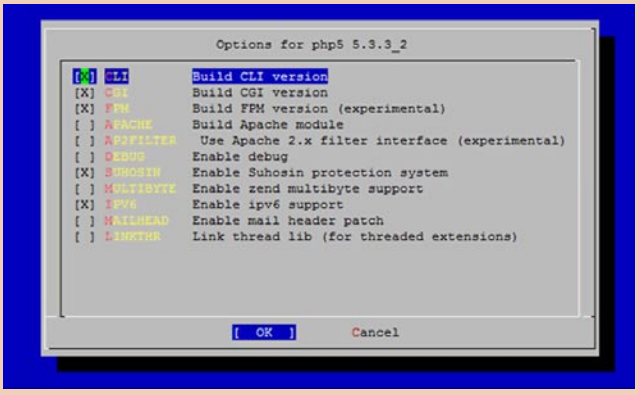
This article is one of trying a newly popular web server, not a migration tool away from Apache . Lately have read alot about NGINX so decided to install it as the default web server for a new web service I am developing.

I did find some helpful tutorials on the Internet, but most dealt with PHP 5.2.x or older. After hours of searching and chatting with other folks it turns out 5.3.x already comes compiled with FastCGI by default.

Using PHP 5.3.x allows you to skip several steps from existing 5.2.x tutorials. No need to install and configure „spawn-fcgi" or other plug-ins since FastCGI is compiled with php 5.3.x

Installing NGINX on FreeBSD is as simple as installing any other port but does take some tweaking in order for it to work correctly with PHP.

01



Installing PHP 5.3 via FreeBSD Port

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

It is very important to choose the *Build FPM version* which includes the FastCGI Process Manager.

```
# make install clean
```

Wait for installation to complete.

```
# ee /etc/rc.conf
```

```
Type the following: php_fpm_enable="YES"
```

```
Save
```

```
# shutdown -r now
```

Upon reboot you will have a successful installation of PHP 5.3.x on your FreeBSD system.

02 Installing NGINX via FreeBSD Port

```
# cd /usr/ports/www/nginx
```

As recommended in other tutorials install the following modules (you may need additional modules for your specific installation:

```
HTTP_MODULE
HTTP_REWRITE_MODULE
HTTP_SSL_MODULE
HTTP_STATUS_MODULE
# make install clean
```

After NGINX is done installing and compiling you may want to automatically start the web server when FreeBSD boots up.

```
# ee /etc/rc.conf
Type the following: nginx_enable="YES"
 Save
# shutdown -r now
```

Upon reboot you will have a successful installation of NGINX on your FreeBSD system.

03

```
server {
    listen       80;
    server_name  localhost;
    #server_name yourserver.com www.yourserver.com;

    #charset koi8-r;

    #access_log  logs/host.access.log  main;

    location / {
        root   /usr/local/www/your-dir;
        index index.php index.html index.htm;
    }
```

Configuring NGINX for PHP Handling

In order for NGINX to properly handle PHP files you will have to configure the nginx.conf file.

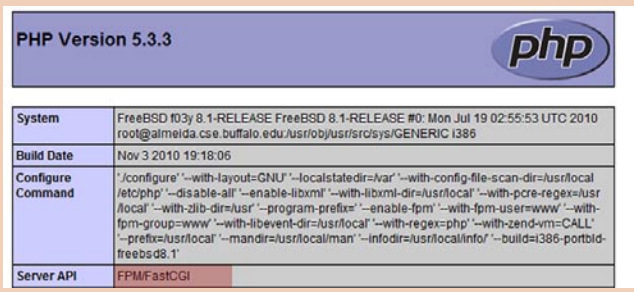
```
# cd /usr/local/etc/nginx/
# ee nginx.conf
```

Begin by editing your server information.

```
# redirect server error pages to the static page /50x.html
#
error_page   500 502 503 504  /50x.html;
location = /50x.html {
    root   /usr/local/www/nginx-dist;
}

# proxy the PHP scripts to Apache listening on 127.0.0.1:80
#
#location ~ \.php$ {
#    proxy_pass   http://127.0.0.1;
#}

# pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
#
location ~ \.php$ {
    #root           html;
    fastcgi_pass   127.0.0.1:9000;
    fastcgi_index  index.php;
    #fastcgi_param  SCRIPT_FILENAME  /scripts$fastcgi_script_name;
    fastcgi_param   SCRIPT_FILENAME  /usr/local/www/your-dir$fastcgi_script_name;
    include        fastcgi_params;
}

# deny access to .htaccess files, if Apache's document root
```

Next uncomment the section in brown and modify the directory settings to match your web directory. (Figure 3)

04



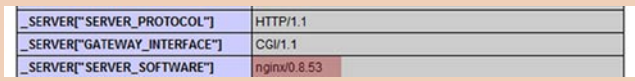
Testing Configuration

```
# cd /usr/local/www/your-dir
ee info.php
```

Type the following:

```
<?php
phpinfo();
?>
Save
```

FPM/FastCGI is your Server API



NGINX 0.8.53 successfully installed as web server. (Figure 5)

On the 'Net

- *http://www.php.net/*
- *http://wiki.nginx.org/*
- *http://www.apache.org/*

DIEGO MONTALVO

Diego Montalvo is a web/ mobile application developer which has developed web services such as buildasearch.com, bobchatter.com, urloid.com, pressoid.com and many others. Diego is currently leaving out of a suitcase exploring Mexico. Feel free to contact Diego at diego@earthoid.com

Text Terminal magic with tmux

Once you get used to something you seldom like to go back to old ways. So much so that you get uncomfortable without it.

What you will learn…

- You will figure out how to improve your productivity by a big factor in the most useful part of UNIX viz, the command line. Your creativity will expand and so will your efficiency and effectiveness of using UNIX.

What you should know…

- Reasonable experience with UNIX command line and shell interaction. No need for shell scripting knowledge. Basic command over the command line and few tools that are commonly used in the command line would suffice.

Tmux is one such tool

Anybody working in the UNIX world for a long time would have heard of GNU Screen, a form of terminal multiplexing which is a way by which you can use one console for multiple parallel sessions. This can be thought of as a text mode window manager which allows you to create infinite shells/terminals.

Screen has a multi view mode in which multiple people can collaborate and do a sort of conferencing in which what one types is seen by others.

This can be priceless for remote debugging sessions.

GNU Screen also has an uncanny ability to double up as a *nohup* for all the commands you run. So when there is a network outage or a machine crash, as long the UNIX server is up and running your command goes to completion.

This is in stark contrast to other situations that can be quite painful.

The other convenience is that of *attaching* and *detaching*. You can fire up a set of commands and detach and logout. You can login and attach to see what is going on.

Several time taking operations like download/uploads(Bittorrent) and logfile monitoring can be done this way.

But GNU Screen has been around for a long time and tmux is a new comer. Tmux is a massive improvement on screen having evolved and learnt from past mistakes as is the case with most software programs in the UNIX world.

Tmux is a pleasant piece of software that is welcome in the BSD world since its license is the liberal and meaningful BSD license. This allows it to be part of OSes like OpenBSD without requiring the user to install a separate package as is the case with GNU Screen.

This is very important since now you can rely on any OpenBSD system having tmux by default and you can write your programs to use it without fear.

Before we get to the nitty gritty of tmux let us take a look at a picture.

What you see here is a picture of tmux in action.You can find multiple panes here(4) and also two windows.

You can switch between them of course and type commands and the screens refresh automatically if you are monitoring something like network bandwidth using ifstat or bwm-ng or disk performance with iostat or system monitoring with systat.

You can also have fancy clocks and so on.

It takes a little while to get the hang of tmux particularly when you have not heard of Screen. Tmux has got a huge list of commands but we will focus on the most useful ones for the moment.

Every tmux command is prefixed with [*Ctrl-b*]. This can be changed but let us say that you as a beginner might

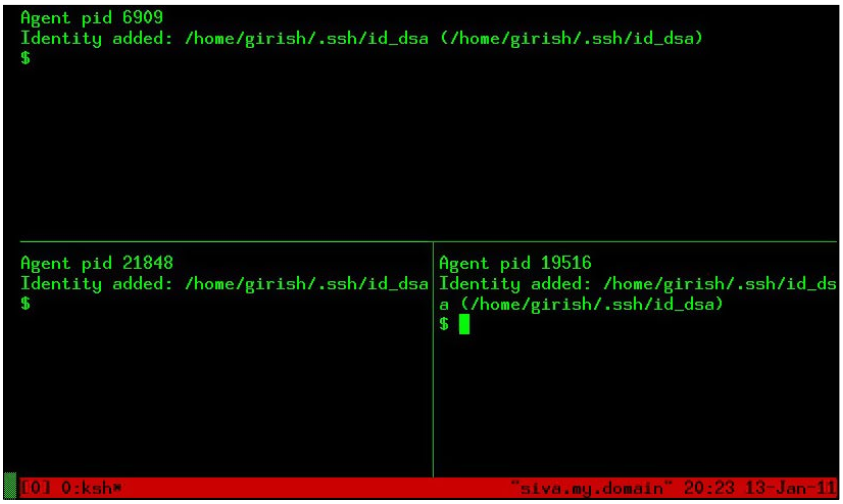


Figure 1. *tmux-pane*

prefer not to change this. For instance you can create a new window by using [*Ctrl-b+C*]. It is [*Ctrl-B*] and then [*c*]. [*c*] stands for create.

You can create a few windows and switch back and forth using [*Ctrl-B+n*] or [*Ctrl-B+p*]. [*n*] stands for next and [*p*] for previous.

I normally just stick to a certain pattern. It cycles so you don't have to worry. Eventually you will get the window you want.

Tmux also has a status display line which can be set with the command in `~/.tmux.conf`

```
set-option -g status-bg red
```

You can use *yellow*, *green* or *blue* if you want in place of red.

This line shows what command is running on each window.

This can be immeasurably useful since most of the time a particular utility/shell command invokes several other commands behind the scenes and we can see all that with tmux.

You can split a window into panes by using two commands for two ways of splitting a window into panes.

[*Ctrl-b+„*] for splitting horizontally and [*Ctrl-b+%*] for splitting windows into vertical panes.

You can cycle between the panes in a Window by using [*Ctrl-B+o*].

In fact you choose a window by using [*Ctrl-B+n*], then if a window has many panes you cycle between them using the above mentioned command.

Windows themselves are quite useful and fun; now panes are even better.

You can bind a key obviously entered after the prefix key sequence [*Ctrl-B*] and tmux will execute it for you everytime you press the hotkey sequence. You can also store it in a config file.

For instance, if you want to fire up a video everytime you press a key sequence in tmux, you only have to do this.

```
Ctrl-B+:
```

Enter this in the *colon* prompt a la vi.

```
: bind-key 4 run-shell „mplayer ~/videos/
        foo.mpg”
```

Now all you have to do to invoke this command is press

```
Ctrl-B + 4
```

By storing this in `~/.tmux.conf` you are saving yourself a lot of trouble.

Also by putting tmux in `~/.profile`.

Tmux has got a paste buffer which is something I took a long time to discern.

The paste buffer is accessed using the [*Ctrl-B+[*] key combo.

Then you simply press Page UP and Page Down keys. Once you are done just type [*q*].

You can detach from a tmux session by typing

```
Ctrl-b +d
```

You can then logout and go to some other part of the world and then attach to the same session by typing.

```
$ tmux attach
```

Other people who login as you can attach to a tmux you are running thereby enabling them to see what you are typing and vice versa.

Tmux is overall a very cool tool and the more you use it, the more you grow to love it.

Just the way UNIX has always been...

GIRISH VENKATACHALAM

Girish has close to 15 years of UNIX experience and he loves OpenBSD more than he loves anything else in the technology world.

Writing 'bots using

XMPP

One of my favorite topics, using XMPP/Jabber for productive, real world applications!

What you will learn…

- How to write a XMPP robot
- How to send an XMPP message from the UNIX command line

What you should know…

- How to install python packages
- How to establish an account on a XMPP server

XMPP isn't just for instant messaging. XMPP can be used for any messaging or presence application. Due to XMPP's well documented, standardized nature, it is relatively easy to create robots to handle almost any remote control task you might imagine.

Remotely controlled programs have earned the nickname 'bot. So, we'll be building some bots managed by XMPP. And we'll be building them using one of the 'net's current favorite scripting languages, python.

'bots can do almost anything their programmers desire them to do, like any program. The interesting things about 'bots is they can be controlled by someone elsewhere, frequently via email, or more recently, instant messaging.

Over the years, 'bots have been developed to handle tasks such as retrieving ftp-able files via email, managing mailing lists via email.

Web applications for managing mailing lists, and other activities could also be considered 'bots. The only limit is your imagination.

Getting Ready

We'll start by gathering together all the needed prerequisites.

Since we're working with Python, we'll need a Python interpreter, version 2.5 or newer.

Next up is xmpppy, which requires python's expat (part of the distribution tarball) and dnspython.

The final component is the python jabberbot module. All of the python modules can be installed from `pkgsrc{,-wip}` (wip/py-jabberbot, wip/py-xmpppy, net/py-dns, textproc/py-expat, lang/python2.6.) If you do install out of pkgsrc, installing wip/py-jabberbot will install all the required components.

Installing manually is pretty standard python install incantation:

```
<code>python setup.py install</code>
```

See the Listing 1 for URL's of all the needed pieces. (and a few others we'll also make use of..)

Lets build a 'bot!

Our first 'bot will be something very simple. A simple bot that does nothing more than send a greeting when a user says *hello* to it, and gives some basic system information. This is a very stripped down, partially reimplemented, version of the system status bot that is on the jabberbot home page.

One important, and perhaps less than obvious task, is to create an id on the jabber/xmpp server to use. Create the account as you would any jabber/xmpp account, with your favorite agent.

Using the jabberbot package to create 'bot's is pretty straight forward. You subclass the JabberBot class, and

Listing 1. *Source URLs*

```
jabberbot
    home page:  http://thp.io/2007/python-jabberbot/
    source:  http://thp.io/2007/python-jabberbot/jabberbot-0.10.tar.gz

eliza
    home page:  http://www.jezuk.co.uk/cgi-bin/view/software/eliza
    source:  http://www.jezuk.co.uk/files/eliza.py-0.2.tar.gz

xmpppy
    home page:  http://xmpppy.sourceforge.net/
    source: http://downloads.sourceforge.net/sourceforge/xmpppy/xmpppy-0.5.0rc1.tar.gz

dnspython
    home page:  http://www.dnspython.org/
    source:  http://www.dnspython.org/kits/1.8.0/dnspython-1.8.0.tar.gz
```

then implement the functions/keywords you want to be recognized, by marking the function definition with the prefix `@botcmd` on the preceding line. The function name itself is the command name.

The function signature must include three arguments. The first argument is the class object itself (*self*). The second argument is a XMPP message object from xmpppy (class xmpp.) The final argument is a list of words that followed the keyword in the message text.

This bot, sysinfo.py, has four commands, demonstrating how to return our results to the client, and using three of the JabberBot provided functions.

Dissecting the source, the first 4 lines are boiler plate to make sure we have the correct modules loaded to do everything we want to do.

Lines 33-34 define the jabber id and password to use. Line 35 creates the bot, and signs in. Line 36 starts the bot, and has it running until the end of time (or you send SIGINT, which ever comes first.)

The real work happens in lines 6 through 31, where we define the functions for the commands we are implementing. Lines 8-15 are the most complex, requesting information from the posix module about the system, turning it into a string to return to the client. As is evident, the string returned from the `@botcmd` function is the string returned to the XMPP client.

Lines 17-20 return the servers current date and time, in it's own time zone.

Lines 22-25 use JabberBot's `get_sender_username()` function to get the username portion of the Jabber ID (JID.) `get_sender_username` requires an argument of the message structure that each function receives.

Our final function definition in lines 27-31 is the moral equivalent of `whoami(1)`. It uses two functions from xmpppy, `getFrom()`, which returns the sender of the message from the xmpppy XMPP object (which has both the JID and the resource), and `getStripped()`,

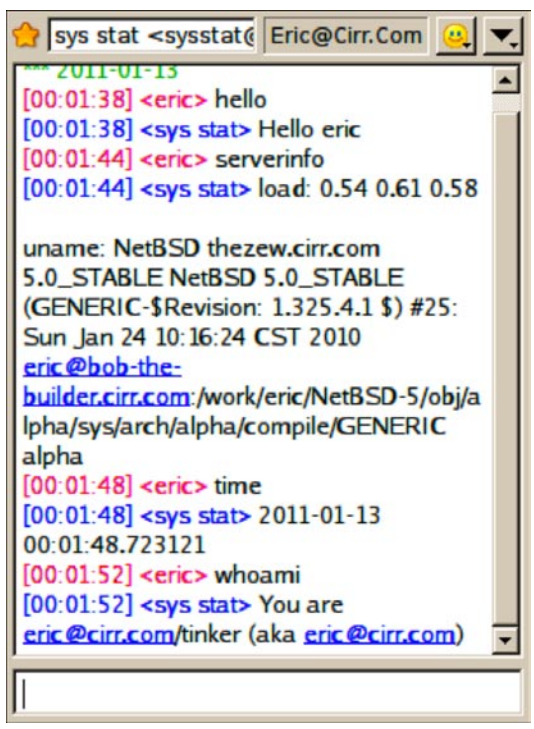


Figure 1. *Talking to sysstat@cirr.com*

which operates on an xmpppy JID object, returning the *bare* JID.

If you want to play with it immediately, feel free to contact *sysstat@cirr.com*. See Figure 1 for example output.

Listing 2. *sysinfo.py*

```
#! /usr/pkg/bin/python2.6
from jabberbot import JabberBot, botcmd
import datetime
import posix

class SystemInfoJabberBot(JabberBot):
    @botcmd
    def serverinfo( self, mess, args):
        """Displays information about the
            server"""
    ldavg = posix.getloadavg()
    uinfo = posix.uname()

    resp = 'load: %0.2f %0.2f %0.2f\n\n' % ldavg
    resp +=   'uname: %s %s %s %s %s' % uinfo
    return resp

    @botcmd
    def time( self, mess, args):
        """Displays current server time"""
        return str(datetime.datetime.now())

    @botcmd
    def hello( self, mess, args):
    """ Says hello to you"""
    return 'Hello %s' % self.get_sender_
            username(mess)

    @botcmd
    def whoami( self, mess, args):
        """Tells you your username"""
        return 'You are %s (aka %s)' % \
          (mess.getFrom(), mess.getFrom().getStr
            ipped())

username = 'sysstat@cirr.com'
password = '****************'
bot = SystemInfoJabberBot(username,password)
bot.serve_forever()
```

A second 'bot is an implementation of the classic *Eliza* quasi-AI program. For those unfamiliar with *Eliza*, it accepts questions and statements from the user, and responds in a fashion someone like a psychologist.

Listing 3. *doctor.py*

```
#! /usr/pkg/bin/python2.6
from jabberbot import JabberBot, botcmd

import posix
import eliza

class Doctor(JabberBot):

    def __init__(self, username, password,
            res=None, debug=False):
    self.__doctor = eliza.eliza()
    # call the parent class initializer
    JabberBot.__init__(self, username, password,
            res, debug)

    def unknown_command(self, mess, cmd, args):
    # handle unknown (all) commands
    return self.__doctor.respond( '%s %s' % (
            cmd, str(args)))

    @botcmd
    def about(self, mess, args):
    """ tells a bit about eliza """
    reply = 'Dr Eliza is an XMPP robot
            implementation based around '
    reply += 'an algorithm originally implemented
            by by Joseph '
    reply += 'Weizenbaum which simulates
            a therapist.'
    return reply

    @botcmd
    def hello( self, mess, args):
    """ trys a different way to get the JID """
    return 'Greetings %s' % mess.getFrom().getStr
            ipped()

username = 'doctor@cirr.com'
password = '***************'
bot = Doctor(username, password)

bot.serve_forever()
```

To do so, we'll need to install the eliza.py module. It uses the standard setup.py mechanism. Do that now.

Since we *Eliza* need to respond to arbitrary messages from the user, we need to overload/replace it's unknown command handler. By doing so, any text we don't only otherwise recognize/handle is passed along to the eliza text processor/handler. Let's take a look a the source in Listing 3.

For the Eliza bot, the `__init__ function` of JabberBot had to be overloaded to permit the initialization of the eliza module. Once the eliza module has been initialized, it needs to initialize the rest of the JabberBot module by calling the super-class initializer.

The other major difference from our other 'bot is the addition of the `unknown_command` handler. `unknown_command()` converts the command line into a single string, which is passed to the eliza module's response generation function. And eliza's response is then handed back to the person connected.

Feel free to talk to the doctor as *doctor@cirr.com*. Figure 2 contains a sample session with Dr Eliza.

Going the other way

XMPP can also be used to have applications send status messages and perhaps ask someone what the application should do.

Perhaps the simplest way for a system to send status messages is using the *xsend.py* script included in the xmpppy distribution. Using *xsend.py*, shell scripts can easily send a message to an administrator or multi-user conference room for easy monitoring/management.

xsend.py requires a JID/password to use as a sender (which is configured via a `~/.xsend` file), and then can send a message from the command line. The first argument is the JID to send the message to, and the rest of the command line is the message to send. By default, it sends all messages as type *message* (as opposed to type *chat*, which is used for interactive messaging.) Use it like this:

```
xsend.py root@example.com 'Example.org is not responding!'
```

Wrapping up

There are a multitude of ways to use XMPP beyond the obvious use of Instant Messaging. Using 'bots with XMPP is a wonderful way to step up automated communications between programs, systems and people.

Other uses of an XMPP 'bot that I've considered are RSS feed readers (which I will be implementing after I finish writing this article), dynamic DNS updater (imagine being able to send the DNS updates via XMPP, perhaps with a somewhat specialized client, instead of using something like DynDNS), and goodness knows what else. Please, share any ideas you may have with the author, and the community at large!

Look for an rss feed reader (any feed) at *xmpp: rss@cirr.com* in the near future. I'm going to have fun figuring out how to make it work!

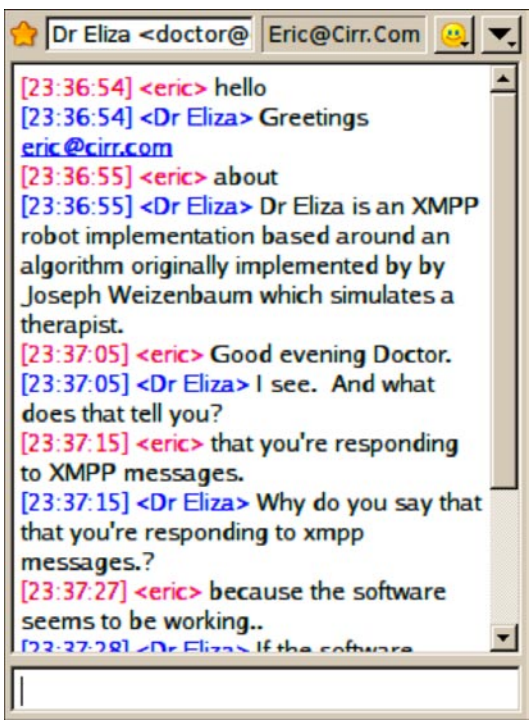


Figure 2. *A conversation with Dr Eliza*

ERIC SCHNOEBELEN

Eric Schnoebelen is a 25+ year veteran of the UNIX wars, using both System V and BSD derived systems. He's spent more than 20 years working with and contributing to various open source projects, such as NetBSD, sendmail, tcsh, and jabberd2. He operates a UNIX consultancy, and a small, NetBSD powered ISP. His preferred OS is NetBSD, which he has running on Alpha, UltraSPARC, SPARC, amd64 and i386.

He can be reached via xmpp as eric@cirr.com.

How to quickly make a

bootable USB stick with FreeBSD

This article covers the steps needed to make a bootable USB stick with FreeBSD – a quick howto that also applies to a USB drive.

What you will learn…

- How to make a bootable USB stick/disk with FreeBSD

What you should know…

- Booting with MFS gives more freedom

Install FreeBSD, or use an existing FreeBSD installation, and follow these steps:

1) First, you need to prepare and format your USB stick:

```
fdisk -BI /dev/da0
bsdlabel -B -w da0s1
newfs -U -O1 /dev/da0s1a
boot0cfg -v -B da0
```

(`-U -O1` [o like in Olympus, not zero] is for UFS1 which provides much faster copying than UFS2; if you decide for UFS2, type `-U -O2` – but expect that the copying will be slower)

2) Then mount it: mount `/dev/da0s1a /usb`

3) Copy all directories (FreeBSD) to the stick

4) After copying, modify the `/usb/boot/loader.conf` (explained below)

5) In the `/boot` directory on your USB stick you must have MFS (Memory File System – `mfsroot.gz`), which you will make (instructions are below)

6) Modify your `/etc/fstab` in MFS and put the following line (only) there: `/dev/md0 / ufs rw 0 0`

7) After you boot your computer with the stick, you will be in the MFS environment from which you will mount your USB stick with `mount_nullfs` (described below)

Modification of /boot/loader.conf on your USB stick

You must have the following lines in your `/boot/loader.conf` (some lines are optional):

```
mfsroot_load="YES"
mfsroot_type="mfs_root"
mfsroot_name="/boot/mfsroot"
nullfs_load="YES"
splash_bmp_load="YES"
vesa_load="YES"
geom_uzip_load="YES"
geom_label_load="YES"
bitmap_load="YES"
bitmap_name="/boot/splash.bmp"
snd_driver_load="YES"
kern.maxfiles="25000"
kern.maxusers="64"
vfs.root.mountfrom="/dev/md0"
# Additional filesystem drivers

udf_load="YES"
linux_load="YES"
fuse_load="YES"
ntfs_load="YES"
ext2fs_load="YES"
reiserfs_load="YES"
```

Making your own MFS

FreeBSD, after the kernel boots, can use the root file system in memory (`mfsroot_load="YES"` command in `/boot/loader.conf` will do the trick). To build such a memory file system, type the command: `dd if = /dev/zero of = mfsroot bs = 1024k count = 42`. The mfsroot file of about 40 MB in size will be created. You need to format it, mount it and copy the most important files into it from your FreeBSD system (`/bin`, `/sbin`, `/etc`, `/root….`):

```
mdconfig -a -f mfsroot md0
newfs /dev/md0
mount /dev/md0 /mnt
```

Once copied, you must unmount it and gzip it: gzip mfsroot

Optionally, you can chroot it to see if everything works, then copy the mfsroot.gz to `/usb/boot` onto your USB flash drive (or disk). If you think it may be a problem to pick the most important files for your MFS (from your FreeBSD installation), search for mfsbsd in Google and either use its toolset or the MFS image alone (contained in the downloadable ISO of mfsbsd).

After booting from the USB stick (you will jump into MFS), you must mount the physical USB stick:

```
/sbin/mount -o ro /dev/da0s1a /usb
/sbin/mount_nullfs /usb/boot /boot
/sbin/mount_nullfs /usb/usr /usr
```

The above commands will help you use the big `/usr` directory on your USB stick instead of the `/usr` dir in MFS. `mount_nullfs /usb/boot /boot` is optional, as in your `MFS/boot` directory only the following files are needed for the little MFS to boot (`/boot/kernel` directory in MFS): `geom_label.ko`, `geom_uzip.ko`, `zlib.ko` a their debug symbols (`zlib.ko.symbols`, etc.). By mounting the `/usb/boot` dir via `mount_nullfs` into the `/boot` directory in your MFS you will be able to load kernel modules.

JURAJ SIPOS

Juraj lives in Slovakia, where he works in a library (in an educational institute). He has been writing and selling computer articles for over ten years. He wrote an xmodmap howto (www.faqs.org/docs/Linux-mini/Intkeyb.html) and in addition to computers he is also interested in spirituality, but not really the guru side of things, but more-so freedom and self-actualization. His website says more: www.freebsd.nfo.sk.

FreeBSD and simple char

device driver for real PCI-hardware

The FreeBSD operating system captivates the hearts and minds of it's fans so much, that finds it's way in very diversive industries such as hosting projects and backbone routers. It can run on small embedded devices, as well as on large, multi-core systems.

What you will learn…

- How to program kernel character device driver
- How to use PCI POST card to visualize system's events

What you should know…

- PCI-subsystem of x86-compatible PC
- How to manipulate with sysinstall, make and gcc
- Basic knowledge of how FreeBSD kernel works

And sometimes, can be used in rather unusual state – for instance, to monitor several statuses of PC that is equipped with PCI POST card. We'll show today how it can be achieved.

Into the groove

System engineers usually begin acquainted with new hardware from a bottom line, in other words from low-level, firmware level. It applies either to non x86-architecture, as well as to x86, particularly to IBM PC-compatible machines. The latter ones are known to have a frimware naming like PC BIOS. Climbing from a bottom line, higher to a top, system engineer functionally and logically transforms to a high-level language programmer. Where architectural processes for complex systems take places. Unfortunaly, modern trends in IT shows very clearly the following: young professionals knows very well how to program nice UI, but completely unsure what is hidden under such terms like low-level programming and x86 BIOS calls. It's a pity, because, either low-level procedures, as well as BIOS calls are used every time to start a box with FreeBSD. Let's make a short trip into bootstrapping process for conventional x86-based PC.

First, when power cord is plugged and a power button is pressed, starts CPU. Almost immediately, RAM chips are initialized. Control vector is transferred to the following RAM address: `0xFFFF0`, or `0xFFFF.FFF0`. Both addresses are

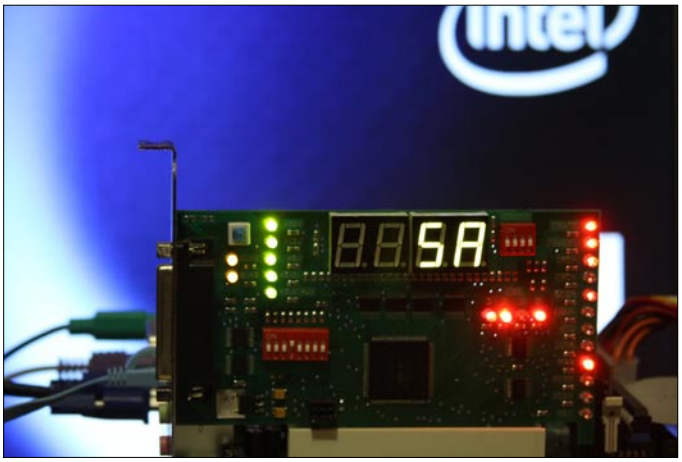


Figure 1. *IC80+ PCI POST card allows to visualize 0x80 / 0x81 ports contents*

```
Booting from Hard Disk...

F1  FreeBSD
F5  Drive 1

F6 PXE
Boot:  F1

BTX loader 1.00  BTX version is 1.02
Consoles: internal video/keyboard
BIOS drive C: is disk0
BIOS 636kB/523244kB available memory

FreeBSD/i386 bootstrap loader, Revision 1.1
(root@almeida.cse.buffalo.edu, Mon Jul 19 01:59:01 UTC 2010)
Loading /boot/defaults/loader.conf
```

Figure 2. *After BIOS initialization here comes BTX loader*

correct, because x86-compatible processor after first initialization must be in so-called *real-mode* state. At that time, memory address space from `0xE0000` to `0xFFFFF` is occupied by FlashROM. Where such procedures like initializations of various PC-components (RS232, LPT, USB) are stored.

In addition to the mentioned physical ports, each PC-system holds a virtual diagnostic port 0x80. PC BIOS (for example, AwardBIOS or PhoenixBIOS) sends into this port specifical values upon start of each init-procedure, so it can be just immediately been scanned by diagnostic equipment (we talk about it later). And finally, PC BIOS tunes then interrupt vectors (0x13, 0x18, 0x19) to have an ability to bootstrap an operating system – from a floppy disk, USB device, or ATA/SATA/SCSI device, in particular the interrupt vectors int

All these procedures are written in assembly-language and therefore it is assumed that resulting binary code is optimal in terms of execution time by CPU.

If all subsystems are functioning properly, PC BIOS will set up int 0x19 vector, so that BTX Loader could start. Which in turn would load whole FreeBSD system. All these things, we've discussed so far, are stored inside firmware (aka PC BIOS for x86 architecture). And if you woke up in the morning, and see your FreeBSD box doesn't want to boot – it could be a hardware error. Unidentified visually by PC BIOS, because it can be burned capacitor between ISA- and PCI-bus, or malfunctioning USB-port, which fails to initialize properly. Here comes diagnostic equipment, like POST (*Power-On Self Test*) card. It's mission is to show you on seven-segment LED display what's wrong. All you need is to have a list of POST-codes by your side, in order to figure out, what's exactly component failed.

IC80 POST-Card from IC Book Labs

I've searched thoroughfully marketplace to find out, what is difference exist between all these POST-cards. There is about a dozen different manufacturers – from eminent grandees, to noname cards, made somewhere in Guangdong province, PRC. Some developers provide a detailed desciption, manuals, and have a tech-support line, others only have look-alike printed circuit board. Since, I pefer to deal with a product, that offers a maximum number of features, I've chosen a POST-card from IC Book Labs [1] company. As it turned out, I made a right decision – more than 10 years of development, support for AMIBIOS, AwardBIOS, PhoenixBIOS, InsydeBIOS. Dual diagnostic LED-display that shows contents of virtual ports 0x80 / 0x81 / 0x84 / 0x1080 / 0x2080. There's also a switch, that allows to go through all POST step-by-step. A really swiss knife.

Though it's price differs from nonamed product by a factor of 10, it provides more features by the same factor. Anyway, let's get started.

Listing 1. *pciconf shows attached devices to system*

```
[root@a-bsd ~]# pciconf -l
hostb0@pci0:0:0:0:      class=0x060000 card=0x464c8086 chip=0x27708086 rev=0x02 hdr=0x00
vgapci0@pci0:0:2:0:     class=0x030000 card=0x464c8086 chip=0x27728086 rev=0x02 hdr=0x00
none0@pci0:0:27:0:      class=0x040300 card=0xd6048086 chip=0x27d88086 rev=0x01 hdr=0x00
pcib1@pci0:0:28:0:      class=0x060400 card=0x00000000 chip=0x27d08086 rev=0x01 hdr=0x01
pcib2@pci0:0:28:2:      class=0x060400 card=0x00000000 chip=0x27d48086 rev=0x01 hdr=0x01
pcib3@pci0:0:28:3:      class=0x060400 card=0x00000000 chip=0x27d68086 rev=0x01 hdr=0x01
uhci0@pci0:0:29:0:      class=0x0c0300 card=0x464c8086 chip=0x27c88086 rev=0x01 hdr=0x00
uhci1@pci0:0:29:1:      class=0x0c0300 card=0x464c8086 chip=0x27c98086 rev=0x01 hdr=0x00
uhci2@pci0:0:29:2:      class=0x0c0300 card=0x464c8086 chip=0x27ca8086 rev=0x01 hdr=0x00
uhci3@pci0:0:29:3:      class=0x0c0300 card=0x464c8086 chip=0x27cb8086 rev=0x01 hdr=0x00
ehci0@pci0:0:29:7:      class=0x0c0320 card=0x464c8086 chip=0x27cc8086 rev=0x01 hdr=0x00
pcib4@pci0:0:30:0:      class=0x060401 card=0x464c8086 chip=0x244e8086 rev=0xe1 hdr=0x01
isab0@pci0:0:31:0:      class=0x060100 card=0x464c8086 chip=0x27b88086 rev=0x01 hdr=0x00
atapci0@pci0:0:31:1:    class=0x01018a card=0x464c8086 chip=0x27df8086 rev=0x01 hdr=0x00
atapci1@pci0:0:31:2:    class=0x01018f card=0x464c8086 chip=0x27c08086 rev=0x01 hdr=0x00
ichsmb0@pci0:0:31:3:    class=0x0c0500 card=0x464c8086 chip=0x27da8086 rev=0x01 hdr=0x00
re0@pci0:1:0:0: class=0x020000 card=0x00018086 chip=0x816810ec rev=0x02 hdr=0x00
none1@pci0:4:0:0:       class=0x118000 card=0x00000000 chip=0x001cb00c rev=0x05 hdr=0x00
```

Boot sequence

I've inserted this IC80 POST card into free PCI-slot and pushed a power button. My mainboard was produced by Intel (D945GCLF2), so I referred to POST-codes that this manufacturer reserved for it's products [2]. Complete startup sequence for all initialization procedures was as follows:

```
22, 23, 25, 28, 34, 12, 58, 50, 51, EB, 58, 92, 90, 94,
                    95, BB, B8,
BA, 5A, 92, 90, 94, BB, BA, EB, BB, BA, 5A, BB, BA, E7,
                    E9, and finally 00
```

- 22 – Reading SPD from memory DIMMs
- 23 – Detecting presence of memory DIMMs
- 25 – Configuring memory
- 28 – Testing memory
- 34 – Loading recovery capsule
- 12 – Starting Application processor initialization
- 58 – Resetting USB bus
- 50 – Enumerating PCI busses
- 51 – Allocating resources to PCI bus
- EB – Calling Legacy Option ROMs
- 58 – Resetting USB bus
- 92 – Detecting presence of keyboard
- 90 – Resetting keyboard
- 94 – Clearing keyboard input buffer
- 95 – Instructing keyboard controller to run Self Test (PS2 only)
- BB – reserved by Intel
- B8 – Resetting removable media
- BA – Detecting presence of a removable media (IDE, CD-ROM detection, etc.)
- 5A – Resetting PATA/SATA bus and all devices
- ...
- E7- Waiting for user input
- E9 – Entering BIOS setup
- 5A – Resetting PATA/SATA bus and all devices
- BA – Detecting presence of a removable media (IDE, CD-ROM detection, etc.)
- 00 – Ready to boot

After last event (ID: 00) boot control is passed to BTX Loader and the usual bootstrapping for FreeBSD begins (see Figure 2).

FreeBSD system is up

After all necessary services enlisted in rc.conf are completed, we can log into FreeBSD and see, how the POST device looks like (in terms of the operating system of course. See Listing 1).

We see all integrated devices, including video- and network-card. And there are 2 strange devices for which there is no loaded driver: `none0@pci0:0:27:0` and `none1@pci0:`

Listing 2. *More advanced information for devices with 'unloaded' driver*

```
[root@a-bsd ~]# pciconf -lv | grep none -A3
none0@pci0:0:27:0:      class=0x040300 card=0xd6048086 chip=0x27d88086 rev=0x01 hdr=0x00
    vendor     = 'Intel Corporation'
    device     = 'IDT High Definition Audio Driver  (BA101897)'
    class      = multimedia
--
none1@pci0:4:0:0:       class=0x118000 card=0x00000000 chip=0x001cb00c rev=0x05 hdr=0x00
    vendor     = 'IC Book Labs'
    device     = 'IC80+PCI POST Diagnostics Card'
    class      = dasp
```

Listing 3. *FreeBSD contains no driver for IC80+PCI POST Diagnostics Card*

```
[root@a-bsd ~]# dmesg | grep pci4
pci4: <ACPI PCI bus> on pcib4
pci4: <dasp> at device 0.0 (no driver attached)
```

Listing 4. *Makefile for sample kernel driver 'hello_world'*

```
KMOD=   hello_world
SRCS=   hello_world.c

.include <bsd.kmod.mk>
```

Listing 5. *Source file for sample kernel driver 'hello_world'*

```
#include <sys/types.h>
#include <sys/param.h>
#include <sys/module.h>
#include <sys/sysproto.h>
#include <sys/sysent.h>
#include <sys/kernel.h>
#include <sys/systm.h>
/*
 * The function for implementing the syscall.
 */
static int
hello (struct thread *td, void *arg)
{
        printf ("hello kernel world\n");
        return 0;
}
/*
 * The 'sysent' for the new syscall
 */
static struct sysent hello_sysent = {
        0,                      /* sy_narg */
        hello                   /* sy_call */
};
/*
 * The offset in sysent where the syscall is allocated.
 */
static int offset = NO_SYSCALL;
/*
 * The function called at load/unload.
 */
static int
load (struct module *module, int cmd, void *arg)
{
        int error = 0;
        switch (cmd) {
        case MOD_LOAD :
                printf ("Driver loaded at %d\n", offset); /* logging to syslog */
                uprintf ("Driver loaded at %d\n", offset);  /* logging to terminal */
                break;
        case MOD_UNLOAD :
                printf ("Driver unloaded from %d\n", offset); /* logging to syslog */
                uprintf ("Driver unloaded from %d\n", offset); /* logging to terminal */
                break;
        default :
                error = EOPNOTSUPP;
        }
        return error;
}
SYSCALL_MODULE(hello_world, &offset, &hello_sysent, load, NULL);
```

Listing 6a. *Source file for kernel driver 'ic80'*

```
#include <sys/types.h>
#include <sys/module.h>
#include <sys/systm.h>  /* uprintf */
#include <sys/errno.h>
#include <sys/param.h>  /* defines used in kernel.h */
#include <sys/kernel.h> /* types used in module
                   initialization */
#include <sys/conf.h>   /* cdevsw struct */
#include <sys/uio.h>    /* uio struct */
#include <sys/malloc.h>
#include <sys/types.h>

#define BUFFERSIZE 5

/* Function prototypes */
static d_open_t      ic80_open;
static d_close_t     ic80_close;
static d_read_t      ic80_read;
static d_write_t     ic80_write;

/* Character device entry points */
static struct cdevsw ic80_cdevsw = {
    .d_version = D_VERSION,
    .d_flags = D_PSEUDO | D_NEEDGIANT,
    .d_open = ic80_open,
    .d_close = ic80_close,
    .d_read = ic80_read,
    .d_write = ic80_write,
    .d_name = "ic80",
};

typedef struct s_ic80 {
    char msg[BUFFERSIZE];
    int len;
} t_ic80;

static struct cdev *ic80_dev;
static int count;
static t_ic80 *ic80msg;

MALLOC_DECLARE(M_IC80BUFFER);
MALLOC_DEFINE(M_IC80BUFFER, "ic80buffer", "a buffer
                   for ic80 post card kernel module");

static int

{
    int err = 0;

    switch (what) {
    case MOD_LOAD:
        ic80_dev = make_dev(&ic80_cdevsw, 0, UID_ROOT,
                    GID_WHEEL, 0666, "ic80");
        ic80msg = malloc(sizeof(t_ic80), M_IC80BUFFER,
                    M_WAITOK);
        printf("Driver IC80 loaded\n");
        break;
    case MOD_UNLOAD:
        destroy_dev(ic80_dev);
        free(ic80msg, M_IC80BUFFER);
        printf("Driver IC80 unloaded\n");
        break;
    default:
        err = EOPNOTSUPP;
        break;
    }
    return(err);
}

static int
ic80_open(struct cdev *dev, int oflags, int devtype,
                    struct thread *p)
{
    int err = 0;
    printf("Opening device \"ic80\" ...\n");
    return(err);
}

static int
ic80_close(struct cdev *dev, int fflag, int devtype,
                    struct thread *p)
{
    printf("Closing device \"ic80\" ... \n");
    return(0);
}

/*
 * The read function just takes the buf that was saved
                    via
 * echo_write() and returns it to userland for
                    accessing.
 * uio(9)
 */
```

`4:0:0`. Let's start pciconf in more verbose mode: see Listing 2.

Excellent! The first device for which there is no driver has been loaded – an integrated Intel HD audio device. What about the second one? This is our POST-card. Make sure once again, that no single driver was loaded for it during bootstrap process (see Listing 3).

Actually, the functionality of this card is quite simple – to display on LED0 and LED1 display what has been sent to diagnostic ports 0x80 and 0x81. And I'm quite sure, we're able to write necessary software for it. Well, we're moving to the next section!

Designing simple char-device driver

As an example let's first analyze how to program a very simple kernel driver. It's mission is to write to syslog and terminal a message *Driver loaded*, once it's registered by system. And once the kernel module is unregistered by system, it should write *Driver unloaded*. This will be a truly *Hello, world!*, but with kernel background.

But first, please make sure that the following packages are installed on the system: gcc, make, kernel sources, and share sources. Usually, the first 2 packages are already present in system. You only need to install the latter two. Run sysinstall and install the following packages:

```
sysinstall->Configure->Distributions->src->share
sysinstall->Configure->Distributions->src->sys
```

The good starting point for writing a kernel driver from scratch is to look at: `/usr/src/share/examples/kld/syscall/module/syscall.c`.

And, it would be also worth to look at this page [3] – with explanation about module structure (see Listing 4 and Listing 5).

Start compilation process:

```
# make
```

And now we load module into kernel address space:

Listing 6b. *Source file for kernel driver 'ic80'*

```
static int
ic80_read(struct cdev *dev, struct uio *uio, int
                    ioflag)
{
    int err = 0;
    int amt;

    /*
     * How big is this read operation?  Either as big
                    as the user wants,
     * or as big as the remaining data

    amt = MIN(uio->uio_resid, (ic80msg->len - uio-
                    >uio_offset > 0) ?
         ic80msg->len - uio->uio_offset : 0);
    if ((err = uiomove(ic80msg->msg + uio->uio_offset,
                    amt, uio)) != 0) {
        uprintf("uiomove failed!\n");
    }
    return(err);
}

static int
ic80_write(struct cdev *dev, struct uio *uio, int
                    ioflag)
{
    int err = 0;

    /* Copy the string in from user memory to kernel
                    memory */
    err = copyin(uio->uio_iov->iov_base, ic80msg->msg,
                    4);

    /* Now we need to null terminate, then record the
                    length */
    *(ic80msg->msg + 4) = 0;
    ic80msg->len = 4;

    if (err != 0) { uprintf("Write failed: bad
                    address!\n"); }
    count++;

    outb( 0x80, strtol(ic80msg->msg, 0, 16) );

    return(err);
}

DEV_MODULE(ic80,ic80_loader,NULL);
```

```
# kldload ./hello_world.ko
```

Is it really there?

```
# dmesg | grep Driver
hello_world loaded at 210
```

Okay, the driver functions correctly. You can unload it from memory like that:

```
# kldunload hello_world
```

We got now a very simple driver. Our next step is to add more functionality, i.e. a driver must be able to create character device under `/dev/` tree, and it also must be able to visualize any value of hexadecimal type on the LED0-display of the POST-card.

The latter procedure is done by sending hex value to diagnostic port 0x80. Yes, that's easy (see Listing 6).

One short comment about this source. We create a structure named `ic80_cdevsw`, where we place the functions, that will be called upon access a character device `/dev/ic_80`. For example, upon every close, open, write or read operation.

```
static struct cdevsw ic80_cdevsw = {
 .d_version = D_VERSION,
 .d_flags = D_PSEUDO | D_NEEDGIANT,
 .d_open = ic80_open,
 .d_close = ic80_close,
 .d_read = ic80_read,
 .d_write = ic80_write,
 .d_name = „ic80”,
};
```

Inside `ic80_loader()` function we program the mechanism, how the module should be registered, and unregistered by operating system. Within `ic80_write()` function the value for diagnostic port 0x80 is passed from user-space land to kernel and send directly to port 0x80. Internal buffer `ic80msg` shouldn't be very long, because it should store 5 characters only.

Listing 7. *System load value is pushed to new char device*

```
#!/bin/sh
        while true;
   do
      sleep 1s;
      id='top -d 1 | grep load | awk '{ print $6 }' |
                  head -c 4 | tail -c 2';
      echo $id > /dev/ic80;
   done
```

Driver is ready. Now, when you load it into the kernel address space it will register a new character device `/dev/ic80`, with permissions 0666 and owned by root: wheel. After that you can send any value in `0xXX` format (for example, 0xAB) into `/dev/ic80` and it will be displayed on LED0-screen. I've decided to control system average load, and prepared the following script. Once it is started by user, and once your CPU is heavy loaded, the immediate change of LED0-indicator is guaranteed. My tough and heavy CPU loader job was performed with help of `burnMMX` utility (from `/usr/ports/sysutils/burn` package).

Actually, you can display on LED0-indicator whatever information you need. Values should be in range between 0x00 and 0xFF. For instance, it could be a temperature, either of CPU, or mainboard. Moreover, don't forget about LED1-indicator – you can enhance the kernel driver in order to show twice as more information.

On the 'Net

- *http://en.icbook.com.ua/ [1]*
- *http://www.intel.com/support/motherboards/desktop/sb/CS-025434.htm [2]*
- *http://www.redantigua.com/c-ex-kernel-freebsd-hello.html [3]*

Conclusion

As you can see, designing a character driver is not as difficult as it seems. It can be fun sometimes, especially if you have very unusual hardware and FreeBSD by your side. And you also have a desire to combine them both into one amazing solution. Anyway, I'm sure that every novice FreeBSD user has a potential to create something possible from impossible pieces. Just like any FreeBSD guru, isn't it?

ANTON BORISOV

The very first Anton's experience with UNIX was FreeBSD. It was TWM, wget and Netscape Communicator. Many things have changed greatly since then, but a true simplicity remained unchanged – The Power to Serve. That's why Anton chooses FreeBSD for 'impossible' missions.

FreeBSD Mall

Your FreeBSD & PC-BSD Resource

www.FreeBSDMall.com

FreeBSD 8.1 Jewel Case CD/DVD

Set contains:

- **Disc 1:** Installation Boot (i386)
- **Disc 2:** LiveFS (i386)
- **Disc 3:** Essential Packages (i386)
- **Disc 4:** Essential Packages (i386)

FreeBSD 8.1 CD $39.95
FreeBSD 8.1 DVD $39.95
FreeBSD 7.3 CDROM $39.95
FreeBSD 7.3 DVD $39.95

FreeBSD Subscriptions

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

FreeBSD Subscription , start with CD 8.1 ..... $29.95
FreeBSD Subscription, start with DVD 8.1 ..... $29.95
FreeBSD Subscription, CD 7.3 ..... $29.95
FreeBSD Subscription, DVD 7.3 ..... $29.95

PC-BSD 8 DVD (Hubble Edition)

PC-BSD 8 DVD $29.95
PC-BSD Subscription $19.95

BSD Magazine

BSD Magazine $11.99
BSD Magazine Subscription $11.99

The FreeBSD Handbook

The FreeBSD Handbook, Volume 1 (User Guide) $39.95
The FreeBSD Handbook, Volume 2 (Admin Guide) $39.95
☆ Special: The FreeBSD Handbook, Volume 2 (Both Volumes) $59.95
☆ Special: The FreeBSD Handbook, Both Volumes, & FreeBSD 8.1 ..... $79.95

The FreeBSD Bundle

Inside the Bundle, you'll find:

- FreeBSD Handbook, 3rd Edition, Users Guide
- FreeBSD Handbook, 3rd Edition, Admin Guide
- FreeBSD 8.1 4-disc set
- FreeBSD Toolkit DVD

☆ Special: The FreeBSD CD Bundle $89.95
☆ Special: The FreeBSD DVD Bundle $89.95

The FreeBSD Toolkit DVD $39.95

FreeBSD Mousepad $10.00

FreeBSD Caps $20.00

PC-BSD Caps $20.00

For *MORE* FreeBSD & PC-BSD items, visit our website at FreeBSDMall.com!

CALL 925.240.6652 Ask about our software bundles!

t-shirts
$18-$21.99

BSD's and Solaris on the Desktop

Are they ready to serve?

As I am a great unix fan, I use BSD daily, but I mainly use the beast on the servers. In my company, we run Linux on Desktops and I would like to change that too. Therefore I underwent this venture in order to see whether Unix is ready to replace Linux on the desktop or not.

What you will learn…
- How do they compare?
- How extensive hardware support they offer.
- Which software you can use.

You will not learn...
- How to install the systems or software.

For understanding this article...
- You can be a beginner Linux user, or an advanced BSD admin, each will learn his part.

The tested systems were FreeBSD 8.1, NetBSD 5.1, OpenBSD 4.8 and 4.7, OpenIndiana (fork of OpenSolaris). They were all latest stable versions, except for OpenIndiana, which was Solaris build 148 only in pre-release stage.

Some people might be interested why I omitted PC-BSD or Desktop-BSD in the testing, if they are actually Desktop-ready solutions.

The reason is, I do not want to test various customised distributions, I wanted to compare and test the systems with their default tools.

I also have to admit, I find stuff like PC-BSD very monstrous and sluggish, absolutely unfit for an office solution, where the desktop has to be clean, simple and fast, if I should use KDE for office use, then only the simple and fast KDE3.

I was also not interested in the ease of install, since I believe, offices have IT-people for that, office workers usually do not install systems, printer drivers and such.

Desktop Features

What does desktop use mean?

For many users it can be many things, but I was focused on what we can describe as a small office use. The office I run has three full time office workers who work most of the day on the computer, one is an accountant, who runs MS Windows in a virtualised window, as we use a version of accounting that does not have a version for Linux and if run in Wine, it frequently freezes. We occasionally have to take some photos from the mobile phone and send it over Internet. We have Ethernet network, but one of our colleagues works part of the week from her home and I go for meetings to various places, so we often use either wireless or ppp over phone – here bluetooth comes handy. We often scan contracts or other documents. Multimedia make our time nicer, so I teseted them too. I believe, many other users do similar work with their computers at home or at their offices. So that leads me to conclusion, people might be interested, how things actually work.

Hardware

Tested hardware was various brands, so I believe this will not be seen as promoting some hardware

- Dell Optiplex 755 desktop, Acer TravelMate 8471 laptop, Fujitsu-Siemens S6120D laptop
- HP LaserJet 4, Samsung ML1510 Laser printer, Canon Pixma MP190 (multifunctional), HP F-2480 Deskjet (multifunctional)
- intel video and audio integrated cards
- network Ethernet 10/100 PCI cards of various make, Intel cards on laptops
- wireless network

- bluetooth phone connection
- USB thumb 128M, 2G, 8G with vfat FS
- SonyEricsson mobile phone C510

Software

- Desktop environment with auto-mounting
- Performance
- OpenOffice
- PDF reader
- web browser with flash plug-in
- Virtualization (for applications which need other, often closed-source and evil, operating system environment)
- Multimedia (just for curiosity)

Hardware match

PRINTING

All the systems tested were able to print to the old HP laser printer, all the BSDs were able to print to the Samsung laser using the Splix package. There was no native Splix package for OpenIndiana, however there is a simple tutorial how to compile the package from sources on OpenSolaris forums. It was different with the two multifunctional machines.

OpenBSD didn't print on Canon Pixma, although it had the driver. It was very strange, since with a help of an OpenBSD developer I applied a patch and made it work in OpenBSD 4.7. Both the scanners started to move, however in the middle of scanning the process froze with an IO exception. Making sane-backends finding HP F Deskjet and Pixma needed adjusting kernel.

OpenIndiana printed well to both the multifunctional printers, but it did not manage scanning with any of the scanners at all. OpenIndiana is also missing the HPLIP package that would give gear to Sane to manage the HP multifunctional Scanner.

FreeBSD was a clear winner here. It printed to everything and at the same time it scanned with both the scanners.

And not only did it scan, unlike the other BSDs it even scanned and printed at the same time – without rebooting into the ulpt0-disabled recompiled kernel. Good work gentlemen!

NetBSD has a very outdated HPLIP package, so it failed with HP-F-series printer, however with Pixma it printed well after recompiling gutenprint using *with-cups* option. NetBSD found both the scanners after recompiling kernel, but failed to work with them completely.

NETWORKING, WIRELESS, BLUETOOTH

Bluetooth worked very well with NetBSD, OpenBSD and FreeBSD, and connected through ppp0 and SonyEricsson mobile to the Internet.

All possible tested Ethernet cards worked on all systems – which is a huge improvement for OpenIndiana concerning the fact that Solaris was very weak at this point just two years ago.

Intel wireless worked reliably only on OpenBSD after downloading the binary from a designated page.

VIDEO and AUDIO

I have to say I had no major problems with either sound or video drivers on any of the PCs with any of the BSDs. Sometimes some tweaking was neccessary, but mainly adjusting mixerctl, nothing like recompiling drivers or similar. OpenIndiana on the Siemens lifebook laptop

Figure 1. *FreeBSD was scanning easily with the tested printer-scanner-copier devices*

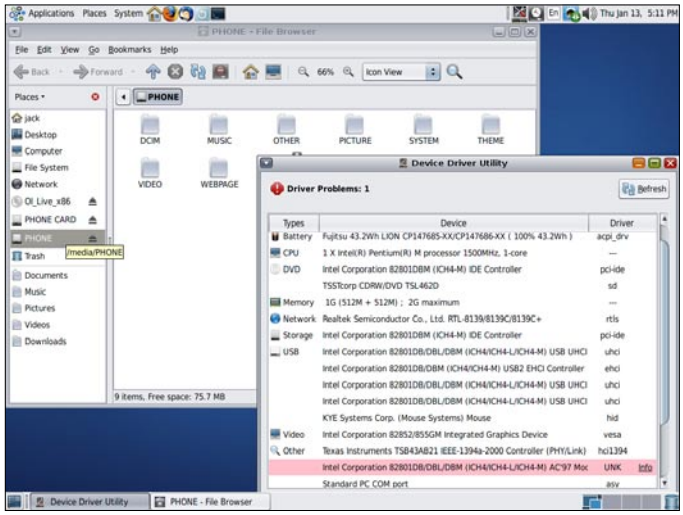


Figure 2. *OpenIndiana's interesting drivers overview tool. Mounted phone in Nautilus on the left*

was not able to open X until I chose VESA as the video driver.

USB

Surprisingly, FreeBSD had serious issues in mounting USB devices, in fact, I only succeeded with en old 120M flash stick. NetBSD wanted first disklabel run on the devices, but then it was able to mount everything. The USB flash and the Phone card worked best with OpenBSD and (surprisingly) with OpenIndiana, everything perfectly.

Software match

Desktop environment with auto-mounting

All the BSDs come with pretty uncustomised Gnome 2.6-3.2 and KDE 3.5.10 and also Xfce 4.6 environments, I have tested them and they work nicely. Although all the BSDs offer KDE 4.5 at the time of writing, I already explained why I did not test KDE 4 for the office. The OpenIndiana, OpenSolaris legacy, comes with a customised set of icons and nicely tweaked desktop style, OpenIndiana is the only one that uses Hal to mount devices by default.

In FreeBSD and NetBSD you can use Hal if you like, and if you like, you can use the auto-mounting that Gnome and KDE offer by default, it requires setting a user-mount option in kernel and defining the mount points in fstab, then adjusting properties. OpenBSD only allows the latter way, no Hal in OpenBSD.

Performance

There was an interesting difference in performance. OpenIndiana was robust (ate about 800MB memory), but all packages were stable, I haven't noticed any crash.

It ran also quite fast, after the boot, which was never-ending. OpenBSD was incredibly stable, besides running apps under linux emulation, I noticed no crash, but it was visibly slower than the other systems. Interestingly enough, it consumed least memory.

Just under 300 MB while NetBSD and FreeBSD needed slightly more, about 320-30 MB (may be because of the Hal?), but these two were the fastest.

FreeBSD happened to crash an application two times during the testing, NetBSD was, however, and I am sorry to say that, very unstable in terms of desktop applications.

OpenOffice

OpenOffice worked for me in all the tested systems, only while OpenBSD and OpenIndiana had ready a package, in FreeBSD and NetBSD I had to compile it in ports/pkgsrc and it took me a while (and about 10GB space), however that is still fine, since you can compile that once and then use the package for all other computers at the office.

There were also a number of other Office applications ready in packages an ports/pkgsrc, so if you are a friend of KDE office suite or Gnumeric+Abiword, these packages are available.

PDF reader

Evince present, kpdf present, Xpdf present, Acrobat reader under Linux emulation present for the BSDs, while Evince + native Acrobat ready for OpenIndiana. So, absolutely no problem.

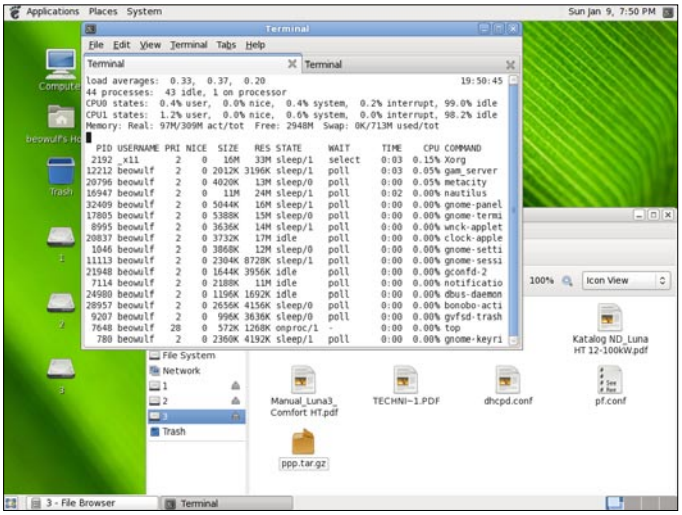


Figure 3. *OpenBSD was able to mount simply everything, while consuming the least resources*

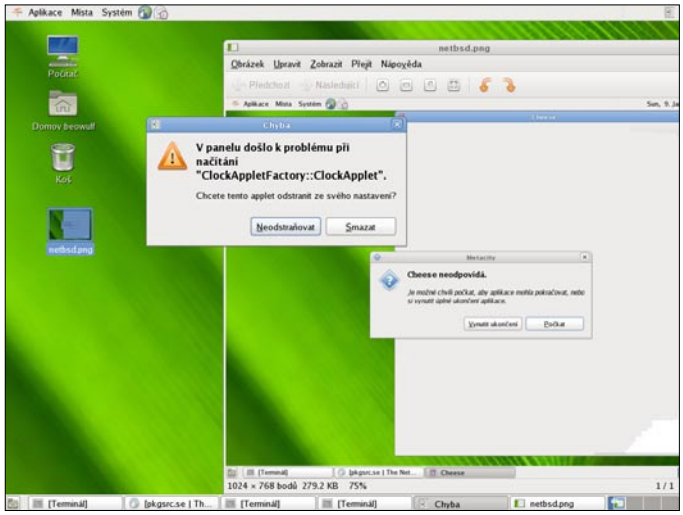


Figure 4. *NetBSD announced another failure while I was viewing the screenshot of a previous one*

Web browser with a Flash plug-in

That was a difference here. All the systems offer Firefox, Seamonkey, and other browsers so far fine, however with Flash plug-in, that was a different story. The OpenIndiana needed a plug-in to download and place into mozilla/plugins folder, then it worked natively. NetBSD and FreeBSD emulate Linux environment and Flash 10 works fine with nspluginwrapper. OpenBSD used to have a very well working Flash 7 plug-in in Opera, however now Opera freezes every 15 seconds and Flash 7 is in various periods of the year found nowhere, so practically it is unusable.

Theoretically you could use the free alternatives, gnash or swfdec plug-in, however neither of them worked reliably when I was testing them.

There are tools for downloading a Flash video and then playing it from a command line, but I do not consider this equal to the working browser plug-in.

Virtualization

(for applications which need other, often closed-source and evil, operating system environment)

OpenIndiana offers Solaris zones and VirtualBox. I tested VirtualBox and it worked flawlessly.

FreeBSD offers qemu+kqemu, VirtualBox under emulation and to some extent Xen, however, that is reported to have some bugs yet. NetBSD has well working Xen, qemu, and now also VirtualBox. I tested the firs two and worked well.

OpenBSD has qemu+kqemu but qemu only worked well when without qemu. It was also painfully slow.

For DOS apps – all BSDs were able to emulate DOS with Dosbox and run our old company system reliably.

Multimedia

It is not important for the office, but just for general curiosity- all the BSDs were able to play every imaginable DVD or video and audio format I tried,

Tabe 1. *Overview comparison*

	FreeBSD 8.1	NetBSD 5.1	OpenBSD 4.8	OpenIndiana build 147	Linux 10/2010
Hardware					
Desktop laptop basic drivers	All worked	All worked	All worked	All worked	All worked
Audio+Video	All worked	All worked	All worked	With issues	All worked
Network drivers	All worked	All worked	All worked	All worked	All worked
wireless	Not working	With issues	All worked	Not working	All worked
bluetooth	All worked	All worked	All worked	Not working	All worked
USB thumb	Not working	With issues	All worked	All worked	All worked
Phone card	Not working	With issues	All worked	All worked	All worked
Laser printers	All worked	All worked	All worked	With issues	All worked
Printer-scanner	All worked	Printing only	Printing only	Printing only	All worked
Software					
Desktop env.	Gnome, KDE, KDE 4, Xfce	Gnome, KDE, KDE 4, Xfce	Gnome, KDE, KDE 4, Xfce	Gnome	Gnome, KDE, KDE 4, Xfce
Performance	Fast, stable	Fast, unstable	Slow, stable	Fast, stable	Fast, stable
Ram	315	340	295	780	850
Office suite	All worked	All worked	All worked	All worked	All worked
PDF	All worked	All worked	All worked	All worked	All worked
Web browser	Firefox, Seamonkey, Opera, Chrome, Konqueror, Galeon	Firefox, Seamonkey, Opera, Chrome, Konqueror, Galeon	Firefox, Seamonkey, Opera (fails), Konqueror, Galeon	Firefox	Firefox, Seamonkey, Opera, Chrome, Konqueror, Galeon .....
Flash plug-in	All worked	All worked	Not working	All worked	All worked
Virtualization	Jail, Kqemu, Virtualbox, Xen?, dosbox	Qemu, Virtualbox, Xen, dosbox	Kqemu, dosbox	Virtualbox, Zones	KVM, Kqemu, Virtualbox, Xen, dosbox, dosemu
Multimedia	All worked	All worked	All worked	With issues	All worked

OpenIndiana had issues with some codecs, but after talking to the developers on the irc, I found that this is being worked at currently. Talking about such a fancy stuff as multimedia, people may be interested in Skype – it works on FreeBSD. It is claimed to work under zones in OpenIndiana, it used to work in times of Skype 1.x on NetBSD (currently it does not), and it is not supported by OpenBSD on purpose. It is not a secret that skype works as a spyware, and following a guide published on the *http://forum.skype.com/* I could attest it myself, we can see that the OpenBSD people know what they are doing, and at our office there is no place for such an application.

Conclusion

I would like to remind everyone, that the following conclusion is not about comparing the BSDs with Solaris as systems, it is about comparing their USE AS A DESKTOP IN A SMALL OFFICE.

NetBSD's pkgsrc collection is also huge, browsers play flash, documents of all kinds are opened in a while, virtualization offer is rich, however the problem with scanners, fact that its version of HPLIP is 8 years behind and mostly the desktop apps stability issues make it fail the desktop office task.

OpenIndiana is a new player here, since Solaris started to develop for i386 platform just a few years ago, but it did surprisingly well, even if with relatively small repository at hand, it opened all possible files, played flash through the browser, and was very stable, handling USB disks correctly, but a lot of development is unfinished yet, so access to hardware is limited with the small number of packages there are, therefore many printers, scanners, which are crucial for the office work will need more development. We can see it is only in the beginning of its way. By the time of writing this article, build 148 was made available so we can expect some improvement already now.

OpenBSD has smaller collection of packages, but contains the most important ones which are up to date and perfectly stable, and what more, is able to operate even on machines with very little memory. Has the best functioning drivers for network and USB. It is just one step from being able to use the scanners in multifunctional machines. So, concerning hardware it is very close to the goal. However its inability to provide a browser with a flash plug-in and its slugishness, and a slow emulator, make it loose much of its attractiveness.

FreeBSD offers the largest collection of software ranging from fancy multimedia and network to developer tools. The system that worked fast and with only a few failures and it was the only machine that cooperated flawlessly with our Printer-Scanner-Copier machines. The only downside was, however, the lack of support for USB thumb, phone card and wireless drivers. This result is probably the closest to what we want. So can we say it is the winner?

While I would very much like to say so, I can't. The winner is clearly Linux at this time, since it supports all the hardware mentioned and offers the most extensive software support. I cannot replace our Linux work stations with unix yet. However I have watched the development in the last five years and the development is really fast so while the OpenIndiana baby is doing its first steps on the right trail, the BSD monsters (and especially the FreeBSD) may very soon (if not already next year) beat Linux in this field. Already now, the BSDs are much better at memory management with weaker machines.

Finally a little remark, whenever I communicated with the developers of all four systems, and I really gave them some time, they were really helpful, open and communicative. Hats off!

PETR TOPIARZ

The author owns and runs a small language school, lives in a small town near Prague in Czech Republic, Europe. He started with Linux on desktop back in 2005 and with BSDs on servers in 2006, and currently administers four small company networks. The author simply likes small things.

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/*

Games Geeks Play!

In this article, we explore the various gaming options available for the BSD users.

A common misconception is that gaming and open source – the two words can never seem to gel together. BSD users are desktop publishers, programmers and a lot many other things, but most emphatically hardly gamers. The myth remains that the scene in the open source diaspora is bleak when it comes to gaming. Well, its high-time we broke the myth. This so-called bleak scene has been showing some signs of improvement of late. So it seems only logical that we check out the progress of the BSD world in the gaming sector, following which, we shall also check out some Open Source gaming platforms and consoles to contest the likes of MS Xbox, Sony Play Station and Nintendo Wii.

Games For The Freedom Lovers

The options are galore when it comes to Open Source games, and there are some very good ones that can give the proprietary guys a run for their money. Following are some of the lesser known yet must-try gaming options for BSD users:

Urban Terror

Urban Terror is a first person shooter which is based on Quake III. In Urban Terror, you are exposed to the regular hack and slash gaming along with a few jumps and power slides. The gameplay is fun and once you are accustomed to the controls, it can keep you hooked for hours at length.

The one unique thing about Urban Terror is the fact that the game aims to be realistic. You will not find characters with magic potions or childish supernatural fantasies. On being shot, your character bleeds and if medical aid (in Multiplayer mode) is not administered in time, it may prove fatal. Cool, eh? (see Figure 1).

Urban Terror is available for Linux, Windows and (don't blink, its true) Mac. Well, that's the official compatibility list (which, as the site itself confesses, isn't updated often). I play it under Dragonfly, and it works smooth enough. To get the game, visit the site at *www.urbanterror.info/news/home*.

Scorched 3D

Scorched 3D is a remake of the DOS classic Scorched Earth. It is a strategy game meant for Windows, Linux,

Figure 1. *Urban Terror Gameplay*

BSD, Mac and other UNIX-based machines. It features online multiplayer mode, superb artillery, a plethora of characters and scenes. Sounds good? Get the game at *www.scorched3d.co.uk*.

Age of Conquest

Age of Conquest is a strategy game in which you take turns setting up your empire, conquering others' empires and so on. It supports online multiplayer modes with over 39,800 registered users, and also runs on Android and Apple's iOS. The catch? Well, shell out $19.99 to play the full version. The game is available at *www.ageofconquest.com*.

Savage 2

Savage 2 is a First Person Shooter where you are in control of several supernatural characters and the goal is to kill the bad ones. Simple! The graphics are mind blowing and you'll keep coming back to it again and again. The only drawback seems to be a bug that causes the launch to be sluggish under NetBSD (Figure 2).

Recently, Savage 2 was sub-divided into two versions: The Battle for Newerth and A Tortured Soul. To download the game, visit the website at *www.savage2.com* Be warned though, A Tortured Soul is not 100% Open Source yet.

UFO Alien Invasion

This game is a remake of the DOS game UFO: Enemy Unknown. It is a strategy game where you have to establish bases to train your troops and develop modern weaponry. The storyline is good, and the combat against aliens is well laid out and quite detailed. The gameplay is complex and a beginner is sure to be left dazed at the sheer variety of game modes. The latest version is 2.3 and it can be downloaded from *www.ufoai.sourceforge.net*, see Figure 3.

Want more? Try *Battle for Wesnoth*, *Lips of Suna*, *Project X* and *Forgotten Elements*. Who needs Halo!

Open Source Gaming Platforms/consoles

XGameStation

XGameStation is a gaming console cum platform developed as an open source project. It truly lives up to the spirit of Open Source in the sense that it is the world's only console that is available in a build-it-yourself kit. Yes, you read it right! You can customize and tweak the console as much as you like. So this essentially means freedom from proprietary mumbo-jumbo such as that available in MS Xbox.

Figure 2. *A Scene from Savage 2: The Battle for Newerth*

XGameStation (XGS) comes in two major variants: Micro and Pico. The Micro version is unique because it provides out-of-the-box support for Atari-compatible joysticks and controllers. The Pico version, on the other hand, is a stripped down model of Micro XGS with lesser RAM and Flash capacity. Both of them, however, are build-it-yourself kits. With the dawn of Sony Play Station and Nintendo Wii, XGS has suffered obvious setbacks, yet it is available in most major Asian markets on a severely limited basis.

Pandora

Now this is what I call true gaming, the FOSS way! Pandora claims to be the most powerful gaming console ever! And with a 600 MHz+ CPU (that runs UNIX, by the way), 800x480 16.7 million colors touchscreen LCD and OpenGL compatible 3D Hardware, it surely has features to brag about.

Figure 3. *UFO Alien Invasion: Detailed Gameplay*

Pandora, the handheld gaming console, is developed by the OpenPandora Project (the name clearly tells you about it being Open Source). It has high-end PDA capabilities, and it comes with Angstrom Linux pre-installed. Wait, the best is yet to come! Don't like the default Angstrom Linux? Fret not! Just install your favorite BSD distro! At present, Pandora supports most of the major names, including NetBSD, OpenBSD, FreeBSD and Dragonfly. What's more! Pandora can help you access the net using your favorite browser (Firefox and Chromium come pre-installed). The battery life is quoted at 10 hours.

You won't find Pandora in the nearby gadget store. You'll have to pre-order at their site *www.openpandora.org* The cost is approximately 349$ though with taxes and all it is likely to reach 380$.

Delta 3D

Delta 3D is an Open Source gaming simulator. It provides all that one can ask for when it comes to game development engines – particle editor, world editor, mode creator, compiler and model viewer. It is released under GNU LGPL and is freely redistributable. The fully customizable source code can be had from *www.delta3d.org*.

The Bottom Line

Recently, Electronic Arts, the gaming stalwart known for productions such as Need For Speed and FIFA, expressed the need for a true Open Source gaming world. That sums it up! FOSS seems to be the most plausible solution to the ever-intense war between consoles and games, and when it comes to FOSS, the torch-bearer surely is BSD. With the advent of freedom in the gaming world, the end-gamers are sure to expect a treat!

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

Hakin9

PRACTICAL PROTECTION

IT SECURITY MAGAZINE

Why can't office employees

get along with open source office suites?

I have been working for 6 years now in an office setting. Since the organization I work for does not have that "big" funds for purchasing bleeding-edge software, we put our hands on some open source counterparts of the proprietary ones.

Since I am the IT guy, I am the one installing and deciding which open source software will be installed. It seems that there is no problem for my younger officemates as they tend to adjust and work their way around quickly. But for the others, I received a lot of complaints.

The office suites we use were from Microsoft Office (97, XP, 2003), OpenOffice.org (2.x, 3.x), IBM Symphony (1.x) and Google Docs.

Those using *OpenOffice.org* and IBM Symphony have been quite comfortable since the interface is almost the same as Microsoft Office 2003. The only problem I received from those using these software were some formatting issues. These were minor formatting issues which can be resolved by a couple of minutes editing the document and major formatting issues when the user edit a file in the new Microsoft document format (*docx*, *xlxs*, *pptx*), which takes longer to fix.

Now here comes the hard part. How to convince the not so young employees to use open source office suites instead of proprietary ones. Now let's see what they say about the open source office suites: *Since we use proprietary office suite, we will be having hard time studying that thing*, *We have tried that software, but it just doesn't work*, *The feature I'm using in the proprietary software is not included in that software, so how can I work?*, and *The learning curve, the interface, and everything does not fit my taste*.

As you can see, the reasons they give were all not acceptable. First of all, the new versions of the proprietary office suite is very different from their versions 7 years ago. This means that if an employee will use the bleeding-edge software, he/she will have a harder time in figuring out the placement of menu and icons. If the open source office suite doesn't work, then why would a lot of developers, institutions (private/government), and schools were shifting from proprietary to open source software? It just means that open source works well!

Well not all feature of the proprietary software is included in the open source counterpart. But take a look at this, would you pay big bucks for features you don't use? Not all employees need the power and feature of proprietary office suite. The management may work this one out by identifying the key employees needing a special feature *ONLY* available in the proprietary office suite. But I assure you, most of the functions and features an employee will need is available and is provided by open source office suite.

I think that office employees don't want open source software since they are afraid to try new things. They are afraid that they would exert more time in their work using open source software. But we all know that open source software really works. What we can do is demonstrate them the power and functionality of open source software. We should show them that it really does the job, just like their proprietary counterparts. We all have our things to do to be able to convince them using the open source software. Not only do we help the open souce community, but we also help our own company in terms of savings by using *Free Software*.

JOSHUA EBARVIA

Joshua Ebarvia is a java programmer, systems administrator and college lecturer. His passion is working and using operating systems specially UNIX-based and UNIX-cloned systems. You can reach him at joshua.ebarvia@gmail.com

In the next issue:

- Practical Security Auditing with FreeBSD
- Introduction to openssl: Command Line
- Mutt On OS X
- and Other !

Next issue is coming in February!

Support

Because sometimes, you just need some.

The iXsystems Professional Services Team is comprised of veteran FreeBSD® developers, administrators, and project committers.

Our expert staff provides unparalleled levels of support for FreeBSD®, PC-BSD® and FreeNAS™ ranging from problem resolution to consultation to custom development. For more information on our Professional Services Offering, visit our website at http://www.iXsystems.com/BSDsupport and complete the inquiry form.

iXsystems®

Call iXsystems toll free or visit our website today!

+1-800-820-BSDi | www.iXsystems.com