

LINUX JOURNAL™

Since 1994: The Original Magazine of the Linux Community

TIME-SAVING
COMMAND-LINE TRICKS

INSTALL THE
CYANOGENMOD ANDROID
DISTRO ON YOUR PHONE

TRY SUPER-LIGHTWEIGHT
TINY CORE LINUX

EMBEDDED LINUX

INTERVIEW WITH HEXAPOD
DEVELOPER MATT BUNTING



DEBUG
EMBEDDED
LINUX
PLATFORMS
WITH
PYTHON
AND GDB

APPLYING
SPEECH
I/O IN
EMBEDDED
APPLICATIONS

Getting Started
with **Buildroot**

Authenticate
Users with **OAuth**

JUNE 2011 | ISSUE 206
www.linuxjournal.com



More TFLOPS, Fewer WATTS

Microway delivers the fastest and greenest floating point throughput in history

Enhanced GPU Computing with Tesla Fermi

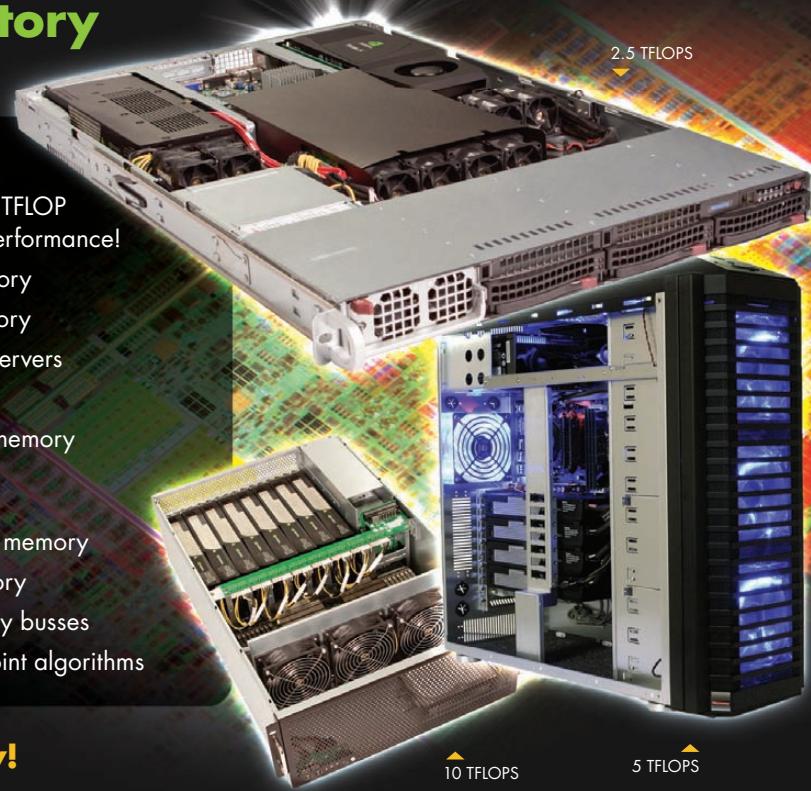
- ▶ 480 Core NVIDIA® Tesla™ Fermi GPUs deliver 1.2 TFLOP single precision & 600 GFLOP double precision performance!
- ▶ New Tesla C2050 adds 3GB ECC protected memory
- ▶ New Tesla C2070 adds 6GB ECC protected memory
- ▶ Tesla Pre-Configured Clusters with S2070 4 GPU servers
- ▶ WhisperStation - PSC with up to 4 Fermi GPUs
- ▶ OctoPuter™ with up to 8 Fermi GPUs and 144GB memory

New Processors

- ▶ 12 Core AMD Opterons with quad channel DDR3 memory
- ▶ 8 Core Intel Xeons with quad channel DDR3 memory
- ▶ Superior bandwidth with faster, wider CPU memory busses
- ▶ Increased efficiency for memory-bound floating point algorithms

Configure your next Cluster today!

www.microway.com/quickquote
508-746-7341

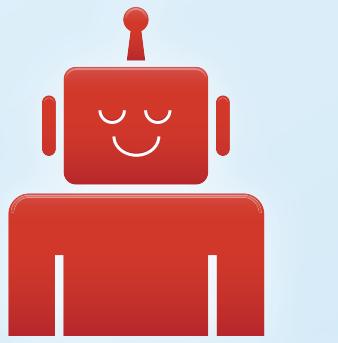


FasTree™ QDR InfiniBand Switches and HCAs

- ▶ 36 Port, 40 Gb/s, Low Cost Fabrics
- ▶ Compact, Scalable, Modular Architecture
- ▶ Ideal for Building Expandable Clusters and Fabrics
- ▶ MPI Link-Checker™ and InfiniScope™ Network Diagnostics

Achieve the Optimal Fabric Design for your Specific MPI Application with ProSim™ Fabric Simulator

Now you can observe the real time communication coherency of your algorithms. Use this information to evaluate whether your codes have the potential to suffer from congestion. Feeding observed data into our IB fabric queuing-theory simulator lets you examine latency and bi-sectional bandwidth tradeoffs in fabric topologies.



Lullabot™

Learn Drupal & jQuery FROM THE COMFORT OF YOUR LIVING ROOM

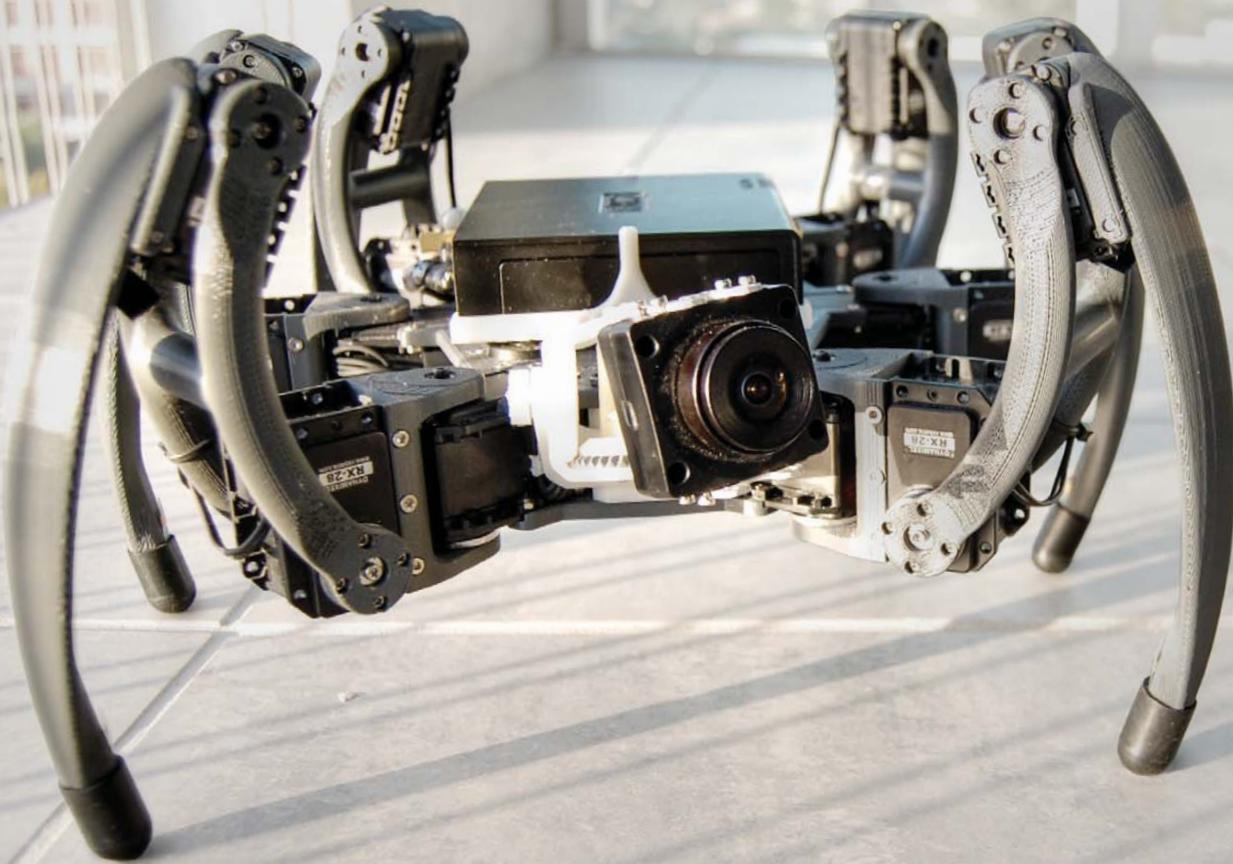


The Lullabot Learning Series includes everything you need to become a Drupal & jQuery expert from the comfort of your living room! The videos are available in both DVD format and high-definition video download.

Purchase the videos at <http://store.lullabot.com>

CONTENTS

JUNE 2011 Issue 206



FEATURES

40
**Hexapod—a
Linux-Powered
Spider Robot**

Interview with Matt Bunting, the hexapod robot developer.

Anton Borisov

46
**Debugging
Embedded
Linux Platforms
with GDB
and Python**

GDB provides support for scripting debugging actions using a Python interpreter.

Tom Parkin

52
**Breaking
Free the
Gumstix DSP**

Setting the Gumstix Overo Fire ablaze with the DSP.

James McColl

58
**Speech I/O
for Embedded
Applications**

Speech I/O works! See how to apply it in your next embedded application project.

Rick Rogers

ON THE COVER

- Time-Saving Command-Line Tricks, p. 32
- Install the CyanogenMod Android Distro on Your Cell Phone, p. 64
- Try Super-Lightweight Tiny Core Linux, p. 67
- Interview with Hexapod Developer Matt Bunting, p. 40
- Debug Embedded Linux Platforms with Python and GDB, p. 46
- Applying Speech I/O in Embedded Applications, p. 58
- Getting Started with Buildroot, p. 72
- Authenticate Users with OAuth, p. 76

10 Gig On Board

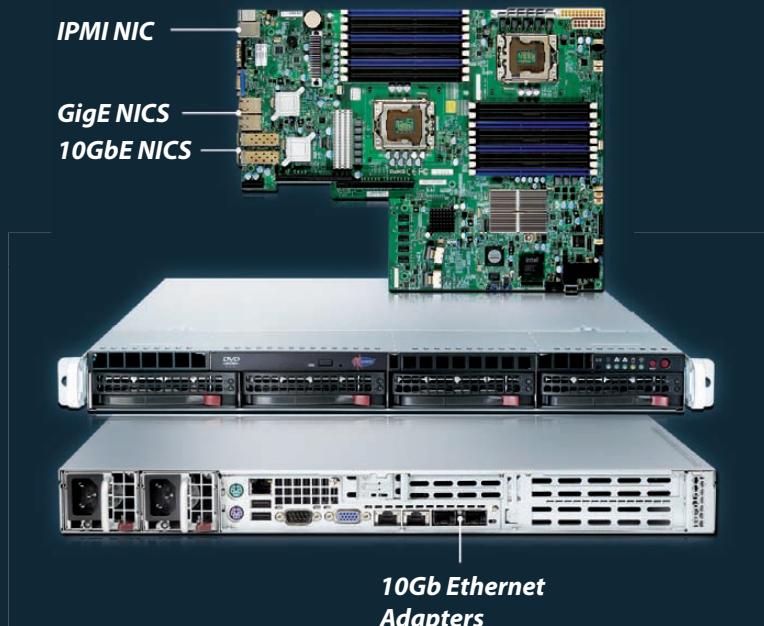
Blazing Fast, Embedded 10Gb Ethernet

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

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



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



KEY FEATURES:

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



Call iXsystems toll free or visit our website today!

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

Intel, the Intel logo, Xeon, and Xeon Inside are trademarks or registered trademarks of Intel Corporation in the U.S. and/or other countries.



**Powerful.
Intelligent.**

CONTENTS

JUNE 2011
Issue 206

COLUMNS

18 Reuven M. Lerner's At the Forge

Backbone.js

24 Dave Taylor's Work the Shell

More Fun with Days and Dates

26 Mick Bauer's Paranoid Penguin

DNS Cache Poisoning, Part II: DNSSEC Validation

32 Kyle Rankin's Hack and /

Lightning Hacks—the Command Next Door

80 Doc Searls' EOF

Whatever Sinks Your Boat

INDEPTH

64 CyanogenMod 7.0—Gingerbread in the House

What happens when you mix powerful, Linux-powered cell phones with an active Open Source community?

Shawn Powers

67 Tiny Core Linux

An intro to this very small, run-in-memory distro.

Joey Bernard

72 Roll Your Own Embedded Linux System with Buildroot

Embedded Linux, the easy way.

Alexander Sirokin

76 A Primer to the OAuth Protocol

OAuth is a simple way to authenticate users.

Adrian Hannah

IN EVERY ISSUE

8 Current_Issue.tar.gz

10 Letters

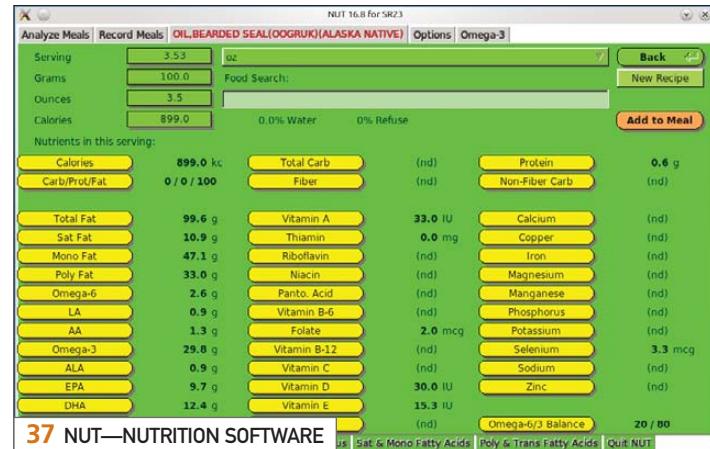
14 UPFRONT

34 New Products

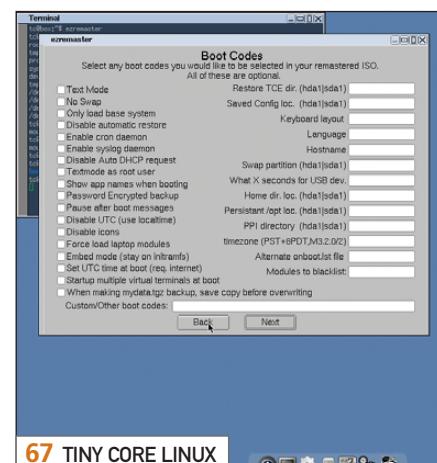
36 New Projects

65 Advertisers Index

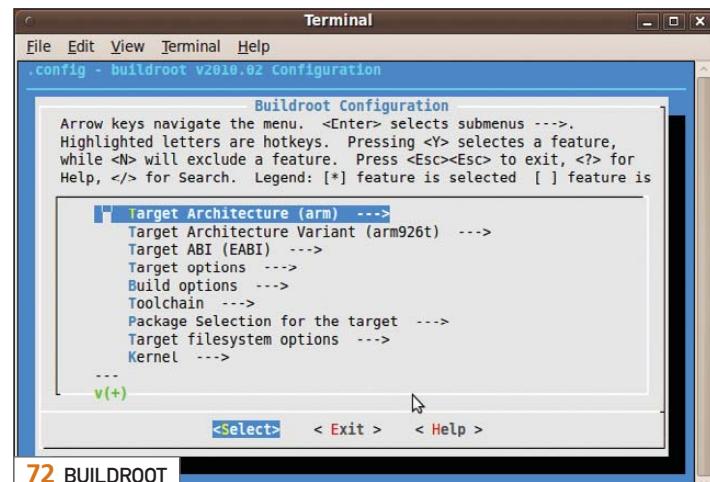
79 Marketplace



64 CYANOGENMOD 7.0



67 TINY CORE LINUX



72 BUILDROOT

USPS LINUX JOURNAL (ISSN 1075-3583) (USPS 12854) is published monthly by Belltown Media, Inc., 2121 Sage Road, Ste. 310, Houston, TX 77056 USA. Periodicals postage paid at Houston, Texas and at additional mailing offices. Cover price is \$5.99 US. Subscription rate is \$29.50/year in the United States, \$39.50 in Canada and Mexico, \$69.50 elsewhere. POSTMASTER: Please send address changes to *Linux Journal*, PO Box 16476, North Hollywood, CA 91615. Subscriptions start with the next issue. Canada Post: Publications Mail Agreement #41549519. Canada Returns to be sent to Pitney Bowes, P.O. Box 25542, London, ON N6C 6B2

The Business of Cloud Computing

June 13-15, 2011 / Hyatt Regency Mission Bay Spa and Marina San Diego, CA

Everyone is talking about cloud computing, but your business needs more than talk and buzzwords—you need to understand how to harness this enormous catalyst for change.

In this forum you'll hear from executives who have captured the savings, flexibility, security and scalability to enable their businesses to grow and innovate. You can't afford to miss the lessons learned from such luminaries as:

James Williams, Chief Information Officer, NASA Ames Research Center

Also hear from Fortune 500 company executives, start-ups, law firms, banking and educational institutions. In addition, learn about associations helping to set standards and mitigate risk in the cloud such as the Cloud Security Alliance and Open Grid Forum.

SPONSORSHIP AND EXHIBITING OPPORTUNITIES

If you are interested in sponsoring, speaking or exhibiting at this event, please call 212-532-9898 x 0 or email info@opalevents.org

REGISTER

To register, visit us online at www.opalevents.org/print/LJ or email us at marketing@opalevents.org

ref code: BCCA1105



LINUX JOURNAL™

Since 1994: The Original Magazine of the Linux Community

DIGITAL EDITION NOW AVAILABLE!

Read it first

Get the latest issue before it
hits the newsstand

Keyword searchable

Find a topic or name
in seconds

Paperless archives

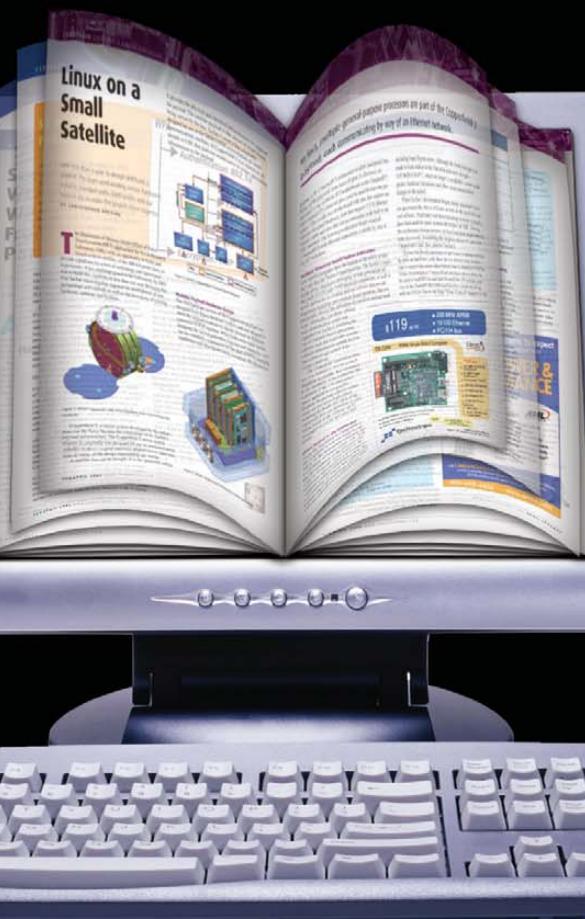
Download to your computer for
convenient offline reading

Same great magazine

Read each issue in
high-quality PDF

Try a Sample Issue!

www.linuxjournal.com/DLISSUE



LINUX JOURNAL

Executive Editor	Jill Franklin jill@linuxjournal.com
Senior Editor	Doc Searls doc@linuxjournal.com
Associate Editor	Shawn Powers shawn@linuxjournal.com
Art Director	Garrick Antikajian garrick@linuxjournal.com
Products Editor	James Gray newproducts@linuxjournal.com
Editor Emeritus	Don Marti dmarti@linuxjournal.com
Technical Editor	Michael Baxter mab@cruzio.com
Senior Columnist	Reuven Lerner reuven@lerner.co.il
Security Editor	Mick Bauer mick@visi.com
Hack Editor	Kyle Rankin lj@greenfly.net
Virtual Editor	Bill Childers bill.childers@linuxjournal.com

Contributing Editors

Ibrahim Haddad • Robert Love • Zack Brown • Dave Phillips • Marco Fioretti • Ludovic Marcotte
Paul Barry • Paul McKenney • Dave Taylor • Dirk Elmendorf • Justin Ryan

Proofreader	Geri Gale
--------------------	-----------

Publisher	Carlie Fairchild publisher@linuxjournal.com
------------------	--

General Manager	Rebecca Cassity rebecca@linuxjournal.com
------------------------	---

Senior Sales Manager	Joseph Krack joseph@linuxjournal.com
-----------------------------	--

Associate Publisher	Mark Irgang mark@linuxjournal.com
----------------------------	---

Webmistress	Katherine Druckman webmistress@linuxjournal.com
--------------------	--

Accountant	Candy Beauchamp acct@linuxjournal.com
-------------------	---

Linux Journal is published by, and is a registered trade name of, Belltown Media, Inc.
PO Box 980985, Houston, TX 77098 USA

Editorial Advisory Panel

Brad Abram Baillio • Nick Baronian • Hari Boukis • Steve Case
Kalyana Krishna Chadalavada • Brian Conner • Caleb S. Cullen • Keir Davis
Michael Eager • Nick Faloutsos • Dennis Franklin Frey • Alicia Gibb
Victor Gregorio • Philip Jacob • Jay Kruizinga • David A. Lane
Steve Marquez • Dave McAllister • Carson McDonald • Craig Oda
Jeffrey D. Parent • Charnell Pugsley • Thomas Quinlan • Mike Roberts
Kristin Shoemaker • Chris D. Stark • Patrick Swartz • James Walker

Advertising

E-MAIL: ads@linuxjournal.com
URL: www.linuxjournal.com/advertising
PHONE: +1 713-344-1956 ext. 2

Subscriptions

E-MAIL: subs@linuxjournal.com
URL: www.linuxjournal.com/subscribe
PHONE: +1 818-487-2089
FAX: +1 818-487-4550
TOLL-FREE: 1-888-66-LINUX

MAIL: PO Box 16476, North Hollywood, CA 91615-9911 USA
Please allow 4-6 weeks for processing address changes and orders
PRINTED IN USA

LINUX is a registered trademark of Linus Torvalds.



The newly updated **LINUX JOURNAL ARCHIVE** is here!



The archive includes **all 200 issues of *Linux Journal***, from the premiere issue in March 1994 through December 2010. In easy-to-use HTML format, the fully searchable, space-saving archive offers immediate access to an essential resource for the Linux enthusiast: *Linux Journal*.



SHAWN POWERS

The Bottle Labeled "Drink Me"

Let's face it, the Linux install base is shrinking. No, of course I don't mean numbers, I mean the actual size of the devices onto which Linux is installed. Just like with Alice's trip down the rabbit hole, we're seeing our favorite OS embedded on smaller and smaller hardware. This month, we talk about some of those places and teach you how to make a "Drink Me" bottle for your own projects.

Reuven M. Lerner starts us out with Backbone.js. Making Web applications, whether big or small, is an invaluable skill. Every Web device supports JavaScript (for the most part), and Backbone.js helps make those Web apps seem like traditional desktop applications. Mick Bauer provides Part II of his series on DNS cache poisoning, which can affect all users. Whether you are using a Droid in your pocket, Linux on your lap or a server in your workroom, DNS is how you get things done on-line. Everyone is vulnerable, so be sure to read up!

If installing Linux on something as mundane as a phone isn't your cup of tea, you'll likely be interested in Anton Borisov's article on the Linux-powered spider robot, hexapod. A device right out of a science-fiction movie, and also my nightmares, the spider bot is powered by Linux. Anton interviews its creator, Matt Bunting, and explains how it works. Tom Parkin talks about bugs this month too, although his article is a little less creepy. Tom shows how to de-bug embedded Linux platforms with GDB and Python. If you're a Linux developer, chances are you're familiar with GDB. Tom demonstrates version 7, which now has Python support.

When it comes to embedded Linux projects, they don't get much smaller than with the Gumstix. James McColl walks us through compiling a custom kernel for the Gumstix Overo Fire. If you want to install Linux on a device you could disguise as a stick of chewing gum, or if you're just interested in learning to compile custom embedded kernels, be sure to check it out.

What could be scarier than the robotic spider, hexapod? Well, perhaps if that same spider bot were able to speak to us. Rick Rogers explores speech recognition and synthesis for embedded systems. Although the technology certainly isn't

limited to autonomous spider robots, I fear our readers might try to do just that. If you do, please don't send me one for review.

We've got other Linux distributions designed for embedded systems this month, one of which is CyanogenMod 7.0. I had the opportunity to interview Steve Kondik from the CyanogenMod team, and I show off some of the new features of this cutting-edge Android ROM. Tiny installs of Linux certainly aren't a new idea, and Joey Bernard shows us a tiny distribution designed for computers. Even cell-phone developers would have a hard time beating the space saved by Tiny Core Linux. At 10MB, it has a full graphical environment and can run completely in RAM.

Perhaps the idea of a pre-made distribution leaves a sour taste in your mouth. That's fine too. Alexander Sirokin shows how to roll your own embedded Linux system with Buildroot. This is useful for times when an existing distribution doesn't suit your needs—for example, if you were building a sentient robotic spider that could talk and understand the spoken word. You'd most likely want to build a custom embedded Linux environment, so you could include the WORLD_DOMINATION.c module and, my favorite, the STAY_AWAY_FROM_SHAWN.c module. The latter is available to any robotic spider programmers free of charge.

This month isn't all about embedded Linux, however. Whether you learn about using the OAuth protocol from my friend Adrian Hannah or want to figure out days of the week in a script with Dave Taylor, this issue has you covered. We've also got our regular lineup of new product announcements, UpFront tidbits and enough geeky tips and tricks to keep any Linux lover happy. And remember, if this embedded issue is making you feel a bit too small, we'll try to save you some of that cake with the "Eat Me" sign next to it. It worked for Alice! ■

Shawn Powers is the Associate Editor for *Linux Journal*. He's also the Gadget Guy for LinuxJournal.com, and he has an interesting collection of vintage Garfield coffee mugs. Don't let his silly hairdo fool you, he's a pretty ordinary guy and can be reached via e-mail at shawn@linuxjournal.com. Or, swing by the #linuxjournal IRC channel on Freenode.net.



ALL ABOUT THE CLOUD

M A Y 2 3 - 2 6 , 2 0 1 1

ATTENTION LINUX JOURNAL READERS:

Plan your participation now in the software industry's most comprehensive annual ISV conference for Cloud Computing!

If you are passionate about:

- securing the cloud
- business in the cloud
- platforms & infrastructure
- social cloud
- integrating in the cloud
- mobile cloud
- government cloud
- monetizing cloud

then you MUST attend All About the Cloud!

**RECEIVE \$100 OFF
ENTER CODE PRMLNX
AT CHECKOUT**

Note: this code is valid off the individual SIIA non-member/OpSource non-customer rate only and does not apply to current attendees.

Register Now Online
ALLABOUTTHECLOUD.NET

DIAMOND SPONSOR

- Microsoft

PLATINUM SPONSORS

- IBM
- SafeNet
- SAP

CODIE AWARDS SPONSOR

- Grant Thornton

OFFSITE NETWORKING EVENT SPONSOR

- Dell Boomi

GOLF TOURNAMENT SPONSOR

- Progress Software

GOLD SPONSORS

- Accenture
- Agilis Solutions
- Host Analytics
- Ping Identity
- Rackspace
- Savvis
- SoftLayer Technologies

ATTENDEE BAG SPONSOR

- NTT America

NETWORKING BREAKS SPONSOR

- AppFirst

SILVER SPONSORS

- Corent Technology, Inc.
- FiancialForce.com
- Informatica
- Keynote
- Pervasive
- SaaShr.com
- XBOSoft, Ltd.

INDUSTRY PARTNERS

- Backbone Magazine
- CIOZone
- Cloudbook
- CloudTweaks
- Integration Developer News
- Mashable
- Saugatuck Technology Inc.
- Software Pricing Partners
- THINKstrategies

VIDEO PARTNER

- DreamSimplicity

P R E S E N T E D B Y

 **OpSource™**

 **SIIA**

**THE PALACE HOTEL
S A N F R A N C I S C O**

letters



Acer Aspire One and Linux

I noticed that someone said the background of the Acer Aspire One cannot be changed, so far as he or she knows—presumably when running a Linux-based OS. Well, the background can be changed easily. Just right-click anywhere on the desktop and select the tabbed option for background. Also, you can add more pictures from your own pictures folder if you want.

But, that won't work if you are using the Netbook version of, for example, Ubuntu. In that case, you need to go to your own pictures folder and click on a picture. Then, there is an option to access background preferences.

I purchased an Acer Aspire One, AOD260, which had Windows XP as its OS. I "upgraded" it myself to the full version of Windows 7 Premium as the primary OS and Linux Ubuntu (full) as the secondary. Both worked fine except for an irritating failure on Internet time updating. Since then, I have changed permanently to Linux Ubuntu (full version) and it is running beautifully.

--
Derek

Netflix and Linux

I have heard you mention the inability to

stream Netflix content and share your angst with that fact. However, recently, we have been given that ability! Some people will shun me for admitting this, but I use Google Chrome as my browser (yes, despite its inherent memory leaks), but the latest stable update I received added Chrome WebApp functionality, and guess what? There is a Netflix app! I don't have a login anymore, because I canceled my subscription because I can't stand booting Windows every time I want to watch a movie. Let me know if it works. I am very curious. It started for me, but I wasn't able to test it.

--
Philip

Sadly, no. The problem with Netflix under Linux is that Netflix uses Silverlight for its streaming/playback/rendering solution. Although we do have Moonlight, a Linux native version of Silverlight, it doesn't support DRM, so the video will not start streaming.—Ed.

Boot Challenge

First, thank you for a wonderful magazine. I'm on my third annual subscription, and I really look forward to reading it every month.

I have a challenge that I would like to put forward, after reading the article on UNetbootin that I hope you may consider taking. [See Joey Bernard's "Linux on a Fingernail" in the Upfront section of the March 2011 issue.]

Recently, I bought a Zotac Zbox that I wish to run Linux on. I have one SD card and a USB stick. Both are recognized by the system. In the BIOS, I can set either as the first boot device, which will become /dev/hda for Linux. I want to make the SD card my "hard drive".

I used UNetbootin to create the USB stick bootable with an installer ISO. The aim is to install Linux on the SD card and later boot from it. Booting goes okay, and I can install Linux (CentOS in my case) on the SD card. Later, I wanted to boot from the SD card, but as you might have guessed, CentOS installed on

/dev/hdb, and hence, will not boot.

I have thought of different options, like attaching an external SATA DVD-ROM drive and booting from that, but that would be too simple, plus, I don't have such a drive. The other option (not tested) is to install Syslinux on the SD card, boot from it, and use GRUB to boot the USB stick with an installer ISO image that has been made with UNetbootin, and then install to /dev/hda. A third option (not tested either) would be PXE boot, but I don't have a PXE server at this time. The fourth option is to go back to my installed CentOS on the SD card and modify the mount options and the bootloader (not tested either). So, I hope you will take my challenge!

--
Simon Stavdal

I would recommend the fourth option, and possibly mounting with the UUID instead of the device name. (Ubuntu does this by default.) You'll likely still have to fiddle with GRUB as well, but it should work. I'll admit though, sorting out these sorts of problems can be very frustrating.—Ed.

Spaces in Filenames, Revisited

Regarding Dave Taylor's article "Dealing with Spaces in Filenames" in the February 2011 issue, I never put spaces in my filenames when I create file/folders. But I do get files from others that have spaces in them. I used to leave those files with the spaces in them, even though they are not good. Your article got me to change that, so I wrote a simple script that seems to work real well. Here it is:

```
for s in *\ *
do
    rename 's/ _/g' "$s"
    sleep 1
    printf "Removing spaces from
\$s\n"
done
```

Thanks for your great article.

--
caseesac

Author Update on “Zotonic: the Erlang Content Management System”, LJ, April 2011

As it stands, the default Zotonic branch is under heavy development, and installation instructions have changed slightly. You can clone the 0.6 version instead using hg clone -r release-0.6.x <https://zotonic.googlecode.com/hg/zotonic>. If you want to live on the edge, see the new instructions for the default branch: code.google.com/p/zotonic/source/browse/doc/INSTALL. You also can check out the very active Zotonic Google group if you need further help.

--
Michael Connors

Fedora Packagers Refuse to Fix Security Hazard in glibc Packages

A security hazard was introduced in glibc as of version glibc-2.12.90-4 and has been left uncorrected for nearly four months. Namely, glibc allows other programs to overwrite adjacent memory blocks.

The bug was narrowed down to an inappropriate use of memcpy instead of memmove in glibc on 08.11.2010—see comments 37 and 38 of https://bugzilla.redhat.com/show_bug.cgi?id=638477.

Unfortunately, this bug thread has disintegrated into a debate over the pros and cons of support for proprietary software, completely ignoring the main issue—a security hazard in glibc that needs to be fixed immediately. Despite critical comments from Linus Torvalds (see comments 199 and 222), nothing has changed.

The only other possible course of action is to advise Fedora 14 users to downgrade or roll back glibc until such time as this bug is fixed.

--
Simon Lewis

Thank you for the heads up. Instances like this are the reason I try not to be smug as a Linux user. Yes, our systems are designed well and are usually rock solid, but a coding error is a coding error. We are not impervious to bugs.—Ed.

Paul Barry’s “Python for Android”, LJ, March 2011

I’ve been a loyal reader for some time, mostly because of articles like “Python

for Android” by Paul Barry. Not only did things work exactly as described, but the descriptions themselves were perfect for a technical but not über-geek like myself.

Sadly, I did run into trouble when trying to follow the article using a physical Android phone. My phone has “security features” that prevent loading of applications that do not pass through the mobile carrier’s “App Market” toll booth. I suspect that I’m not alone with this restriction. My troubles are complicated further by the fact that most postings about “side loading”—getting apps onto phones using various unofficial techniques—require and presume root access. Neither of these will frighten a thoroughbred developer (aka, “geek”), but fear of bricking one’s smartphone remains daunting to most bank accounts.

--
Dan St. Andre

Paul Barry replies: Thanks for the kind words, Dan. Unfortunately, some vendors are intent on sealing tight what should be an open platform. This might make business sense to them, but it certainly makes life difficult for some of their users (especially us hacker types who want to control everything their smartphones can do). I’ve very little experience rooting Android devices, although there’s plenty of help on the Net. You may wish to search answers.oreilly.com for articles by Brian Sawyer. Brian was the editor on my two Head First books and (as well as an editor) is a published Android author. He has tried most things on his smartphone and offers some good advice on getting root access. Search for “How to root your Android phone with one click”, and take things from there. But, be careful, as the possibility of bricking is an issue, so tread carefully.

Cool Application Package

I don’t recall which monthly Linux magazine recently asked readers to submit their favorite applications that they can’t live without.

I’ve been a longtime user of the XFCE, and one thing it has that I haven’t found anywhere else is the ability to run the mouse off the edge of the screen onto the next screen and wrap around

from the last screen to the first again.

Brightside does this for GNOME, and it works on Ubuntu Netbook Remix Unity as well. It also does vertical screen switching for screen layouts that are not side by side.

--
honkytonkwillie

I often set up my desk like this at work. I have three monitors and three operating systems. Using Synergy, I can move my mouse between them all and have them wrap from the last to the first as well. Doing the same on virtual desktops in a single screen might be interesting. Thanks for the tip.—Ed.

Thank You, Kyle

I read most issues of *Linux Journal* cover to cover; it’s the only mag that grabs so much of my attention. For all the excellent contributors, one stands out perhaps even just a little more: Kyle Rankin.

Coming from a Mac background, learning Bash was a weakness for me initially, but Kyle’s column has taught me enough to feel confident doing things I’d never considered before. I feel like I truly own my machine now, and a big part of that is Kyle’s experience coupled with his excellent writing style.

In fact, I recently decided to install an experimental server in my office, so of course, my first choice was Kyle’s *Official Ubuntu Server* book—every bit as good as his column here.

Many thanks to Kyle and the others there at *Linux Journal* who’ve helped turn me from a total noob into someone who’s feeling confident and competent. You folks are the best!

PS: Thanks for your SCaLE 9x sponsorship. I loved the conference, and I really appreciate the role *Linux Journal* played in supporting it.

--
Richard Gaskin

*Yeah, we like Kyle around these parts as well. As far as his book goes, I’ve been a Linux admin for more than 15 years, and I still refer to his *Official Ubuntu Server Book* rather often. Which reminds me, I still need to get my second edition signed.—Ed.*

More on Paul Barry's "Python for Android"

I really enjoyed Paul Barry's "Python for Android" article. Working through it though, I thought about how Paul had the readers develop the program twice—once to run on a common computer for the proof of concept, then again to add the Android UI and remove the debugging print statement.

Once a programmer is more familiar with Python than with Android, I think it can help to have some test tools to deal with the whole program while it's being developed outside the simulator. I put this module, named android.py, in my development directory:

```
class Android (object):
    '''Emulate android API without android.'''
    def __getattr__ (self, name):
        '''Presume that unknown attribute requests are for methods.'''
        def log_method (*args, **kwargs):
            print 'Android.%s ::' % (name,), args, kwargs
        return log_method

    def dialogGetResponse (self, *args, **kwargs):
        class DGR (object):
            def __init__ (self, result='???'):
                self.result = result
            print 'Android.dialogGetResponse ::', args, kwargs
        return DGR ('OK')
```

This traces most API calls along with the given parameters. dialogGetResponse was cobbled together specially to keep LJapp.py from failing. Running LJapp.py with android.py in an ordinary non-Android shell session gives:

```
mwilson@tecumseth:~/sandbox/android-app$ python LJapp.py
Android.makeText :: ('Hello from LJapp.',)
Android.dialogCreateSpinner :: ('LJapp',
    → 'Checking the price of coffee...')
{
Android.dialogShow :: ()
Android.dialogDismiss :: ()
Android.vibrate :: ()
Android.dialogCreateAlert :: ('The current
    → price of coffee beans:',)
Android.dialogSetItems :: ([5.190000000000004],)
Android.dialogSetPositiveButtonText :: ('OK')
Android.dialogShow :: ()
Android.dialogGetResponse :: ()
Android.makeText :: ('bye')
```

Of course in Python, test frameworks and test-driven development are huge things; this is mere baby talk compared to what developers will be doing very soon. I think it's interesting anyway.

--

Mel Wilson

Paul Barry replies: Thanks for sending this. It's a pretty cool piece of code. Could I suggest that you sign up to the SL4A mailing list and share this script with the developers frequenting the list?

At Your Service**MAGAZINE**

PRINT SUBSCRIPTIONS: Renewing your subscription, changing your address, paying your invoice, viewing your account details or other subscription inquiries can instantly be done on-line, www.linuxjournal.com/subs. Alternatively, within the U.S. and Canada, you may call us toll-free 1-888-66-LINUX (54689), or internationally +1-818-487-2089. E-mail us at subs@linuxjournal.com or reach us via postal mail, Linux Journal, PO Box 16476, North Hollywood, CA 91615-9911 USA. Please remember to include your complete name and address when contacting us.

DIGITAL SUBSCRIPTIONS: Digital subscriptions of *Linux Journal* are now available and delivered as PDFs anywhere in the world for one low cost. Visit www.linuxjournal.com/digital for more information or use the contact information above for any digital magazine customer service inquiries.

LETTERS TO THE EDITOR: We welcome your letters and encourage you to submit them at www.linuxjournal.com/contact or mail them to Linux Journal, PO Box 980985, Houston, TX 77098 USA. Letters may be edited for space and clarity.

WRITING FOR US: We always are looking for contributed articles, tutorials and real-world stories for the magazine. An author's guide, a list of topics and due dates can be found on-line, www.linuxjournal.com/author.

ADVERTISING: *Linux Journal* is a great resource for readers and advertisers alike. Request a media kit, view our current editorial calendar and advertising due dates, or learn more about other advertising and marketing opportunities by visiting us on-line, www.linuxjournal.com/advertising. Contact us directly for further information, ads@linuxjournal.com or +1 713-344-1956 ext. 2.

ON-LINE

WEB SITE: Read exclusive on-line-only content on *Linux Journal's* Web site, www.linuxjournal.com. Also, select articles from the print magazine are available on-line. Magazine subscribers, digital or print, receive full access to issue archives; please contact Customer Service for further information, subs@linuxjournal.com.

FREE e-NEWSLETTERS: Each week, *Linux Journal* editors will tell you what's hot in the world of Linux. Receive late-breaking news, technical tips and tricks, and links to in-depth stories featured on www.linuxjournal.com. Subscribe for free today, www.linuxjournal.com/eneWSLETTERS.



2011 MAY 16–19
BALTIMORE, MARYLAND



► REGISTER NOW & SAVE 15%

Use discount code **rc11ljr**

co-presented by **O'REILLY**



O'REILLY®
Velocity
Web Performance & Operations
CONFERENCE

2011 June 14–16
Santa Clara, California



► REGISTER NOW & SAVE 15%

Use discount code **vel11ljr**

Presented by **O'REILLY**

diff -u

WHAT'S NEW IN KERNEL DEVELOPMENT

Ahmed S. Darwish recently initiated an abortive effort to improve the way **system crashes** were logged. When a system, particularly a laptop, crashes very early in the boot process, typically no record is kept. Ahmed's idea was to rely on BIOS routines to write logging information to disk even if a crash occurs almost immediately. There was a lot of interest in Ahmed's work, since it could be a significant aid to debugging. But, **Linus Torvalds** torpedoed the idea, saying, "No way in hell do I want the situation of 'the system is screwed, so let's overwrite the disk' to be something the kernel I release might do. It's crazy. That disk is a lot more important than the kernel." So clearly, this type of feature almost certainly will remain a third-party patch for a long time to come.

The **ARM architecture** has quite a few implementations of the **clk struct**, which annoyed **Jeremy Kerr** and inspired him to rewrite them all into a single generic implementation. A lot of folks pitched in with suggestions. Some of the trickier points involved identifying the proper way to tell when multiple devices were using a single clock. It also was important to make sure that the generic code have appropriate boundaries, so it would be easier to write special case code for the systems that needed it.

Oren Weil of **Intel** has submitted a patch to support Intel's new **management engine**. Unfortunately, there was a variety of problems with the patch submission process itself, including leaving off the crucial Signed-Off-By headers, among other things. So, much of the discussion centered around that instead of the code. One key problem was that patches are never supposed to break the kernel build. This is essential even if subsequent patches would fix the build. The reason is that developers need to be able to rely on "git bisect" to track down bugs, and this is possible only if every version of the kernel builds properly. This also makes for a cleaner development process in general, in which no single project is stuck waiting for another project to fix the build before it can continue doing its own tests and development.

Thomas Gleixner has revamped the **generic interrupt handling code** in the kernel. This was partly done to clean up naming conventions so as to be easier for developers to understand, but it also was done to encapsulate the various features of the system, so developers couldn't abuse them anymore—or at least so developers who abused them would be easy to identify and stop. One of the goals is to extend the IRQ handling code gradually to accommodate the special needs of all the various projects out there, and that's more difficult to do when folks are reaching their misshapen prongs into the guts of the code.

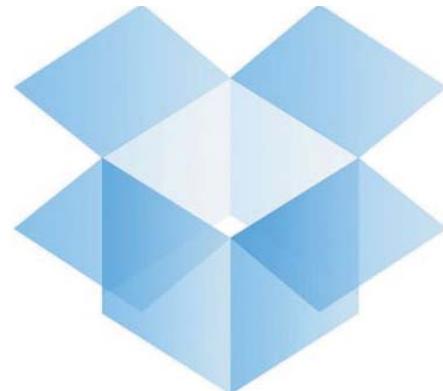
Eric Paris wanted to re-introduce a **global capabilities bounding set** to make it possible to remove a capability from the system in such a way that it could not be gotten back. The idea would be to remove some key capabilities and then hand root access over to untrusted entities. Eric's idea was that this would keep the system secure. Unfortunately, Linux security is implemented primarily in terms of ways to prevent root access, rather than ways to mitigate the effect of granting root access. The assumption made by virtually all kernel developers is that once a user has gained root access, that's the ball game. So something like Eric's subtle manipulation of root privileges does not ring true to most kernel folks.

—ZACK BROWN

Dropbox Tips and Tricks

Dropbox, or one of the alternatives like Ubuntu One or SparkleShare, are great tools for keeping computers in sync. They offer some unique abilities as well. Here are a few of our favorites:

- Keep config folders, like Pidgin's .purple directory in your Dropbox, and symlink to it in your home directory. It saves entering the same information on your computers.
- Have your home computer monitor a specific folder inside Dropbox for torrent files. Then, whenever you want to download a torrent at home, just put the torrent file into your folder on any computer, and it will start downloading at home.
- Keep your favorite desktop wallpaper photos in Dropbox, so you have access to all your NASA astronomy pictures on your various desktop and laptop computers. (This works for non-NASA photos too, but why would you want non-space-based wallpaper?)
- Use Dropbox in combination with Calibre and Calibre2opds to keep your e-book library on-line and always accessible. It makes downloading a book from your collection simple when you're on vacation and forget to pack enough books.



Do you have more Dropbox tips? Send them to me at shawn@linuxjournal.com, and I'll publish the good ones on LinuxJournal.com.

—SHAWN POWERS



NON-LINUX FOSS

If you're a fan of Quicksilver on OS X or GNOME Do on Linux, Launchy might be just the thing your Windows install needs. An open-source application launcher, Launchy stays out of the way until you need it. A simple keystroke brings it to the foreground and helps you launch whatever application you desire. Check it out at www.launchy.net.

—SHAWN POWERS

Adding More Awesome to Your Office

Whether you prefer OpenOffice.org or LibreOffice, which currently still are pretty similar, out of the box, they are missing some of the conveniences installed by their commercial counterparts. Granted, they are fully functional, but if you want a robust clip-art library and a decent selection of document templates, you'll want to add some extensions and templates.

Browse on over to extensions.services.openoffice.org and templates.services.openoffice.org to pick up some add-ons that will make your office suite shine. Whether you want to add a few graphics to your document or spice up your next presentation, the options are extensive.

Also, if your user has write access to the system files, you'll get the option to install extensions for all users or just the current user—an awesome boon for sysadmins like myself!

—SHAWN POWERS

We Want to Hear from YOU at LINUXJOURNAL.COM

The Internet is a marvelous thing. I know, "Welcome to 1995, Katherine", but hear me out. You hold in your hands a marvelous source of information about Linux and open-source software. I think it's a pretty fantastic use of paper and ink, but it's a one-way street. LinuxJournal.com on the other hand, allows for numerous opportunities to communicate both ways. I'd love to see each of you take advantage of the on-line community to exchange ideas in the comments sections of your favorite articles or communicate with our authors and staff directly via our LinuxJournal.com profiles (I'm www.linuxjournal.com/users/webmistress). Visit often to check out the latest poll and vote. Wondering which office software *Linux Journal* readers prefer? Now you know: www.linuxjournal.com/content/whats-your-favorite-office-program.

We want to hear from you so we can continue to provide you with the content you enjoy most and to get to know you a little better. Don't be shy! Visit LinuxJournal.com and say hi.

—KATHERINE DRUCKMAN

They Said It

All sorts of computer errors are now turning up. You'd be surprised to know the number of doctors who claim they are treating pregnant men.

—Isaac Asimov

Humanity has the stars in its future, and that future is too important to be lost under the burden of juvenile folly and ignorant superstition.

—Isaac Asimov

I am not a speed reader.
I am a speed understander.

—Isaac Asimov

I do not fear computers.
I fear the lack of them.

—Isaac Asimov

If my doctor told me I had only six minutes to live, I wouldn't brood. I'd type a little faster.

—Isaac Asimov

Parallel Programming Crash Course

I've been covering various scientific programs the past few months, but sometimes it's hard to find a package that does what you need. In those cases, you need to go ahead and write your own code. When you are involved with heavy-duty scientific computing, you usually need to go to parallel computing in order to get the runtimes down to something reasonable. This month, I give a crash course in parallel programming so you can get a feel for what is involved.

There are two broad categories of parallel programs: shared memory and message passing. You likely will see both types being used in various scientific arenas. Shared-memory programming is when all of the processors you are using are on a single box. This limits you as to how big your problem can be. When you use message passing, you can link together as many machines as you have access to over some interconnection network.

Let's start by looking at message-passing parallel programming. The most common version in use today is MPI (Message Passing Interface). MPI is actually a specification, so many different implementations are available, including Open MPI, MPICH and LAM, among others. These implementations are available for C, C++ and FORTRAN. Implementations also are available for Python, OCaml and .NET.

An MPI program consists of multiple processes (called slots), running on one or more machines. Each of these processes can communicate with all other processes. Essentially, they are in a fully connected network. Each process runs a full copy of your program as its executable content and runs independently of the others. The parallelism comes into play when these processes start sending messages to each other.

Assuming you already have some MPI code, the first step in using it is to compile it. MPI implementations include a set of wrapper scripts that handle all of the compiler and linker options for you. They are called mpicc, mpiCC, mpif77 and mpif90, for C, C++, FORTRAN 77 and FORTRAN 90, respectively. You can add extra options for your compiler as options to the wrapper scripts. One very useful option is -showme. This option simply prints out the full command line that would be used to invoke your compiler. This is useful if you have multiple compilers and/or libraries on your system, and you need to verify that the wrapper is doing the right thing.

Once your code is compiled, you need to run it. You don't actually run your program directly. A support program called mpirun takes care of setting up the system and running your code. You need to tell mpirun how many processors you want to run and where they are located. If you are running on one machine, you can hand in the number of processors with the option -np X. If you are running over several machines, you can hand in a list of hostnames either on the command line or in a text file. If this list of hostnames has repeats, mpirun assumes you want to start one process for each repeat.

Now that you know how to compile and run your code, how do you actually write an MPI program? The first step needs to initialize the MPI subsystem. There is a function to do this, which in C is this:

```
int MPI_Init(&argc, &argv);
```

Until you call this function, your program is running a single thread of execution. Also, you can't call any other MPI functions before this, except for MPI_Initialized. Once you run MPI_Init,

MPI starts up all of the parallel processes and sets up the communication network. After this initialization work is finished, you are running in parallel, with each process running a copy of your code.

When you've finished all of your work, you need to shut down all of this infrastructure cleanly. The function that does this is:

```
int MPI_Finalize();
```

Once this finishes, you are back to running a single thread of execution. After calling this function, the only MPI functions that you can call are MPI_Get_version, MPI_Initialized and MPI_Finalized.

Remember that once your code goes parallel, each processor is running a copy of your code. If so, how does each copy know what it should be doing? In order to have each process do something unique, you need some way to identify different processes. This can be done with the function:

```
int MPI_Comm_rank(MPI_Comm comm, int *rank);
```

This function will give a unique identifier, called the rank, of the process calling it. Ranks are simply integers, starting from 0 to N-1, where N is the number of parallel processes.

You also may need to know how many processes are running. To get this, you would need to call the function:

```
int MPI_Comm_size(MPI_Comm comm, int *size);
```

Now, you've initialized the MPI subsystem and found out who you are and how many processes are running. The next thing you likely will need to do is to send and receive messages. The most basic method for sending a message is:

```
int MPI_Send(void *buf, int count, MPI_Datatype type,
    int dest, int tag, MPI_Comm comm);
```

In this case, you need a buffer (buf) containing count elements of type type. The parameter dest is the rank of the process that you are sending the message to. You also can label a message with the parameter tag. Your code can decide to do something different based on the tag value you set. The last parameter is the communicator, which I'll look at a little later. On the receiving end, you would need to call:

```
int MPI_Recv(void *buf, int count, MPI_Datatype type,
    int source, int tag, MPI_Comm comm, MPI_Status *status);
```

When you are receiving a message, you may not necessarily care who sent it or what the tag value is. In those cases, you can set these parameters to the special values MPI_ANY_SOURCE and MPI_ANY_TAG. You then can check what the actual values were after the fact by looking at the status struct. The status contains the values:

```
status->MPI_source
status->MPI_tag
status->MPI_ERROR
```

Both of these functions are blocking. This means that when

you send a message, you end up being blocked until the message has finished being sent. Alternatively, if you try to receive a message, you will block until the message has been received completely. Because these calls block until they complete, it is very easy to cause deadlocks where, for example, two processes are both waiting for a message to arrive before any messages get sent. They end up waiting forever. So if you have problems with your code, these calls usually are the first places to look.

These functions are point-to-point calls. But, what if you want to talk to a group of other processes? MPI has a broadcast function:

```
int MPI_Bcast(void *buf, int count, MPI_Datatype type,
    int root, MPI_Comm comm);
```

This function takes a buffer containing count elements of type type and broadcasts to all of the processors, including the root process. The root process (from the parameter root) is the process that actually has the data. All the others receive the data. They all call MPI_Bcast, and the MPI subsystem is responsible for sorting out who has the data and who is receiving. This call also sends the entire contents of the buffer to all the processes, but sometimes you want each process to work on a chunk of the data. In these cases, it doesn't make sense to send the entire data buffer to all of them. There is an MPI function to handle this:

```
int MPI_Scatter(void *send, int sendcnt, MPI_Datatype type,
    void *recv, int recvnt, MPI_Datatype type, int root,
    MPI_Comm comm);
```

In this case, they all call the same function, and the MPI subsystem is responsible for sorting out which is root (the process with the data) and which are receiving data. MPI then divides the send buffer into even-size chunks and sends it out to all of the processes, including the root process. Then, each process can work away on its chunk. When they're done, you can gather up all the results with:

```
int MPI_Gather(void *send, int sendcnt, MPI_Datatype type,
    void *recv, int recvnt, MPI_Datatype type, int root,
    MPI_Comm comm);
```

This is a complete reversal of MPI_Scatter. In this case, all the processes send their little chunks, and the root process gathers them all up and puts them in its receive buffer.

Taking all of the information from above and combining it together, you can put together a basic boilerplate example:

```
#include <mpi.h>
// Any other include files

int main(int argc, char **argv){
    int id,size;
    // all of your serial code would
    // go here
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &id);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    // all of your parallel code would
    // go here
    MPI_Finalize();
    // any single-threaded cleanup code
    // goes here
    exit(0);
}
```

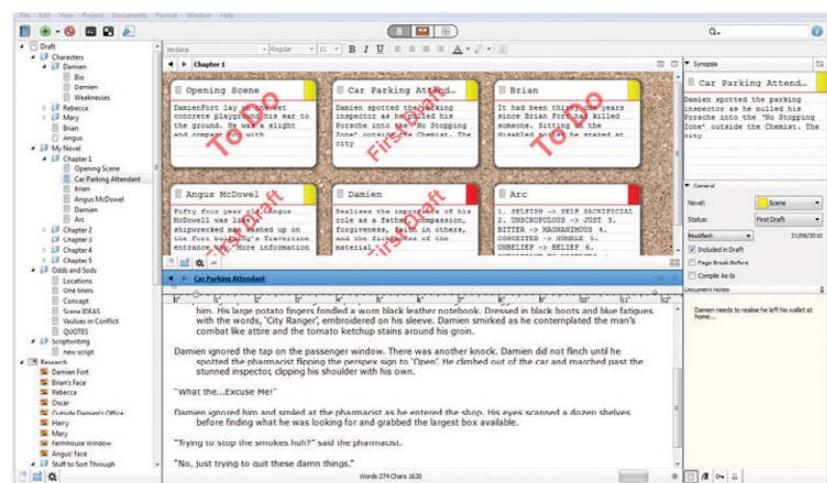
Hopefully, you now feel more comfortable with MPI programs. I looked at the most basic elements here, but if you feel inspired, you should grab a good textbook and see what other functions are available to you. If not, you at least should be able to read existing MPI code and have a good idea of what it's trying to do. As always, if you'd like to see a certain area covered in this space, feel free to let me know.

—JOEY BERNARD

Scrivener, Now for Linux!

The folks over at www.literatureandlatte.com have a rather nifty writer's tool called Scrivener. For years, it's been an OS X-only program for novelists and screenwriters that acts like a project management tool for big writing projects. Linux users may be familiar with Writer's Café from www.writerscafe.co.uk, which is a similar program. Although Scrivener is a little more expensive (\$45 vs. \$40 for Writer's Café), its features make it something any novelist should check out. And, if you try it during the beta period, Scrivener is free.

Unfortunately, users with existing licenses for the OS X version of Scrivener cannot transfer that license to the Linux version. Perhaps once the final version is released, the Literature and Latte folks will change their minds. Either way, if you're a writer, you'll want to check out Scrivener or Writer's Café. Both are neat packages, and now both are Linux-compatible!



—SHAWN POWERS



REUVEN M. LERNER

Backbone.js

Write simple MVC applications in JavaScript with Backbone.js.

JavaScript is changing. Actually, I'm not sure how much of that is true; the underlying language hasn't changed too much over the years. But, even if the language itself isn't changing, everything else about it is. There's a growing interest in server-side JavaScript, with high-speed applications based on Node.js (as described in last month's column). Browser-based JavaScript is not only pretty standard, but also executes very efficiently. And, of course, a number of high-quality, open-source JavaScript libraries exist, such as jQuery, MooTools and Prototype, that make it easy to work with JavaScript within the browser.

So, have our JavaScript demons been exorcised forever? Not at all. As applications migrate to the browser, there is a growing interest in making them even more desktop-like. Sure, JavaScript UI libraries make it relatively easy to implement desktop-like functionality, such as drag and drop. But if you want to create a truly sophisticated, desktop-like application, you're going to find yourself lost in a forest of event callbacks, not to mention widgets that might or might not be appropriate for such an application.

So, have our JavaScript demons been exorcised forever?

Thus, it shouldn't come as a surprise to find that in the last year or two, a new type of Web application has emerged—one written almost purely in JavaScript, which executes inside the browser, and which only occasionally contacts a server. This turns the usual model of Web development—in which the majority of the processing takes place on the server, emitting HTML and JavaScript that handles things until the next call to the server—on its head, making the server little more than a storage facility that stores and retrieves information determined by the browser application.

You could argue that Google Maps, Gmail and Google Docs—to choose three famous examples, but by no means the only ones—have been demonstrating such capabilities for several years. But until recently, it was relatively difficult for average developers to create applications that were heavily based on JavaScript.

Fortunately, things have changed, and in a big way. If you want to create a rich JavaScript application, you have a variety of toolkits from which to choose. The question no longer is whether you can create such an application, but rather, which tools you will use to create it and how it'll talk to the server. Just off the top of

my head, I can recall Backbone.js, Knockout, JavaScript MVC, SproutCore, Closure and Cappuccino, and you can be sure that I'm mentioning only a small fraction of the toolkits that exist. It might go without saying nowadays, but I should add that the leading toolkits are all released under open-source licenses, making it possible to download, try and explore each of these libraries without having to worry about licensing restrictions when downloading or deploying them.

This month, I'm starting a series of columns about several of these in-browser application development frameworks, and how you can use them to create richer, more interesting Web applications. In each case, I'll explore how easy it is to get started with the framework, its relative advantages and disadvantages, and discuss how you might have it interact with data on a server.

During the past decade, we have seen a clear trend toward MVC frameworks on the server side that provide RESTful APIs. Ruby on Rails isn't the only framework that has promoted such a development style, but it certainly has pushed developers hard in those directions, making non-REST and non-MVC development frustratingly difficult. It turns out that many of the new, modern JavaScript frameworks also have adopted the MVC model, each in its own way and always with differences between the server-side model that Rails developers might expect.

Using MVC on the browser and on the server (which I like to call MVC-squared, but maybe that's just me) turns a Web application into two separate software systems: one on the server that's basically exposing a RESTful, JSON API to the world, and another in the browser that's consuming a RESTful, JSON API from a server. Decomposing the program into these two parts makes it easier to split the development work across two individuals or groups, as well as to organize the code in a smarter way. I'll have more to say about this in the coming months, as I connect JavaScript applications to back-end storage systems.

This month, I take an initial look at Backbone.js, a very small JavaScript library that has been getting a fair amount of press recently. And, I explain how to build a simple application using Backbone.js, creating functionality that exists completely within the browser.

The Basics

Backbone.js, as I indicated above, follows the model-view-controller (MVC) paradigm that has been used by software developers for several decades, and that has

become ubiquitous in server-side Web development during the past few years. An MVC application has three distinct parts: the model (which provides an interface to the data itself), the view (which presents information to the user) and the controller (which directs the user's requests to the right models, and then renders the results in the view). By dividing the program logic along these lines, it becomes fairly obvious where each function should go, making the code more maintainable.

In the MVC world of Backbone.js, the split works in a similar way. You retrieve and store data in a model object, individual methods (and URL routes) are defined in a controller object, and the view shows things in the user's browser.

But, if you're coming from the server-side world, there are some subtle (and not-so-subtle) differences between server-side and client-side MVC. For starters, it's pretty clear in a server-side program that the model retrieves data from a database, either relational or non-relational. By contrast, the model in a JavaScript application can come from...well, it can come from a variety of sources, one of which would be a server-side Web application. I'll look into this more next month;

for my examples this month, let's assume that the data is already in JavaScript and doesn't need to be loaded from anywhere.

In a server-side application, the view is actually a combination of HTML, CSS and JavaScript, rather than being a single file or format. Actually, the view doesn't have to be HTML; it also can be XML, JSON or a variety of other formats, from CSV to PDF to images. By contrast, the view in a Backbone.js application typically is going to rewrite a single part of the current page, rather than load an entirely new one.

So with this in mind, let's create a basic Backbone.js application. I've decided to jump onto the social bandwagon and develop a tiny application that lets people look at a list of recipe titles, click on a title that sounds interesting, and then read the contents of the recipe in question. The same principle could apply to an address book, a diary or even an unusually formatted blog.

So, let's start with the data model. Creating a data model in Ruby on Rails (using ActiveRecord) is easy. You define a subclass of ActiveRecord, thus inheriting all of its capabilities. Of course, JavaScript doesn't have a traditional object model with classes and inheritance,



Double-Sided High Capacity Server Solutions

Energy Efficiency, High Availability, Flexible Scalability on Demand, Easy Deployment, Easy Management, Efficient Infrastructure, 6Gb/s MegaRAID®

Introducing our whole new series of Double-Sided High Capacity servers, focus on performance, reliability and mobility. Featuring the **Intel® Xeon® Processor X5680**, **LSI 6Gb/s MegaRAID® SAS 9280 Controller Cards**, optimized hard drive signal trace routing and improved hard drive tray designs to dampen vibrations and maximize drive performance. 100% cooling redundancy; even if an internal cooling fan fails, these systems will continue operating without any performance loss.



**Powerful.
Intelligent.**



LSI, the LSI logo, MegaRAID are trademarks or registered trademarks of LSI Corporation in the U.S. and other countries.

Intel, the Intel logo, Xeon, and Xeon Inside are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries.



Yang Ming International Corp. (RackMountPro.com)

The Leading Server Builder in America. Enhancing Cloud Computing, The Optimized Technologies for Cloud

595 Yorbita Road, La Puente, CA 91744

Tel: (800) 526-8650

Fax: (626) 956-0098

sales@rackmountpro.com

so Backbone.js needs to use a different paradigm. Instead, what you do in Backbone.js is invoke the “extend” function on Backbone.Model. Attributes passed to Backbone.Model.extend either are treated as data fields or as methods, depending on whether they’re data or functions. For example, if you want to model a single appointment, you could do it as follows:

```
Appointment = Backbone.Model.extend({
  person: null,
  meeting_at: null,
  note: null
});
```

Note that you also could define an “initialize” attribute, which would take the place of the default constructor method. In this particular case, I’m not planning to do anything fancy, which means I can use the default. To create a new appointment, you can say:

```
var new_appointment =
new Appointment({person: 'Barak Obama',
  meeting_at: '2011-jul-14',
  note: 'Meet with the president'});
```

You also can replace individual attributes within an appointment:

```
new_appointment.set({person: 'Joe Biden'});
```

Or, you can retrieve an attribute from an appointment:

```
new_appointment.get('person');
```

Collections and Controllers

Of course, most people have to schedule more than one appointment, which means that this example program needs to keep track of more than one at a time. Now, you normally might assume that you simply could store more than one appointment in a JavaScript array. But, in the world of Backbone.js, you actually use a special kind of object, known as a collection, to store appointments.

Why a collection and not simply an array? Mostly because it works together with other items in Backbone.js. For example, you can set things such that whenever you add or remove an element to your collection, it automatically will invoke another method. For another, collection objects incorporate the Underscore library for JavaScript, which defines a number of methods from functional programming, such as map and pluck, so retrieving information from your collection is quite straightforward.

Just as you defined a model by extending Backbone.Model, you define a collection by extending

Backbone.Collection:

```
Appointments = Backbone.Collection.extend({
});
```

Any attributes that you define on the collection are then available, as data or functions, on collection objects of this type. In this particular case, I defined two different attributes, the `initialize` constructor and the `update_appointment_counter` method:

```
Appointments = Backbone.Collection.extend({
  update_appointment_counter: function() {
    $("#number-of-appointments").html(this.length);
  },
  initialize: function(models, options) {
    $("#number-of-appointments").html(this.length);
    this.bind("add", options.view.add_appointment_row);
    this.bind("add", this.update_appointment_counter);
  }
});
```

In this case, the constructor uses jQuery to initialize the appointment length counter (to zero, given that the collection is only now being initialized) and then adds two handlers to the “add” event. Each time you add a new appointment to this collection, two different functions will fire. One of them (`options.view.add_appointment_row`) will add a new row to the HTML table containing a list of appointments, and the other (`this.update_appointment_counter`) updates the counter. As you can see, the functions can be defined in a variety of places; it probably would have made more sense to put both of these methods on the view.

Experienced JavaScript programmers know what “this” is; thus, `this.update_appointment_counter` makes sense. But, what is `options.view`? Well, it might help to see how you create your collection, inside the view constructor:

```
initialize: function() {
  this.appointments = new Appointments(null, {view:this});
},
```

Basically, you’re saying that the `appointments` attribute for the view is an `Appointments` collection, starting with no data. Passing a second parameter allows you to set one or more options in a JavaScript object, which is then available as “options”. Because the view passes itself (!) as the “view” option when creating the collection, you then can access the view from within the collection as `options.view`.

The upshot is that your view, thus, has access to your collection (as `this.appointments`), and your collection has access to our view (as `options.view`). This sort of simple, two-way communication is typical for Backbone.js, which tries to make things as simple and short as possible.

The code doesn't include a controller. That's because controllers are necessary only if you want to provide a number of different URLs—well, fragments at the end of a URL—that invoke different methods. For now, you can do without it, but a larger application certainly will require it.

Views

As always in the MVC paradigm, the view is where things are displayed to (and interact with) the end user. In the Rails world, a view is almost always rendered by the controller; your application doesn't need to create it explicitly. In the Backbone.js world, a view is just another object that can be created, often by a model, and which has many controller-like functions. You create it, as you might expect, with:

```
AppView = Backbone.View.extend({
```

```
});
```

So, you can think of Backbone.js views as fragments of HTML that are added to the current page, plus some of the functionality that you might associate with a controller. Each view is associated with a DOM element. By default, it's a regular "div" element, but you either can set it in one place (using the "el" attribute), or you can set it using a combination of the "tagName", "className" and "id" attributes as well.

As with models and collections, you can use the "initialize" constructor to set up one or more objects. In the case of this example application, you'll initialize your Appointments collection without any element members, as you saw above when I discussed that collection.

You also will define an event handler, such that clicking on the "add-appointment" button will do so:

```
events: {  
  "click #add-appointment": "add_appointment"  
},
```

When you click on the button, the following



For more information visit
www.siliconmechanics.com/R350,
www.siliconmechanics.com/A350,
or call us toll free at **866-352-1173**.



There's a lot of heavy lifting going on these days in the world of data processing. Just ask Jason, a Silicon Mechanics Sales Expert who fields questions every day about the best way to address a vast range of computing workloads. One solution finding enthusiastic adoption is hybrid CPU / GPU computing, such as with the Silicon Mechanics Rackform iServ R350 and the Rackform nServ A350.

We start with your choice of two state-of-the-art processors, for fast, reliable, energy-efficient processing. Then we add two NVIDIA® Tesla GPUs, to dramatically accelerate parallel processing for applications like ray tracing and finite element analysis. Load it up with DDR3 memory and you have herculean capabilities and an 80 PLUS Gold Certified power supply, all in the space of a 1U server.

When you partner with Silicon Mechanics, you get more than breakout technologies that will carry the weight of your workload—you get an expert like Jason.

Expert included.

Silicon Mechanics and the Silicon Mechanics logo are registered trademarks of Silicon Mechanics, Inc.

Listing 1. appointments.html

```

<!DOCTYPE html>
<html>
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/
➥1.4.4/jquery.min.js"></script>
<script src="http://ajax.cdnjs.com/ajax/libs/underscore.js/
➥1.1.4/underscore-min.js"></script>
<script src="http://ajax.cdnjs.com/ajax/libs/backbone.js/
➥0.3.3/backbone-min.js"></script>

<title>Appointments</title>
</head>
<body>
<h1>Appointments</h1>

<table>
<tr>
<th>Person</th>
<th>Date/time</th>
<th>Note</th>
</tr>
<tr id="new-appointment">
<td><input type="text" name="person" /></td>
<td><input type="text" name="meeting_at" /></td>
<td><input type="text" name="note" /></td>
</tr>
<tr align="center">
<td colspan="3"><input type="button" id="add-appointment"
➥value="Add Appointment" /></td>
</tr>
</table>

<hr />

<p>Number of appointments: <span id="number-of-appointments">
➥</span></p>

<table id="appointments">
<tr>
<th>Person</th>
<th>Date/time</th>
<th>Note</th>
</tr>
</table>

<script type="text/javascript">
(function ($) {
    Appointments = Backbone.Collection.extend({
        update_appointment_counter: function() {
            $("#number-of-appointments").html(this.length);
        },
        initialize: function(models, options) {
            $("#number-of-appointments").html(this.length);
            this.bind("add", options.view.add_appointment_row);
            this.bind("add", this.update_appointment_counter);
        }
    });
    AppView = Backbone.View.extend({
        el: $("body"),
        initialize: function() {
            this.appointments = new Appointments(null, {view:this});
        },
        events: {
            "click #add-appointment": "add_appointment"
        },
        add_appointment: function() {
            var person = $("#new-appointment
➥td input[name=person]").val();
            var meeting_at = $("#new-appointment
➥td input[name=meeting_at]").val();
            var note = $("#new-appointment
➥td input[name=note]").val();
            this.appointments.add({person: person,
➥meeting_at: meeting_at, note: note});
        },
        add_appointment_row: function(model) {
            $("#appointments").append("<tr><td>" +
                model.get('person') + "</td>" +
                "<td>" + model.get('meeting_at') + "</td>" +
                "<td>" + model.get('note') + "</td></tr>");
        }
    });
    var appview = new AppView(
        })(jQuery);
    </script>
    </body>
    </html>
}

```

code is executed:

```
add_appointment: function() {
  var person = $("#new-appointment td input[name=person]").val();
  var meeting_at = $("#new-appointment td
    ↵input[name=meeting_at]").val();
  var note = $("#new-appointment td input[name=note]").val();

  this.appointments.add({person: person, meeting_at: meeting_at,
    ↵note: note});
},
```

In other words, when you click on the “add-appointment” button, the “click” event handler executes the add_appointment function. This function grabs the values from the little form and uses those values to instantiate a new appointment, adding it to the collection of appointments.

But, you also have event handlers running on the collection! The first handler updates the appointment counter, and the second adds a new row to the table of appointments. It adds the row by cheating a little bit. Although it would have been more elegant to have a second view with an element of “tr” that would add a new row, I decided to mimic some of the on-line tutorials I’ve seen, adding a new row in a slightly simpler way—namely, an ugly text string.

If I weren’t interested in creating an entirely new view, I could have used the “template” function that Backbone.js inherits from underscore.js, giving me ERb-like templates that can be filled in more nicely. Something else that I could have done is break this application into smaller pieces. Although it’s nice to have everything in a single file when working on something small, a larger Backbone.js application

could well be put into multiple files, with each file defining a different object. Developers experienced with any modern server-side MVC framework, such as Rails or Django, will understand the advantages of putting things into separate files.

Conclusion

Backbone.js is one of the smallest and easiest-to-understand MVC frameworks for JavaScript applications. It has become quite popular, as evidenced by the number of blog posts about it in the past few months. The support that its authors, Jeremy Ashkenas and others at DocumentCloud, have offered to many Backbone.js users has been quite impressive to see as well.

Although this column obviously didn’t go into great depth about Backbone.js, one shortcoming in this application should have been obvious. What happens when the user wants to store data? Right now, the appointment calendar is not only simple-minded in its interface and execution (for example, there’s no way to look at just today’s appointments, let alone remove or edit existing ones), but it also fails to provide persistent storage.

Next month, I’ll discuss how you can connect a Backbone.js application to a persistent back-end database or server-side MVC application (thus providing an MVC-squared solution), giving users and developers the best of both worlds—flexible development with dynamic JavaScript, but with a robust back end that can persist data easily.■

Reuven M. Lerner is a longtime Web developer, architect and trainer. He is a PhD candidate in learning sciences at Northwestern University, researching the design and analysis of collaborative on-line communities. Reuven lives with his wife and three children in Modi'in, Israel.

Resources

The home page for Backbone.js is on GitHub, at documentcloud.github.com/backbone. This page points not only to the code, but also to some tutorials and documentation. In a step that I hope many other authors will follow, the authors of Backbone.js put up a copy of the source code, thoroughly commented in a beautiful format, at documentcloud.github.com/backbone/docs/backbone.html.

I encourage anyone interested in Backbone.js to read through the code and comments. I certainly learned some things about Backbone.js in particular and JavaScript in general from reading through this code.

A number of tutorials and blog postings describe how to do interesting things with Backbone.js. A short and to-the-point tutorial is at www.plexical.com/blog/2010/11/18/backbone-js-tutorial.

A more involved example by Alex Rothenberg, who packaged up this work as a Ruby gem, is at www.alexrothenberg.com/2011/02/11/backbone.js-makes-building-javascript-applications-fun.html.

Finally, an excellent two-part tutorial on Backbone.js is available at liquidmedia.ca/blog/2011/01/backbone-js-part-1 and liquidmedia.ca/blog/2011/01/an-intro-to-backbone-js-part-2-controllers-and-views.



DAVE TAYLOR

More Fun with Days and Dates

Figuring out how to calculate the year for a given date and day of week is a task that's not as easy as it sounds.

I received a very interesting note from a reader—a note that referred to a very interesting problem:

Many UNIX commands (for example, `last`) and log files show brain-dead date strings, such as “Thu Feb 24”. Does anybody out there have a script that will convert that to a year, given a five-year interval and defaulting to the present?

Given a day of the week, a month and a day, is it possible to calculate quickly the most recent year in the past when that particular date occurred on that day of the week? Of course it is!

Various formulas exist for calculating this sort of thing, but I realized pretty quickly that the handy `cal` utility can do the work for us. If you haven't experimented with it, you'll be surprised at what it can do. Here are two quick, relevant examples:

```
$ cal
      March 2011
Su Mo Tu We Th Fr Sa
      1  2  3  4  5
6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

```
$ cal mar 2007
      March 2007
Su Mo Tu We Th Fr Sa
      1  2  3
4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

Any light bulb starting to glow for you? If you know the month and day, you simply can go backward looking at that month's day-of-week layout until finally you find a match.

In a rudimentary fashion, the basic idea can be illustrated with a loop, like this:

```
repeat
  cal $month $year | grep $day
```

```
if day-of-week matches
  echo date $month $day most recently occurred in $year
else
  year=$(( $year - 1 ))
end repeat
```

Of course, the problem is a bit more complicated (as they always are), partially because of the complexity of calculating what day a specific date occurs in the `cal` output. There's another complication too, however; the requested date actually might have occurred in the current year, so it's not as simple as starting with the year 2010 and going backward.

Normalizing Data

The first task is to figure out how to get the information from the user. We'll have only three input parameters and do relatively little testing for misspelled day names and so on:

```
if [ $# -ne 3 ] ; then
  echo "Usage: $(basename $0) weekday month day"
  echo "  (example: $(basename $0) wed aug 3 )"
  exit 1
fi
```

That's straightforward and pretty typical, offering a nice usage tip if you forget how to use the script. As is typical of scripts, we return a nonzero result upon error too.

We can't work with completely arbitrary data, however, so when we grab the first few parameters, we'll transliterate them into lowercase and chop off all but the first three letters:

```
weekday=$(echo $1 | tr '[:upper:]' '[:lower:]'; | cut -c1-3)
month=$(echo $2 | tr '[:upper:]' '[:lower:]'; | cut -c1-3)
day=$3
```

Given “Monday, February 8”, it'd be converted automatically to “mon” and “feb” for subsequent testing.

The Current Date

We also need the current date fields for testing, and to do this, I'll demonstrate a very neat trick of date that makes this incredibly efficient:

```
eval $(date "+thismonth=%m; thisday=%d; thisyear=%Y")
```

The eval function interprets its argument as if it were a direct script command. More interesting, date can output arbitrary formats (as documented in strftime if you want to read the man page) by using the + prefix, with %m the month number, %d the day of the month and %Y the year. The result is that date creates the string:

```
thismonth=03; thisday=01; thisyear=2011
```

which then is interpreted by the shell to create and instantiate the three named variables. Neat, eh?

It turns out that users can specify a month by name or number on the command line. If it's already a number, it'll survive the transforms intact. If it's a name though, we also need the number, so we can figure out whether the date specified could be earlier this year. There are several ways to do this, including a case statement, but that's a lot of work. Instead, I'll lean on sed as I quite frequently do:

```
monthnum=$(echo $month | sed  
's/jan/1/;s/feb/2/;s/mar/3/;s/apr/4/;s/may/5/;s/jun/  
➥6/;s/jul/7/;s/aug/8/;s/sep/9/;s/oct/10/;s/  
➥nov/11/;s/dec/12/' )
```

Here's where a misspelled month name is a problem, but that's a challenge beyond the scope of this script. For now, however, we'll just roll with it.

Could the Date Occur in the Current Year?

The next set of tests is one I rewrote a couple times to ensure that I wasn't tripping myself up, because my first thought simply was to use a test like this:

```
if [ $month -le $thismonth -a $day -le $thisday ]
```

But, then I realized that in edge cases it wouldn't actually work properly. For example, let's say it's April 4 and you're checking for March 11. The month test succeeds, but the day test fails—not what we want. Instead, let's use a cascading set of conditional tests:

```
if [ $monthnum -gt $thismonth ] ; then  
    # month is in the future, can't be this year  
    mostrecent=$(( $thisyear - 1 ))  
elif [ $monthnum -eq $thismonth -a $day -gt $thisday ] ; then  
    # right month, but seeking a date in the future  
    mostrecent=$(( $thisyear - 1 ))  
else  
    mostrecent=$thisyear  
fi
```

With just this much code, we can at least test the normalization of data input and comparison tool. I ran this set of tests on March 1, by the way:

```
$ whatyear.sh Monday Aug 3  
Decided that for 8/3 we're looking at year 2010  
$ sh whatyear.sh mon jan 9  
Decided that for 1/9 we're looking at year 2011  
$ whatyear.sh mon mar 1  
Decided that for 3/1 we're looking at year 2011  
$ whatyear.sh mon mar 2  
Decided that for 3/2 we're looking at year 2010
```

It correctly identified that the current date could be a match, but that the subsequent day (mar 2) had to be in the previous year for it to be a possibility.

Good. Next month, we'll put the rest of the LEGO pieces in the model and have a working script. The big task left? Parsing the output of cal to figure out the day of the week for a given date. ■

Dave Taylor has been hacking shell scripts for a really long time, 30 years. He's the author of the popular *Wicked Cool Shell Scripts* and can be found on Twitter as @DaveTaylor and more generally at www.DaveTaylorOnline.com.

Low Cost Panel PC *PDX-057T*

- Vortex86DX 1GHz Fanless CPU
- 5.7" VGA LCD with Touchscreen
- 1RS232/422/485 serial port
- Mini-PCI Expansion slot
- 2 USB 2.0 Host Ports
- 10/100 BaseT Ethernet & Audio
- PS/2 mouse & keyboard
- CompactFlash & MicroSD card sockets
- Resolution/Colors: 640x480 @ 256K
- Low Power Consumption
- Linux, Embedded XP or Windows CE OS Options
- Wide Input Voltage and Wireless Options



2.6 KERNEL

Setting up a Panel PC can be a *Puzzling* experience. However, the PDX-057T comes ready to run with Linux Operating System installed on flash disk. Just apply power and watch the Linux X-Windows desktop User Interface appear on the vivid color LCD. Interact with the PDX-057T using the responsive integrated touchscreen. Everything works out of the box, allowing you to concentrate on your application rather than building and configuring device drivers. Just Write-It and Run-It... Starting at \$440 Qty 1.

For more info visit: www.emacinc.com/panel_pc/pdx089.htm

Since 1985
OVER
24
YEARS OF
SINGLE BOARD
SOLUTIONS

EMAC, Inc.
EQUIPMENT MONITOR AND CONTROL

Phone: (618) 529-4525 • Fax: (618) 457-0110 • www.emacinc.com



MICK BAUER

DNS Cache Poisoning, Part II: DNSSEC Validation

Configure your DNS server to check zone signatures using DNSSEC.

Last month, I described, in detail, the problem of DNS cache poisoning and why it's fundamentally changed our understanding of DNS security. Whereas previously it seemed good enough to keep one's DNS server patched and limit the hosts for which it performed recursive queries and zone transfers, we now have no choice but to pay attention to the authenticity of DNS data that our local resolvers and recursing DNS servers receive from other servers.

This is because the way DNS recursion works makes it too easy for an attacker to trigger events that lead directly to that attacker's injecting forged DNS data into a recursing server's cache, resulting in *all* users who rely on that server being redirected to impostor "evil twin" sites for specific e-commerce and on-line banking sites, or malicious, malware-spreading Web sites and so forth.

I concluded last month by explaining that although the short-term fix to Kaminsky's cache-poisoning attack is to patch DNS software so that recursing servers randomize their source UDP ports for DNS queries, this only makes the attack take longer (albeit, much longer); it doesn't eliminate it as a threat. The best protection, rather, is for administrators of authoritative DNS servers to sign all their zone data cryptographically, and for administrators of recursing or caching DNS servers to configure their servers to check the signatures of *all* signed zones they come across.

All of this signing/validating functionality is achieved by way of DNSSEC, a set of extensions to the DNS protocol. Most modern DNS server software packages now support DNSSEC (with djbdns as the most notable exception).

This month, I explain how to configure your recursing/caching DNS server to check DNS zone data signatures. Because the Internet Software Consortium's BIND package is by far the most popular DNS server application for UNIX and UNIX-like systems, my examples all involve BIND.

If you administer your own DNS zones, you also should sign your own zones and publish your certificates and signatures, but that's out of scope for this article. (See Resources for links to other DNSSEC information and tutorials. I may cover zone signing and DNSSEC key management in a future column as well.)

Note that my preferred Linux server distribution

nowadays is Ubuntu Server 10.10, so my examples all apply directly to Ubuntu and other Debian derivatives. If you run some other distribution, my examples still should be useful, because the only peculiar thing Ubuntu and Debian do in how they package BIND is to break up its configuration file (named.conf) into several parts (named.conf.options, named.conf.local and named.conf.default-zones) that are read into named.conf via "include" statements.

DNSSEC Overview

Mainly what I want you to get out of this article is how to enable DNSSEC validation on your BIND-based nameserver. I probably could fill most of this space with an overview of what DNSSEC is, how it works and so forth, but in the interest of conciseness, I give the low-attention-span version instead.

DNSSEC is a Public Key Infrastructure (PKI) for DNS zone data. When a zone administrator digitally signs all of the different types of Resource Records (RRs) in a given zone, and publishes those signatures and the zone's signing key's public certificate, it then becomes possible for any recursing nameserver that makes queries against that zone to validate those signatures and, therefore, to have cryptographic proof that the answer to a given DNS query hasn't been forged or tampered with.

This probably doesn't sound simple to begin with, but in practice, it's much more complicated even than that. This is because DNS is both hierarchical and distributed, with a "root" zone at the top and individual hostnames and other Resource Records at the bottom. In between are layers of zones and subzones.

Consider the top-level domain (TLD) .us as an example. It consists of more than 50 subzones, each representing a different state or protectorate in the United States of America—for example, mn.us for Minnesota, wi.us for Wisconsin and so forth. Within each "state" subzone there can be hundreds of sub-subzones representing cities, counties, state or municipal government agencies and so forth. lib.mn.us, for example, is used by public libraries in the state of Minnesota, and stpaul.lib.mn.us is used by the Saint Paul Public Library system.

Suppose I'm the DNS administrator for mycowtown.lib.mn.us, and I sign all of the records in

that zone and publish the corresponding RRSIG, DNSKEY and other related records. How praiseworthy of me!

However, if someone tries to resolve names in my domain, for example, interwebs.mycowtown.lib.mn.us, they'll speak to as many as *four* other nameservers before they make it all the way down the hierarchy to my beautiful, signed zone—that is, the respective authoritative nameservers for “.” (the root zone), .us, .mn.us and .lib.mn.us. What's to stop someone from tampering with the answer to one of those *prior*, recursive queries? (Who's authoritative for .us? Who's authoritative for .mn.us? Who's authoritative for .lib.mn.us?)

Obviously, there has to be a “chain of validation” all the way from the zone I really want to validate (mycowtown.lib.mn.us), all the way up to the root domain. As it happens, “.” root is signed, and I'll show you how to download and verify the initial root key shortly. So are .us and .mn.us. However, .lib.mn.us isn't yet signed (at the time of this writing). Does that mean it's pointless to sign zones below that?

Not at all. The Internet Software Consortium, creators and maintainers of BIND, maintain a DNS Look-aside Validation (DLV) database of keys for zones having precisely this sort of gap in their chains of validation. If I sign mycowtown.lib.mn.us and register my key-signing key with dlv.isc.org, resolving nameservers that are configured to use DLV still will be able to construct a complete enough chain of validation by seeing that isc.org vouches for the validity of my key-signing key, which actually is used to sign the keys (zone-signing keys) with which I actually sign zone data.

In recent versions of BIND, DNS Look-aside Validation is not only enabled by default, but preconfigured as well.

There's just one more DNSSEC mechanism I should describe before diving in to nameserver configuration, and that's automated key management. I alluded to there being two kinds of DNS keys: key-signing keys (KSKs) and zone-signing keys (ZSKs). Both types of keys must be regenerated periodically—every few months in the case of ZSKs, with which you actually sign zone data (although in my opinion, the need to do so does not speak well of how securely PKI is implemented in DNSSEC). Naturally, every time you change a KSK or ZSK, you must re-sign your entire zone.

Saying that this makes zone-signature management a bit of a headache is a gross understatement; however, there are various ways to automate this process. Luckily, right now you and I are concerned only with validating keys, not maintaining them, and recent versions of BIND 9 have a mechanism for automatically checking and updating a caching nameserver's cache of DNS keys.

This is the managed-keys{} statement in named.conf, which can be used in lieu of the static trusted-keys{} definition. When you set up BIND with the root zone's signing key, you'll do so using a managed-keys{} statement that specifies an “initial” key that is itself not a KSK or ZSK, but is used in a transparent, cryptographic transaction in which your nameserver queries a root zone authority for a copy of its current public ZSK and caches the answer it receives.

But I'm getting a little ahead of myself. Let's set up a caching nameserver and then enable DNSSEC validation on it.

Setting Up a Caching Server

If you already have a caching-only nameserver (or a general-purpose nameserver that also caches), and you need to know only how to set



Linux - FreeBSD -OpenSolaris - etc.

Proven Technology.

Proven Reliability.

When you can't afford to take chances with your business data or productivity, rely on a Genstor server customized to your specifications.

POWER

PERFORMANCE

Fly into the Cloud with Genstor Systems



- Up to 48 cores in a 1U.
- AMD Opteron 6100 series.
- Single high-efficiency power supply.
- Up to 512GB DDR3 memory.
- Ideal as front end processing servers.



- Up to 12 cores in a 2U
- Dual redundant high efficiency power.
- Up to 96GB DDR3 memory.
- Server Power Capping via Intel Intelligent Power Node Manager.
- Ideal as front end processing and/or storage.



- Up to 4 GPU cards.
- Dual redundant high efficiency power.
- Up to 192GB DDR3 memory
- Up to 24 cores using 2 CPUs.
- Up to 8 3.5" disks.

Genstor Systems, Inc.



Powerful.
Intelligent.

1501 Space Park Drive
Santa Clara, CA 95054

www.genstor.com

E-mail: sales@genstor.com
Phone: 877-25 SERVER
408-980-0121

Intel®, the Intel® logo, Intel® Xeon®, and Xeon® Inside® are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Listing 1. Default Ubuntu/Debian named.conf.options File

```
options {
    directory "/var/cache/bind";
    auth-nxdomain no;      # conform to RFC1035
    listen-on-v6 { any; };
};
```

Listing 2. named.conf.local

```
acl mynetworks { 192.168.100.0/24; 10.10.0.0/16; };
options {
    directory "/var/cache/bind";
    allow-query { mynetworks; };
    auth-nxdomain no;      # conform to RFC1035
    listen-on-v6 { none; };
};
```

up DNSSEC validation, you can skip ahead to the Setting Up DNSSEC Validation section below. In the interest of completeness, however, and for the purpose of pointing out a few default settings it's good to change, here's a quick procedure on setting one up.

First, install whatever (sub-)version of BIND9 your distribution supports. At the time of this writing, Ubuntu 10.10 includes BIND version 9.7.1; to install it, use this command:

```
sudo apt-get install bind9 bind9utils
```

The bind9 package provides BIND 9.7.1 itself in the form of the daemon named, plus its configuration files, man pages and libraries. The bind9utils package provides handy commands, such as rndc and named-checkconf, and the DNSSEC commands dnssec-keygen and dnssec-signzone. Those last two are used only for creating and maintaining actual DNSSEC zone keys and signatures, respectively. You won't actually need those if all you're doing with DNSSEC is validating signatures from other zones.

On Debian and Ubuntu systems, the bind9 package places its configuration files in /etc/bind. The files we're concerned with here are /etc/bind/named.conf, /etc/bind/named.conf.options and /etc/bind/bind.keys.

Actually, of those three files, we'll edit only one, named.conf.options. I mention the other two in order to point out that named.conf uses "include" statements to pull content from /etc/bind/named.conf.options, /etc/bind/named.conf.local (which contains your local zone files) and /etc/bind/named.conf.default-zones (which contains default zone information for local loopback interfaces).

At this point, I have good news for you: your Debian or Ubuntu system's named.conf.options file is, technically,

already set up to run named as a caching-only nameserver. The bad news is, it needs to be tightened up a bit before you can consider it to be a *secure* caching nameserver.

Listing 1 shows the default Debian/Ubuntu named.conf.options file (with comment lines omitted).

Listing 2, in contrast, is much more secure.

Let's discuss why Listing 2 is better. First, I've defined an Access Control List (ACL) that specifies two IP networks in "CIDR notation". Technically, this is not an option, but it needs to be loaded before any option statements, so it needs to go either here or in named.conf prior to this line:

```
include "/etc/bind/named.conf.options";
```

In and of itself, this acl doesn't do anything. But once it's defined, I can create an "allow-query" option that refers to it, and as you can see in Listing 2, that's exactly what I've done. Obviously, in adapting this file for your own use, you should replace the list in my acl statement ("192.168.100.0/24; 10.10.0.0/16;") with a list of your organization's local IP subnets.

The other security tweak I've made is to change the value for the "listen-on-v6" option from "any" to "none". Because none of my local subnets use IPv6, there's no reason to listen on any local IPv6 interfaces. Technically, this shouldn't matter if I don't even have any IPv6 interface attached to my server and if I've set an acl and specified it in an allow-query statement. So, maybe I'm just being paranoid by turning off IPv6 altogether here, but turning off unused features is nearly always a good thing to do.

Once you've edited and saved your /etc/bind/named.conf.options file, you can check your work by running the named-checkconf command with no arguments, like so:

```
bash-$ sudo named-checkconf
```

Assuming that doesn't return any configuration errors (I have a tendency to misplace or omit semicolons, myself), you then can make your running named process reload its configuration and zone files using the rndc command, like this:

```
bash-$ sudo rndc reload
```

Now, you can test your server by logging on to some other host on your network and running a dig query or two against it. For example, if my caching nameserver's IP address is 192.168.100.253, I can have it look up DNS information for www.linuxjournal.com like so:

```
mick@someotherhost:/home/mick$ dig @192.168.100.253
-> www.linuxjournal.com
```

You can, of course, simply configure your client

system to use your caching nameserver as its default nameserver, in which case you can omit the @192.168.100.253 in the above command. But, you probably don't want to do that until you're sure it works.

If it doesn't work, make sure your client system's IP address falls into one of the IP networks you specified in any acls you've set in /etc/bind/named.conf.options, as I described earlier.

At this point, your caching-only nameserver is up and working properly. Now you can configure DNSSEC validation.

Setting Up DNSSEC Validation

Back on your caching nameserver, all you need to do to add DNSSEC validation is add three lines in the options{} section of your /etc/bind/named.conf.options file, plus a new managed-keys{} section, as shown in Listing 3.

The first two new lines in Listing 3, dnssec-enable yes; and dnssec-validation yes;, enable DNSSEC on your caching nameserver. This is actually a redundant setting, because "yes" is the default value for both these settings in BIND versions 9.5 and later, but it doesn't hurt to specify them.

The third new line, dnssec-lookaside auto;,

Listing 3. DNSSEC-Enabled named.conf.local

```
acl mynetworks { 192.168.100.0/24; 10.10.0.0/16; };

options {
    directory "/var/cache/bind";
    allow-query { mynetworks; };
    auth-nxdomain no;      # conform to RFC1035
    listen-on-v6 { none; };
    dnssec-enable yes;
    dnssec-validation yes;
    dnssec-lookaside auto;
};

managed-keys {
    ." initial-key 257 3 8 "
    AwEAAagAIKlVZrpC6Ia7gEzahOR+9W29euxhJhVVLOyQ
    bSEW008gcCjFFVQUTf6v58fLjwBd0YI0EzrAcQqBGCzh
    /R5tIo08g0NfnfL2MTJRKxoXbfDaUeVPQuYEhg37NZWA
    JQ9VnMVDxP/VHL496M/QZxkjf5/Efucp2gaDX6RS6CxP
    oY68LsvPVjR0ZSwzz1apAzvN9dlzEheX7ICJBtuA6G3
    LQpzW5h0A2hzCTMjJPJ8LbqF6dsV6DoBQzgul0sGIcG0
    Y17OyQdxFz57reLSQageu+ipAdTTJ25AsRTAoub8ONGc
    LmqrAmRLKBP1dfwhYB4N7knNnulqQxA+Uk1ihz0= ";
};
```



VESA-Mountable NVIDIA® ION/ION2 System

Fully configured with HDD, RAM. Features GeForce® Graphics, flexible storage options, and Wi-Fi.

Fanless, Rugged Intel® Core™ Platform

No fans, no moving parts. Just quiet, reliable performance. Full featured; no compromises.

Chris
Assembly Team Manager

**Assembled & tested
Ubuntu Linux compatible
systems...**

Genuine expertise.

www.logicsupply.com/linux

LOGIC
SUPPLY®

tells BIND/named to use DLV automatically any time it can't validate a complete chain of trust from a given Resource Record all the way from root (.) downward. See the DNSSEC Overview section earlier in this article if you've forgotten how DLV works.

As I mentioned in that section, recent versions of BIND are preconfigured to find isc.org's DLV repositories. All you have to do to take advantage of this is set "dnssec-lookaside" to "auto", and BIND will do the rest. As more and more TLDs are signed, this feature will become less important.

And, that brings me to the last new element in the named.conf.options file: the managed-keys{} section. This specifies a key for the DNS "root" domain, which is the top of any chain of DNSSEC trust.

You don't necessarily need to specify any keys "lower" in the DNS hierarchy than root; if you start out knowing the root key, you can trust signed replies from root nameservers. That trust flows downward to signed data from TLDs (.gov, .us, .net and so on) and so forth. "Gaps" in the downward

Listing 4. dig Output for Successful DNSSEC Validation

```
; <>> DiG 9.6.0-APPLE-P2 <>> @192.168.100.253 www.isc.org +dnssec
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 62704
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 5, ADDITIONAL: 13
```

Resources

DNSSEC—the DNS Security Extensions—Protocol Home Page: www.dnssec.net

Alan Clegg's "DNSSEC—Living and Loving Life after Kaminsky; Or: How I overcame my fear and signed my zones." Presentation to REN-ISAC on 10-30-2008: www.ren-isac.net/techburst/hardcopy/ren-isac_techburst_20081030_clegg_dnssec.pdf

Geoff Huston's "A Fundamental Look at DNSSEC, Deployment, and DNS Security Extensions": www.circleid.com/posts/dnssec_deployment_and_dns_security_extensions

Ubuntu 10.10 Server Guide: "Chapter 7. Domain Name System (DNS)": <https://help.ubuntu.com/10.10/serverguide/C/dns.html>

BIND 9.7 Administrator's Reference Manual (ARM): ftp.isc.org/isc/bind9/cur/9.7/doc/arm/Bv9ARM.pdf

Tony Finch's "How to set up DNSSEC validation with BIND-9.7": fanf.livejournal.com/107310.html

chain of validation hopefully will be handled by DLV.

For heaven's sake, do not simply copy Listing 3's key entry for "." verbatim! Tony Finch has written a quick-and-easy procedure on checking and validating the (initial) root certificate (see Resources). Summarized, this procedure consists of the following steps.

- 1) Use the following dig command to obtain the current root certificate and save it to the file root-dnskey:

```
bash-$ dig +multi +noall +answer DNSKEY . >root-dnskey
```

- 2) Create a hash of this certificate and save it to the file root-ds with this command:

```
bash-$ $ dnssec-dsfromkey -f root-dnskey . >root-ds
```

- 3) Pull the official root certificate's hash from <https://data.iana.org/root-anchors/root-anchors.xml>, and compare it to the root-ds file you just created. For extra paranoia, you can use PGP to check the signature of root-anchors.xml (see Tony Finch's article).

- 4) If the hashes match, copy the key (the long one, number 257) from root-dnskey into your managed-keys statement, as shown in Listing 3. The first line of this block (after the managed-keys { line) should be the same as in Listing 3.

As with your previous changes, after you save named.conf.options, you should check it with named-checkconf, and then load it with rndc reload.

Finally, to test DNSSEC validation, test some known-signed record, such as www.isc.org, using dig. Be sure to use the +dnssec flag, like this:

```
mick@someotherhost:/home/mick$ dig @192.168.100.253
->www.isc.org +dnssec
```

If everything is working, dig's output should indicate that the "ad" (authenticated data) flag is set. Listing 4 shows the first part of what a successful reply to our example dig command would look like. Note the line that begins ;; flags: qr rd ra ad;.

Conclusion

And with that, your nameserver is successfully validating signed zone data! For now, I wish you thanks and goodbye. As I seem to do every couple years, I'm going to take a hiatus for a few months. I do plan on resuming the Paranoid Penguin after that, however, refreshed and renewed for your reading pleasure.

Until then, take care of yourself and especially your Linux systems! ■

Mick Bauer (darth.elmo@wiremonkeys.org) is Network Security Architect for one of the US's largest banks. He is the author of the O'Reilly book *Linux Server Security*, 2nd edition (formerly called *Building Secure Servers With Linux*), an occasional presenter at information security conferences and composer of the "Network Engineering Polka".

Save the Date!

2011 USENIX Federated Conferences Week

June 14–17, 2011, Portland, OR

www.usenix.org/fedweek11

USENIX ATC '11

2011 USENIX Annual
Technical Conference

Wednesday–Friday, June 15–17

www.usenix.org/atc11

**REGISTRATION
DISCOUNTS
AVAILABLE!**

WIOV '11

3rd Workshop on I/O
Virtualization

Tuesday, June 14

www.usenix.org/wiov11

WebApps '11

2nd USENIX Conference on Web
Application Development

Wednesday–Thursday, June 15–16

www.usenix.org/webapps11

HotCloud '11

3rd USENIX Workshop on
Hot Topics in Cloud Computing

Tuesday, June 14

www.usenix.org/hotcloud11

HotStorage '11

3rd USENIX Workshop on
Hot Topics in Storage and
File Systems

Tuesday, June 14

www.usenix.org/hotstorage11

And more!

Registration will be open in early April. Register by the Early Bird Registration Deadline, Monday, May 23, and receive the greatest savings. Visit www.usenix.org/fedweek11 for new workshop announcements and registration information.

Stay Connected...

use
nix

www.usenix.org



<http://www.usenix.org/facebook>



<http://www.usenix.org/linkedin>



<http://twitter.com/usenix>



<http://blogs.usenix.org>



KYLE RANKIN

Lightning Hacks—the Command Next Door

Even old commands (and old sysadmins) can learn new tricks.

Every year or so, I like to write a column I title "Lightning Hacks". This column is inspired by the lightning talks you encounter at most conferences. In a lightning talk, instead of having one speaker give a 60-minute presentation, multiple speakers give short 5–10-minute presentations. By the end of a lightning talk, you end up hearing about all sorts of cool topics that wouldn't have gotten their own time slot. In this column, I get a chance to talk about a few cool "hacks" I've run across that by themselves wouldn't fill an entire column.

In past Lightning Hacks columns, I've covered a wide variety of different topics, and in the previous edition, I focused strictly on hacks with the ssh command. In this column, however, I showcase a few interesting tricks with commands you might use every day. It's easy to take your favorite programs for granted, but I've found that no matter how long I use Linux, it seems I'm always picking up new time-saving tips, and a Lightning Hacks column is as good a place as any to list a few of my favorites.

New Ways to Escape

It isn't news to frequent readers of this column that I edit all of my text files with vim. I've been using vim for so long, my muscle memory for the Esc key is second nature. When you press the Esc key in vim, you return back to the default navigation mode from whatever mode you were in. When I'm at all unsure where I am or what mode I'm in, my brain automatically has my hand bang on that Esc key like I'm a champion *Hungry Hungry Hippos* player. Of course, on a modern keyboard, the Esc key is quite a way from home row, and what's worse, on some keyboards (like on ThinkPads), the Esc key is in a different place, so someone like me hits F1 half the time. It turns out, however, that Ctrl-[functions just like Esc in vim (and other programs that accept vi-style syntax, like vimperator). Both of those keys are just a pinkie-reach away and keep most of your fingers on home row. I have to say though, old habits die hard. I had to put a post-it note on my monitor that read "Ctrl-[" for a few weeks before my fingers started to catch the hint. It takes only a few weeks for the habit to form though, and before you know it, you'll be shaving that extra millisecond off your typing time.

Now that I've removed yet another function from the poor Esc key, let's give it something else to do. When I

first set up my N900 with terminal sessions, I ran into a problem a lot of portable computer users face: half the special keys aren't on the tiny keyboard. In my case, this presented quite a problem, because I am a heavy Irssi user and my keyboard had no way to press Alt even in the special symbols menu. Without the Alt key, it was a pain to switch between different Irssi windows. I wasn't sure what to do until I discovered that in most terminal sessions, the Esc key can substitute for Alt. It turned out my terminal emulator had an Esc touchscreen key-mapped, so from that point on, I just pressed Esc whenever I needed Alt. It also turns out this works the same way on all of my laptop terminal sessions.

Open Vim to a Specific Line Number

This is a quick-and-simple tip that I can't believe took me so long to discover. Quite often, you will be in a situation where you want to open a file to a specific line number. A few good examples include when you re-kickstart a server, go to ssh in to it and realize the host keys have changed. The error message lists the line number with the conflicting key, so it is easy to go in and erase it. Another common example occurs when you are working on a script or configuration file and see a reference to a syntax error on a specific line.

Back in the day, when I wanted to go to a specific line in vim, I would open the file and press :<number><Enter> to go to the line, but if you want to save a step, just add +<number> to the command line as an argument when you execute vim. For instance, if I want to jump ahead to line 27 in my `~/.ssh/known_hosts` file, I would type:

```
vim ~/.ssh/known_hosts +27
```

Filter Grep with Grep

Grep is one of the crucial tools you should master if you want to spend any time on the command line. One of the most common ways I use grep is to check whether a certain process is running and see its process ID. For instance, if I want to check for the process ID of my cron dæmon, I might type:

```
$ ps -ef | grep cron
root      1215      1  0 Feb23 ?          00:00:00 cron
greenfly  1252  1976  0 20:57 pts/0        00:00:00 grep cron
```

In this example, cron has a process ID of 1215. The

annoying thing though is that because I had cron in my grep command, it also showed up in the output. This means every time I run this command, I have to be sure to ignore that line in the output. Of course, I also could use grep to filter itself:

```
$ ps -ef | grep cron | grep -v grep  
root      1215      1  0 Feb23 ?      00:00:00 cron
```

That works; however, it's unnecessary. If I enclose one of the characters in my first grep filter in brackets, it does the filtering for me:

```
$ ps -ef | grep [c]ron  
root      1215      1  0 Feb23 ?      00:00:00 cron
```

This works because when I enclose c in brackets, it turns it into a character class that contains only one character, c. However, the grep command itself shows up in the process list with those brackets, so it doesn't match my [c]ron pattern anymore.

Hear Your Pings

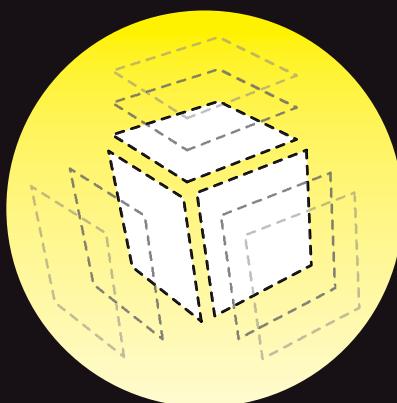
The problem with powerful server hardware is that it

always takes a little time to reboot. Often when I have to reboot a server, I want to know the moment it comes back up, because I want to log back in and finish some work. Usually when I reboot a machine, I start a ping process, so I can watch to see when it is available. The problem is that servers seem to take just long enough to come back up that I end up getting distracted in another window on my desktop and don't see the ping replies. My solution to this is to make ping audible:

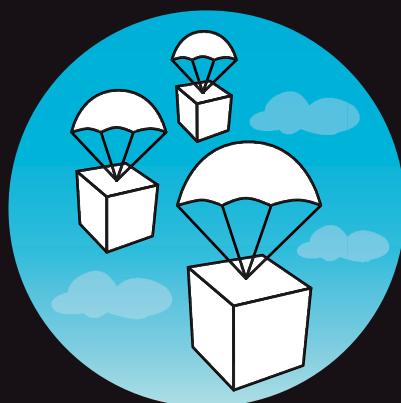
```
$ ping -a 10.0.1.7
```

When you pass the -a command to ping, it generates an audible bell for every ping reply. I get to hear a few system beeps as the server goes down, and then I hear only silence. If I do become distracted and the server comes back, I immediately start hearing this annoying system beep in my ear and know the server is ready.■

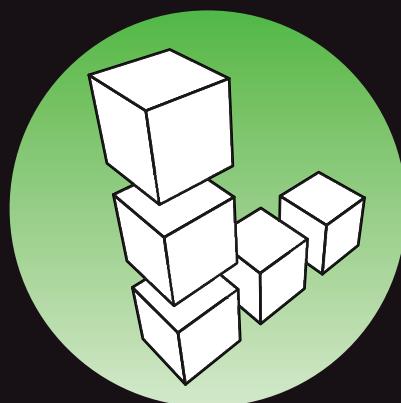
Kyle Rankin is a Systems Architect in the San Francisco Bay Area and the author of a number of books, including *The Official Ubuntu Server Book*, *Knoppix Hacks* and *Ubuntu Hacks*. He is currently the president of the North Bay Linux Users' Group.



Develop.



Deploy.



Scale.

Full root access on your own virtual server for as little as \$19.95/mo

Multiple Linux distributions to choose from • Web-based deployment • Five geographically diverse data centers • Dedicated IP address • Premium bandwidth providers • 4 core SMP Xen instances • Out of band console access • Private back-end network for clustering • IP fail-over support for high availability • Easily upgrade or add additional Linodes • Free managed DNS

For more information visit www.linode.com or call us at **609-593-7103**

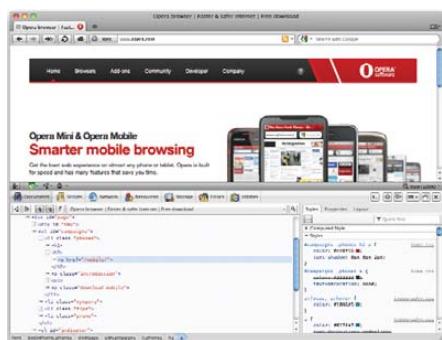
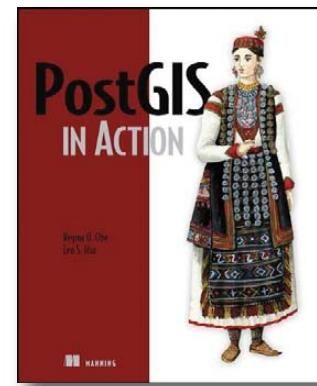


linode.com

Regina Obe and Leo Hsu's *PostGIS in Action* (Manning)

The hardworking community of developers and users of open-source geographic information systems (GISes) is sorely underserved, so it's fortunate to see the new Manning publication *PostGIS in Action* by Regina Obe and Leo Hsu. *PostGIS in Action* is the first book devoted entirely to PostGIS, a freely available open-source spatial database extender, which can answer questions beyond those possible with a mere relational database. PostGIS' feature set equals or surpasses proprietary alternatives, allowing for the creation of location-aware queries and features with just a few lines of SQL code. Readers with experience in relational databases will find a background in vector-based GIS that enables quick ramp-up to analyzing, viewing and mapping data. The advanced will learn how to optimize queries for maximum speed, simplify geometries for greater efficiency and create custom functions suited specifically to their applications.

www.manning.com/obe



Opera Dragonfly

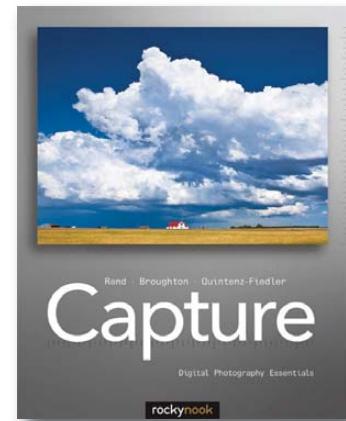
Opera Software's Opera Dragonfly is a new suite of open-source debugging tools for Web developers and designers that got its name because "it eats bugs". The suite covers the full debugging work flow, from inspecting network access and downloaded resources to correcting JavaScript issues and seeing how CSS rules apply to the DOM. Opera Dragonfly supports all the newest Web technologies, including SVG and HTML5 APIs, such as Web Storage. Product benefits, sayeth Opera, include a superior JavaScript debugger, a network inspector to discover why a site "turns to molasses" and a storage inspector to uncover how a site handles the data it collects. Opera Dragonfly loads automatically when one downloads the Opera browser.

www.opera.com

Glenn Rand, Chris Broughton and Amanda Quintenz-Fiedler's *Capture* (Rocky Nook)

The advent of the digital camera has truly transformed photography and made it more accessible to all, especially us geeks. Although we geeks might be good at manipulating images with The GIMP and organizing them with digiKam, we may not be proficient at the mechanics of good exposure, a requirement that has not changed with digital photography. Enter the new book *Capture: Digital Photography Essentials*, written by Glenn Rand, Chris Broughton and Amanda Quintenz-Fiedler. The text addresses both the opportunities and limitations of digital photography, and how to work with those opportunities and around the limitations. Readers will learn to maximize the potential of their images through an understanding of the core principles and more advanced aspects of the digital photographic process.

www.rockynook.com



Cloud.com's CloudStack

One of your more direct routes to the cloud is by hopping a ride onto Cloud.com's CloudStack open-source cloud computing platform, now in version 2.2. CloudStack, says Cloud.com, is a comprehensive, open-source software solution that accelerates the deployment, management and configuration of highly scalable, public or private infrastructure as a service clouds. Data-center operators can build cloud services within their existing infrastructure to offer on-demand, elastic cloud services. Version 2.2 of CloudStack offers features such as improved hypervisor support (VMware vSphere 4, Citrix XenServer 5.6 and KVM), advanced networking configuration, an AJAX Web interface and borderless scalability. The federation of managed and hosted availability zones can be managed within a single CloudStack deployment. The new CloudBridge feature enables applications to interoperate with other cloud solutions including Amazon EC2 and S3 APIs, as well as the upcoming OpenStack API. CloudStack is available for immediate download.

www.cloud.com

Logicalis Enterprise Power Cloud

Keeping our heads in the clouds, let's take note of another cloud-based platform, the Logicalis Enterprise Power Cloud for IBM Power 770 Systems. The solution is for IBM users who require more than Windows and Linux support. It provides a "data center in the sky" with all the capabilities found in on-premises data centers and more. Key benefits include support for AIX and i5/OS, enterprise-class management skills for legacy systems, flexible computing capacity, lower infrastructure and maintenance costs, elasticity to respond to changing business needs, 24x7 monitoring and management, and reduced carbon footprint and energy consumption.

www.us.logicalis.com/cloud

Libelium Meshlium Xtreme

Libelium is moving the transparently networked world forward with its newly released Meshlium Xtreme multiprotocol wireless router—a global first according to the company. Meshlium Xtreme uniquely supports five wireless standards, namely Wi-Fi, ZigBee, GPRS, Bluetooth, GPS and wireline Ethernet, giving a wide choice of methods for connecting wireless sensor networks to the Internet. The product also supports the storage of the sensor data in its internal database system as well as with external Internet servers. Novel features include dynamic Wi-Fi frequency switching, a "discover and store" application for Bluetooth, an aluminum IP67 waterproof enclosure for harsh environments, special holders for attaching to oddly shaped locations and an optional solar panel kit for locations without a power source. The management interface is open source, and the product runs on Debian Linux.

www.libelium.com/products/meshlium



Red Hat's JBoss Enterprise SOA Platform

The motto for Red Hat's JBoss Enterprise SOA Platform, newly updated to version 5.1, is "turn the data you have into information customers can use". New in version 5.1 is JBoss Enterprise Data Services Platform 5.1, an open-source data virtualization and integration platform that includes tools to create data services out of multiple data stores with different formats, presenting information to applications and business processes in an easy-to-use service. JBoss' net benefit, notes Red Hat, is that "data services become reusable assets across the enterprise and value chain, increasing return on data assets, enabling faster time-to-solution, and driving better business execution". JBoss Enterprise SOA Platform 5.1 includes Apache CXF Web services stack, JBoss Developer Studio 4.0, a technology preview of WS-BPEL, a technology preview of Apache Camel Gateway and updated certifications (Red Hat Enterprise Linux 6, Windows 2008, IBM, JDK and more).

www.jboss.com

Dassault Systèmes DraftSight

France's Dassault Systèmes says that the demand for its 2-D CAD software DraftSight to move to the Linux platform "has been overwhelming", which made the latest release on Linux inevitable. This new version of the no-cost DraftSight, which now runs on Linux, Mac OS and Windows, allows users to create, edit and view DWG files. DraftSight features a light footprint, a dynamic community-support site, professional support and education-oriented packages. The program is available for download from Dassault Systèmes' Web site.

www.3ds.com



Please send information about releases of Linux-related products to newproducts@linuxjournal.com or New Products c/o *Linux Journal*, PO Box 980985, Houston, TX 77098. Submissions are edited for length and content.

Fresh from the Labs

Tor Browser Bundle—Tor Goes Portable

<https://www.torproject.org/projects/torbrowser.html.en>

I've never covered a subproject of something I've reviewed before, but I noticed this a few weeks ago when trawling the Tor site (I've no idea how I missed it until now). It seemed so important that I instantly gave it top billing for this month's column.

Tor has become increasingly famous/infamous in the past few months due to its use by Web sites like WikiLeaks, as well as its crucial role in getting information out to the world during the recent Egyptian revolution.

For those unfamiliar with Tor, *LJ* has covered it before—see Kyle Rankin's article "Browse the Web without a Trace" in the January 2008 issue and my New Projects column in the April 2010 issue. But to recap, the Tor Web site sums it up nicely:

The Tor software protects you by bouncing your communications around a distributed network of relays run by volunteers all around the world: it prevents somebody watching your Internet connection from learning what sites you visit, it prevents the sites you visit from learning your physical location, and it lets you access sites that are blocked.

However, in standard form, Tor is a rather cumbersome beast, with all sorts of

background process daemons, complex configuration files, startup services and so on. Even if you're a pretty advanced user, there's still a good chance of something going wrong somewhere, delaying your chance to jump on-line securely. This is where the Tor Browser Bundle comes to the rescue:

The Tor Browser Bundle lets you use Tor on Windows, Mac OS X or Linux without needing to install any software. It can run off a USB Flash drive, comes with a pre-configured Web browser and is self-contained. The Tor IM Browser Bundle additionally allows instant messaging and chat over Tor.

Before I continue, the Web site offers a caveat that *LJ* readers probably will find more important than most: "Note that the Firefox in our bundles is modified from the default Firefox; we're currently working with Mozilla to see if they want us to change the name to make this clearer".

Installation Although the bundle was designed to run on a Flash drive, that needn't be the case. Like many others, I simply saved this to hard drive and ran it from there. Feel free to do the same if you're so inclined.

As for installing the bundle (well, sort of), the Tor people were good enough to offer the following instructions, saving me a lot of trouble:

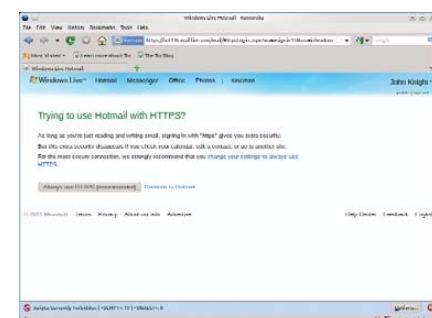
Download the architecture-appropriate file above, save it somewhere, then run: tar -xvzf tor-browser-gnu-linux--dev-LANG.tar.gz (where LANG is the language listed in the filename), and either double-click on the directory or cd into it, then execute the start-tor-browser script. This launches Vidalia, and once that connects to Tor, it launches Firefox.

Usage Before continuing, this bundle is designed to run on machines that don't have Tor installed. If you do have Tor installed and running, stop the process and then you can carry on.

Now, with the Browser Bundle running,



If you get this message in big green letters, Tor's running fine!



The default no-script settings can send some Web sites haywire!

first the Vidalia control panel will start, which is designed to establish a Tor connection as well as manage various Tor options using a GUI front end. I recommend exploring the Vidalia control panel, as it has neat features, such as bandwidth monitoring, network viewer, settings dialog and more.

Provided all has gone well, Firefox should start and will try to load a Web page. This Web page takes a while to load—don't worry; the Tor network is pretty slow at the best of times, and if everything worked, you'll soon have a message that says in big green letters: "Congratulations. Your browser is configured to use Tor."

From here, you can browse like you would any other day, but the uninitiated may be in for a shock. Most modern Web sites have fancy scripts and Flash objects, and these very features are what causes the greatest security holes. Hence, Tor's browser disables these scripts by default.



Extending your options greatly, the Vidalia Control Panel is a great tool when using Tor.

Chances are that the only Web sites that will work without hassle are deliberately minimalist in their design.

However, don't worry. If you look at the screen's bottom right, you'll see an icon with a blue S. Click on that icon, and you can choose either to enable scripts for this particular Web site or enable scripts globally (not recommended for the security reasons just mentioned).

Those willing to take the risk can choose new default settings for security in the preferences, available under Edit→Preferences. Given the nature of this project, the default settings are understandably set for paranoia. If you're undertaking work that involves a serious security risk, be very careful with what you enable or disable. If you're unsure of the risk you're taking, perhaps a more secure, minimalist and less-script-reliant Web service would be a better choice for your activities (assuming an alternative is available, of course).

Something I couldn't get working under the Linux version was Flash in general. My older brother said he used Tor to watch

some overseas TV shows not available in Australia and inaccessible to those with IP addresses external to a certain country. He was using the Windows version of Tor, and I'm guessing that he would've used the Browser Bundle, instead of setting up a machine with Tor permanently installed. The content he was viewing was Flash-based, so he must have been able to enable it for such a session.

I realize that Flash presents a security risk, but many people will want to use the Tor Browser Bundle for something as trivial as watching international TV shows—not really the sort of thing that will have the authorities kicking down your front door. If any readers out there know how to get Flash running with the Linux bundle, feel free to drop me an e-mail. I'd love to hear from you!

Moving back onto more serious topics, in journalism in particular, projects such as Tor will become increasingly indispensable in moving information beyond borders and protecting user privacy against prying eyes. When I last tried Tor, it gave me a headache and was far from intuitive in its

use. However, a clever little bundle such as this gives Tor's power of anonymity to those with average PC skills, and regardless of its use, that's an important thing.

NUT—Nutrition Software

nut.sourceforge.net

If you're watching your weight, monitoring your health and dietary habits, or simply unconvinced by flashy food labels that don't tell the whole story, this is the project for you. According to the Web site:

I have written open-source free nutrition software, NUT, which records what you eat and analyzes your meals for nutrient levels in terms of the "Daily Value", or DV, which is the standard for food labeling in the US. The program uses the free food composition database from the USDA. This free nutritional analysis software was written for UNIX systems (I use Linux), but it can be compiled on just about any system with a C compiler. (To get a free C compiler,



visit us at www.siliconmechanics.com
or call us toll free at 866-352-1173



For more information about the
Rackform iServ R413
visit www.siliconmechanics.com/R413

Charles heads the web development team here at Silicon Mechanics: he's responsible for the configurators and power calculator on our site, among other things. As a software expert, he offers a useful perspective on our server and storage products.

When asked what he would do if he had a Rackform iServ R413 configured with 4 8-core Intel® Xeon® Processor 7500 Series CPUs and 32 DDR3 DIMMs, he said, "32 virtual machines . . . one per core . . . in one rack unit." But he didn't stop there.

Charles knows that to make the best use of a server with that kind of processing horsepower in a virtualized environment, he needs I/O to match. He paired the 4P server with a Storform iServ R516 storage server, configured with 24 2.5-inch Intel X25-E solid state drives. Think of it as a developer's dream team: multi-core processing and high memory counts for blistering performance, and high-performance storage for blazing I/O speed.

Need a "dream team" of your own? Talk to the Experts at Silicon Mechanics for the perfect match.

When you partner with Silicon Mechanics, you get more than just the power and performance of the latest Intel technologies—you get an Expert like Charles.



**Powerful.
Intelligent.**

Expert included.

Silicon Mechanics and the Silicon Mechanics logo are registered trademarks of Silicon Mechanics, Inc. Intel, the Intel logo, Xeon, and Xeon Inside, are trademarks or registered trademarks of Intel Corporation in the US and other countries.

Windows people might look at Cygwin or MinGW, and Mac people might look at xcode.) By experimenting with NUT, you can find the optimal level of the various nutrients and how to implement this with foods available to you. NUT can help reconstruct the lost instruction manual to your care and feeding, because, when the authorities and crackpots disagree on the proper human diet, you can design an experiment using the food composition tables to discover the truth!

Installation I'm unsure of other distributions, but binaries are available for Debian and Ubuntu. I run with the usual source option here. Grab the latest source tarball, extract it, and open a terminal in the new folder. At the time of this writing, NUT didn't have an install script, so you'll need to do a number of steps manually. Assuming the /usr/local folders are fine for installation, issue the following commands as root:

```
# mkdir /usr/local/lib/nut/
# mv raw.data/* /usr/local/lib/nut/
```

If your distro uses sudo (such as Ubuntu), simply prefix those commands with the sudo command.

Once this step is out of the way, compile the program with:

```
$ make
```

If the compiling goes well, you should be able to use the console program immediately. Simply enter the command:

```
$ ./nut
```

This runs the console program, which I look at in the next session. As for the GUI program, that needs to be compiled separately.

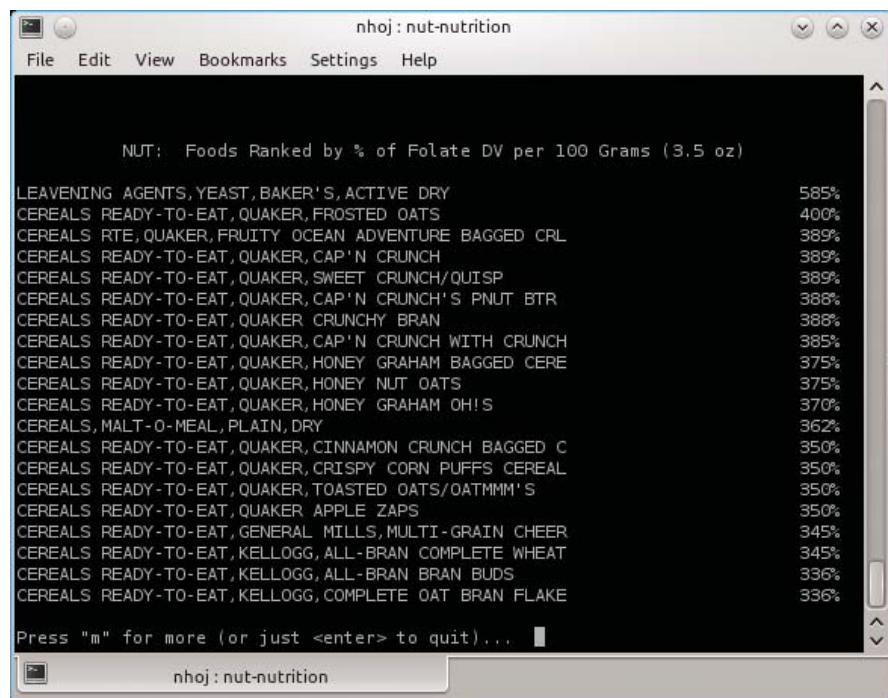
Change into the fltk directory by entering:

```
$ cd fltk
```

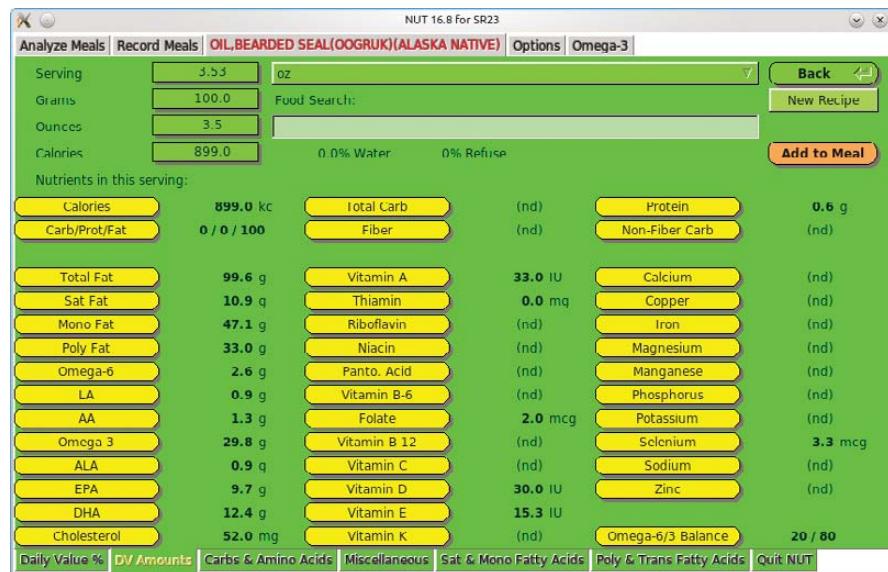
And again, enter the command:

```
$ make
```

I ran into compilation problems when I first tried to compile the fltk component



NUT has an extensive database of food statistics, worth the price of admission alone (console version pictured).



The NUT GUI makes using this program much less tiresome and displays other forms of information simultaneously. Here's the stats for bearded seal oil.

(hence, yesterday I was going to cover only the console program). I'm not sure what I did to get it working, but I think it was downloading fltk 1.3 manually from the fltk Web site, then compiling and installing it separately. If you manage to get it compiled, you can run the GUI

program now by entering:

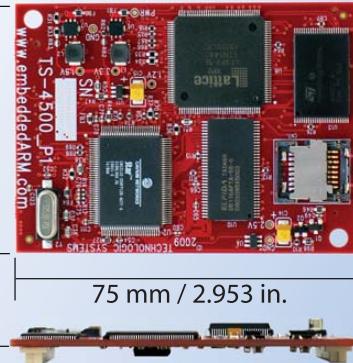
```
$ ./Nut
```

Note the capital letter above—it's the differentiator between the GUI and command-line programs.

TS-SOCKET Macrocontrollers

Jump Start Your Embedded Design

Series starts at **\$92** qty100



TS-SOCKET Macrocontrollers are CPU core modules that securely connect to a baseboard using the TS-SOCKET dual connector standard with common pin-out interface. COTS baseboards are available or design your own baseboard for a custom solution with drastically reduced design time and complexity. Start your embedded system around a TS-SOCKET Macrocontroller to reduce your overall project risk and accelerate time to market.

- TS-4200: Atmel ARM9 with super low power
- TS-4300: Cavium ARM11 with dual 600 MHz and FPU
- TS-4500: Cavium ARM9 at very low cost
- TS-4700: Marvell PXA168 with video and 1.2 GHz CPU
- TS-4800: Freescale iMX515 with video and 800 MHz CPU
- Several COTS baseboards for evaluation & development



Design your solution with one of our engineers

- Over 25 years in business
- Never discontinued a product
- Engineers on Tech Support
- Open Source Vision
- Custom baseboards w/ excellent pricing and turn-around time
- Most products ship next day

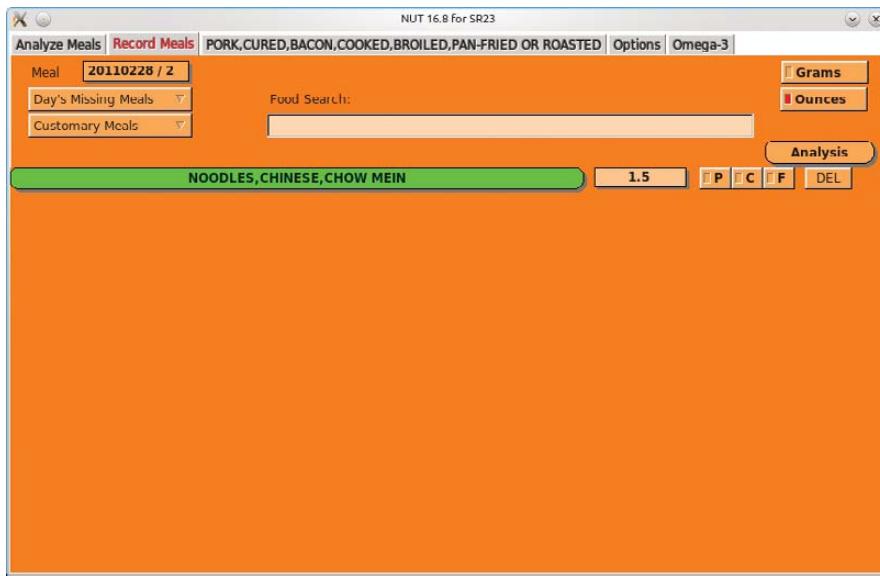


We use our stuff.

visit our TS-7800 powered website at

www.embeddedARM.com

(480) 837-5200



One of the main reasons for using NUT is recording your daily meals and then running detailed analysis against them.

If you'd like quick access to NUT, copy the executables into bin folders. If you're still in the fltk directory, change back into the main directory of the nut folder:

```
$ cd ..
```

Next, enter these commands as either root or sudo:

```
# mv nut /usr/local/bin/  
# mv nut.1 /usr/local/man/man1/  
# mv fltk/Nut /usr/local/bin
```

Now you either can run the command-line version with nut or the GUI with Nut.

Usage Unfortunately, the long installation instructions haven't left me much room to cover the actual usage of NUT, but thankfully, things are pretty simple to use.

The console version uses a series of number-driven menus to navigate between functions and foods. For instance, option 1 is for recording meals, followed immediately by a prompt for the date, the meal number and, finally, the name of the food.

Entering the name of the food needn't be precise, as NUT's main strength is its database. Long lists of premade choices exist, and each choice has detailed information regarding a food's nutritional value, such as protein, carbohydrates, specific vitamins and so on.

Head back into the main menu, and more options exist, such as an analysis of your meals and food suggestions, trend plotting and so on, but most people will want to look at options 4 and 6. Here you can browse the extensive database, comparing nutritional values of all sorts of food and drink to your heart's content. The entries are extensive—everything from Red Bull to bearded seal meat.

As for the GUI, I'm not 100% sure, but it appears to have more options than the console version, such as reset controls and the ability to control various ratios. Perhaps I missed them in the console version, but either way, there's definitely more on the screen, more of the time. Plus, everything is broken down into tabs, making the whole process more intuitive, saving the user from navigating endless submenus.

All in all, this is a very clever program despite the currently long-winded installation process. Once those issues are ironed out, NUT will be a seriously nifty nutrition program. ■

John Knight is a 26-year-old, drumming- and bass-obsessed maniac, studying Psychology at Edith Cowan University in Western Australia. He usually can be found playing a kick-drum far too much.

Brewing something fresh, innovative or mind-bending?
Send e-mail to newprojects@linuxjournal.com.



HEXAPOD A LINUX- POWERED SPIDER ROBOT

A CHAT WITH
MATT BUNTING,
DEVELOPER
OF THE
HEXAPOD
ROBOT.

ANTON BORISOV

Linux is ubiquitous these days. It powers large-scale HPC systems and small embedded devices—from space to underground. And although it's quite easy (to some degree) to build an unmanned device like a rover or to run a helicopter with Linux, it still is a tough job to create a robot that behaves like a biological object. Fortunately, there's a will and there's a way—and there's Linux. The latter became the core of the hexapod—the robotics device that resembles a spider, developed by Matt Bunting.

FEATURE Hexapod—a Linux-Powered Spider Robot

AB: Matt, you're the creator of the hexapod robot that runs on Linux. But before talking about the robot, tell us about your Linux experience and the role Linux plays in your life.

MB: The field of robotics is what introduced me to the world of Linux. I had previously done basic development in UNIX, so the transition of switching my OS to Linux was seamless. Currently, I use Linux for most things involving development, such as robotics projects and tinkering around with OpenCV. I have a couple Atom-based Linux computers at my house running a custom Webcam-based security system.

AB: What is the hexapod in general, and what ideas were behind its design?

MB: A basic hexapod is a six-legged platform. Each leg can be configured in any number of ways. One such way is a Stewart platform. The version I made is a portable platform (body) with non-rigid feet. Each foot in a portable platform may have any number of degrees of freedom (DOF). Some of the simplest hexapods have three total degrees of freedom, but these are constrained to very simple motions like moving only forward or backward. For incredible flexibility of the platform, three DOF (18 total) are needed per leg to position each "foot" at nearly any given (x,y,z) coordinate. This allows the hexapod to have complete flexibility of the body, so the body may translate or tilt in any direction, in any combination. Recently, there has been an interest in four degrees of freedom per leg. This does not increase



Matt Bunting

the amount of body control, but it can extend the range. I wanted to design the 18 DOF for the best flexibility of the body to use the hexapod for various applications, from implementation of neural networks (NNs) to terrain adaptation methods.

AB: And you have succeeded in this design—a spider with 18 DOF, right?

MB: I would certainly say so. Currently, it is a very solid design that can take a quite a bit of abuse. Though it is fully functional, I'm always looking for ways to upgrade it. I have some ideas floating around, but haven't seriously integrated anything yet.

AB: Can someone achieve these goals with another kind of a robot, say with a rover?

MB: Since my goal was to research legged locomotion techniques, a wheeled rover doesn't make sense for what I want to do in robotics. Sometimes I am asked what the practical applications of a hexapod are, but for many cases, a hexapod is impractical. Mine is terribly power-

inefficient, and the learning techniques I have used take too long to be effective for anything practical, like sending a hexapod on a space mission or to fix an underwater bursted pipe.

The legged aspect and complexity of a hexapod is interesting in the area of machine learning because the complexity makes it difficult to learn efficiently. Wheeled robots may learn locomotion very quickly, which does not generalize the effectiveness of the

algorithms involved. One aspect of machine learning I was interested in included the ability of the machine to re-learn after damage to the robot. For example, if the robot learned to walk, began walking in a dangerous environment and a rock were to crush two legs, then the algorithms implemented could sense that the previously used motions are not very effective, and it could begin to learn an optimal walking policy based on the new configuration.

AB: Describe how you "teach" a hexapod to walk. Previously, you mentioned NNs and adaptation methods.



The Hexapod looks like a real biological spider.

Do you load some basic behavior functions into software, and later the spider tries to find the optimal walk for every kind of a terrain, or is there another approach?

MB: I have, so far, explored two different learning mechanisms, both of which learn in very different ways. The first was a reinforcement learning technique called Q-Learning with Softmax action selection. Basically, the hexapod would experiment between different motor states. Starting from one state, the algorithm applies Softmax to the current Q-matrix for action selection probability and chooses one based on a random number generator. The values in the Q-matrix with higher values (good transitions to move forward) are more likely to be selected, while others (poor transitions, like moving backward) are not as likely to be selected. Once an action is selected, the hexapod first takes an image using a Webcam pointing in the forward direction, then performs the action. The action moves the hexapod into a new state. In the new state, another image is taken. The state transition reward is measured by the difference between the two images.

I used a function in OpenCV to measure optic flow. The resulting vector field from the optic flow function is run through some simple math to determine how the hexapod moved based on the overall directionality of the vectors. For example, if the resulting image had a sense of "zooming in" from moving the hexapod forward, all the vectors would point outward from the center of the image. This translates to a high reward. If the vectors are pointing toward the center, the hexapod moved backward, resulting in a cost or negative reward. I also included twisting of the body and translational movements of the camera as negative rewards because the goal was to get it to walk forward very smoothly.

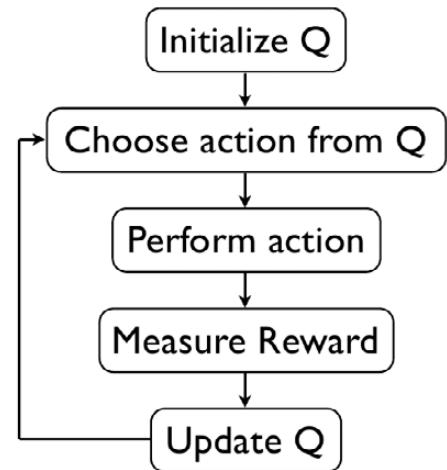
The reward value is used to update the Q-Matrix. Based on the transition reward and possible future rewards from the new state, the Q-value is updated. Softmax action selection gives a nice balance between explorative and exploitative behavior. Once a gait is well learned, there is a small probability that the hexapod will explore other state transitions. If, however, the hexapod were to break a leg and apply a previously learned method, the reward values would decrease tremendously. The newly updated Q-matrix would result in a low probability of exploring the previously learned transitions and begin to experiment with others.

The NN is very much different. I based a good

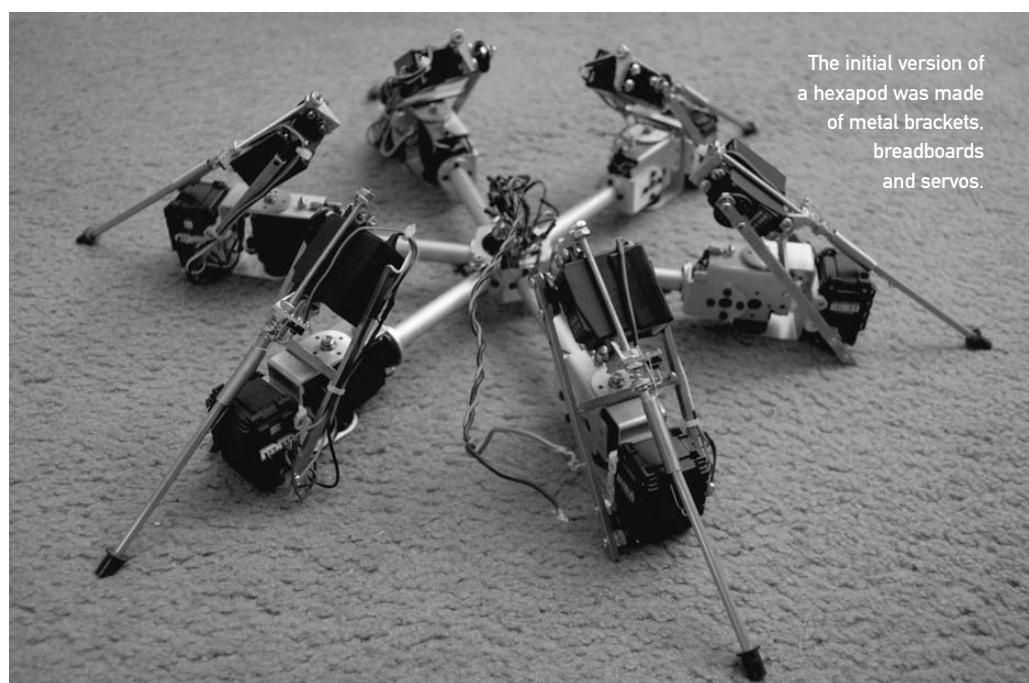
portion of the network on Randall Beer's work on a neural-network-based hexapod. The neurons I used were different, more biologically influenced. I believe that Beer used continuous neurons, which were differentiable, but the ones I used were integrated and fired with adaptation. They don't

truly mimic biology, but they are much closer than continuous-based neurons. I constructed something similar to Beer's network, but I have to add additional neurons to move the legs around. The network is based around an oscillator called a Central Pattern Generator (CPG), also called a Pacemaker Neuron. These are created by mutually inhibiting two neurons. One begins to fire, preventing the other from firing, but slows down due to adaptation, and eventually the other neuron will begin to fire, inhibiting the first one, and so on. These CPGs are used to drive other neurons that operate the legs.

To learn the weights for each connection correctly, a genetic algorithm was used to train the weights. I used a population size of 16, and the fitness function was based on running the hexapod



Q-Learning Technique Algorithm



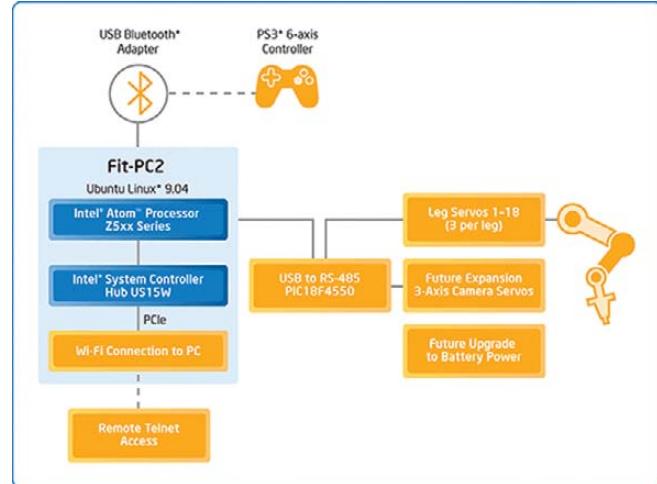
for 10 seconds and measuring the traveled distance and resulting direction angle. The closest the resulting angle to the starting orientation and the farthest distance traveled resulted in the best fitness value. The hexapod learned to walk very effectively after 100 generations.

AB: You mean 100 iterations of the learning, or 100 steps here?

MB: In genetic algorithms, this is the number of generations, which can be thought of as iterations in a way. In each generation, a new population size of 16 is created.

AB: Several sources say that the robot was assembled from trash parts. Is that true?

MB: This is a bit misinterpreted by many people, because they think that the videos I posted on YouTube are where I am at now, not where I was when I started. The hexapod began as a project for a class, and I had limited money, time and resources. I (like any hobbyist, maker or hacker) have a drawer full of parts I've been collecting for years. I started this drawer when I was nine years old, and my mentor at the time called it the "things-I-don't-know-what-they-are-but-they-may-be-useful someday drawer". So when the time came for the class project, I searched through all the servo motors, breadboards and metal brackets I had and threw together a hexapod of mismatched legs to implement the reinforcement technique called Q-learning. This was not the hexapod I put on YouTube.



General Scheme of How the Robot Is Controlled

cooperation with Intel. At the time of searching for an embedded solution, I was looking into Gumstix boards and Via processor-based boards. These were great for kinematics integration, but for heavy computation of vision, I needed something even more powerful. Fortunately at the time, CompuLab had just released its

I wanted ultimate onboard computational power for vision processing, which led to fit-PC2 from CompuLab.

The professor of the class, Dr Tony Lewis, really liked the project and was looking to hire an undergrad into his lab. I also needed a job so he hired me into his lab, the Robotics and Neural Systems Laboratory (RNSL). At RNSL, there is a Dimension 3-D printer, so I took advantage of it and saved up my money to buy matching motors. I gave the hexapod a complete makeover (including a new Atom-based fit-PC2) with no junk parts, and then I posted a video on YouTube. This is the video that Stewart Christie of Intel saw. Also, the video shows no learning at all. It was just a demonstration to forum members at hexapodrobot.com/forum of my platform in action.

AB: Linux is the key to success for the hexapod. Why did you decide to use Linux, and what particular distribution was the final choice?

MB: I wanted ultimate onboard computational power for vision processing, which led to fit-PC2 from CompuLab. At RNSL, Ubuntu is the distribution of choice and is used for everything from simulations to direct robot operation. We use Linux because of the flexibility and the available resources.

AB: The main hardware unit, an Atom motherboard, was donated by Intel. Could you shed some light on your cooperation with Intel?

MB: Originally, I had purchased the computer before any

fit-PC2, which is the ideal computer for robotics. Once I posted the video, a couple days later, Stewart from Intel wrote a comment on my YouTube video regarding interest in the hexapod. I contacted him, and after a month or so, he said that Intel would want me to create two hexapods, one for tradeshows and the other for me, but the Intel folks can borrow mine if they have multiple demonstrations at the same time.

AB: Speaking practically, the hexapod is actually a high-tech model, because the use of 3-D Stratasys printers won't be available for casual amateurs. You've made a model—that is, a framework and legs in CAD software—printed it and attached servos and a camera to the body and that's it. Am I missing something?

MB: 3-D printing has been becoming incredibly cheap, where hobbyists can make their own home 3-D printer for around \$1,000. The quality is not anywhere near what the Stratasys machine can do that I use, but functional parts still may be created. My hexapod is not very different functionally from other hexapods that can be put together with off-the-shelf components, but since I had the Stratasys machine at my disposal, I took full advantage of it and tried to make complex, curved shapes. I actually used SolidWorks for the mechanical design, because it is freely offered by my university, but I will have to look into QCad.

BUILDING A SPIDER

Here are the required resources to build a spider:

- Framework: six servo motors AX-12/RX-10/RX-28 from Robotis, 3D-printed casing.
- Computing unit: one fit-PC2 motherboard based on Intel Atom Z530 from CompuLab.
- Labor hours: 380.
- Distribution: Ubuntu Linux 9.08.
- Software used: GEdit, OpenCV and C++.
- Scientific field: neural networks and adaptation methods.

AB: The robot without its middleware is just another embedded Linux device. Did you use a specific software for your “spider”, or did you write software for it?

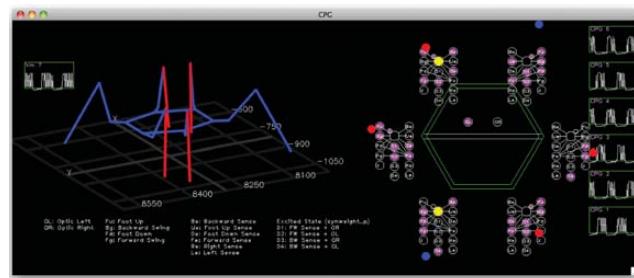
MB: I developed all the software for the hexapod in C++, using OpenCV libraries for simulations and vision processing.

AB: Can you estimate how much time (in hours) it took to construct the hexapod (put together the mechanical and electrical parts) and write the software itself?

MB: Ha ha. It is an enormous amount for sure. I worked on it for hours a day, weeks at a time. The original Q-Learning robot probably took 200 hours. Then I gave it a makeover, which took maybe 100 hours, and then the other projects took maybe 80 each.

AB: You said previously that the operating system beneath the cover is Ubuntu. What IDE did you use to write the robot’s middleware—Qt, WxWidgets, ncurses or something else?

MB: I actually use only GEdit and purely in C++ using OpenCV libraries. The simulations I built were done purely using OpenCV, drawing individual lines at a time. I feel like I re-invent the wheel when I do this, but I sure learn a lot!



First experiment is done in simulated environment.

AB: Do you release schematics and auxiliary software of a robot under the GPL or under another license?

MB: I have been looking to release my kinematics code under a license, but I know nothing in this area. Maybe *LJ*

readers could point me in the right direction?

AB: You’re a senior student at the University of Arizona, and this project is to some degree your graduation work. Do you plan to continue its future development?

MB: I look forward to implementing more functionality into the hexapod as a graduate student at the University of Arizona. Currently, I am working on a miniature version. I do want to explore the link between vision and legged locomotion through the use of biologically inspired neural networks further.

AB: Where, in your opinion, can this “spider” be used and be useful for people?

MB: The hexapod could make a great search-and-rescue-style robot. In a natural disaster like an earthquake, it would be desirable to have a large number of robots searching autonomously for survivors in a building turned to rubble. The environment is treacherous though and could cause damage to the robot. Once damage is inflicted upon the hexapod, a purely inverse kinematic-based hexapod would be rendered useless. If, however, it could re-learn, the hexapod still could operate given its new configuration. Like I mentioned previously though, it is more of a tool to explore machine learning techniques and out of the research, hopefully discover faster algorithms. Hopefully, better algorithms can be used for consumer-based robotics to increase the standard.

AB: Do you plan to use the hexapod or your next prototype CrustCrawler in autonomous walking research?

MB: Sure. The learning mechanisms I’ve implemented are more scientific, and I certainly encourage everyone to explore science with robotics platforms, but it is not necessarily the goal for the CrustCrawler platform. I see that as a means of making it easy for anyone to get started in the field of hexapod robotics. If the different geometry is more conducive to what I want the hexapod to do, then I will, of course, use it. The hexapods are simply tools in my mind for research. I certainly want my hexapods to be able to roam around on their own and accomplish some task even if it is mundane. The nice thing about the CrustCrawler hexapod and my personal hexapod is that they can be easily interchanged, as they use motors that communicate using identical protocols. All I have to do is change a few geometric calibration settings and it will run just fine.■

Anton Borisov has broad spheres of interests, ranging from clusters and embedded devices to artificial intelligence and programmatic puzzles. One thing that unites them all is Linux, his most favorite operating system.

Photos courtesy Matt Bunting, Intel Corp.



Debugging Embedded Linux Platforms with GDB and Python

Give your debugging sessions
go-faster stripes
with the power of Python.

TOM PARKIN

If you write code for Linux systems, chances are you will have used the venerable GNU Debugger (GDB). Acting as a back end for many GUIs and the interface to various JTAG debugging tools in the embedded world, GDB is the foremost debugger for Linux. As of release 7.0, GDB gained a compelling new capability: support for scripting debugging actions using a Python interpreter. In this article, I take a look at how to drive GDB using Python and apply this knowledge to the vexatious issue of debugging an embedded Linux platform.

The Challenge of Embedded Linux Debugging

Debugging Linux programs on an x86 PC platform, although not necessarily easy, at least is well supported by a variety of tools. Most Linux distributions package development and debugging tools to assist with anything from profiling runtime performance through tracing memory leaks and detecting deadlocks.

Embedded platforms are rarely so well served. Although a number of projects seek to provide the kind of polish and integration for embedded development that is taken for granted on the desktop, these are not yet widely adopted in all areas of embedded Linux development. Many embedded devices are developed using what effectively is a handcrafted Linux distribution, closely tied to the specific goals of that device. The time required to integrate a wide range of handy debugging tools into that environment, especially in the fast-paced world of consumer electronics, is an overhead few teams can meet.

Many embedded platforms seek to save on resource overhead through the use of "low-fat" system libraries (such as uClibc in the place of glibc), which may make the integration of some debugging tools more difficult. Indeed, in some cases, the architecture of the CPU used by the target platform will prevent the use of certain tools altogether. The excellent Valgrind instrumentation framework, for example, has limited support for the ARM architecture and no support at all for MIPS or SuperH.

Happily, GDB is widely available for embedded devices because it is easy to cross-compile and supports a wide range of target architectures.

The nature of embedded devices often means that CPU cycles and memory are scarce. Any debugging tool that weighs too heavily on either may make its use impractical on an embedded device, especially when attempting to debug race conditions and the like.

The net result of this inconsistent provision of debugging tools across the embedded Linux world is that most developers have to makeshift as best they can with the tools that are available. Happily, GDB is widely available for embedded devices because it is easy to cross-compile and supports a wide range of target architectures. And with the recent integration of Python scripting support, GDB can extend beyond the typical debugging tasks of single stepping and variable examination.

Scripting GDB with Python

GDB has long supported extension via pre-canned sequences of debugger commands. This ability makes it possible to automate certain parts of a debugging work flow and even implement new debugger functions.

Integrating Python into GDB adds an extra dimension to the possibilities of GDB scripting and extension. In addition to the simple functions and flow control of GDB's native scripting language, the full power of the Python language is made available.

The Python GDB API is presented as a Python module called `gdb`, which provides access to GDB's internal representation of a process under debugging. The module includes interfaces to process information, threads, stack frames, values and types, functions,

symbols and break points. In addition, a mechanism is provided to inject commands into the GDB command-line interface.

The result is that the internals of GDB are now available as a rich set of libraries for programmatic driving of the debugger. This creates a whole range of new opportunities for extension and automation. For example, let's imagine you want to debug calls to `malloc()` in a large application, but you're really interested only in calls from a certain module. Ideally, you want to be able to break execution only when one of the module's functions is in the backtrace at the point that `malloc()` is called. The Python API gives you that flexibility.

Problem Code

To explore the use of Python within GDB, let's debug a small C program, the code for which is shown in Listing 1. It performs the simple task of printing the phrase "Hello World!" in a rather convoluted manner, and it has at least one obvious bug. Besides being over-engineered for the task at hand, `hello_world.c` makes use of two mutexes for serializing access to different data structures, and not all users of these mutexes agree on the order in which locks should be acquired. This quickly yields a runtime deadlock.

Although `hello_world.c` is somewhat contrived, it does demonstrate the kinds of runtime bugs that multithreaded applications can come across when mutexes are required to protect data structures from different contexts.

Before reading any further, it is worth considering how you might debug such a deadlock. On an x86 platform, you could consider using the Valgrind framework's `drd` tool. Alternatively, you may choose to recompile the code with different options to change the behavior. But what would you do if Valgrind did not work on your platform, or if the code you wanted to rebuild was a third-party library for which you had only binaries?

Setting Up the Environment: the Embedded Platform

The example platform for this article uses a little-endian MIPS-based System On a Chip (SOC) device. MIPS is widely used in home routers, such as the popular Linksys WRT54G series, as well as in many set-top box platforms for accessing digital television services. Our platform has a fairly powerful 400MHz CPU, as well as 512MB of DDR RAM, making it a quite capable embedded device. We can communicate with the platform over a serial console and using an Ethernet port.

On the software side, our platform runs a 2.6 series Linux kernel that has been extended by the SOC manufacturer to support the specific CPU we are using. It has a fairly typical userspace based around uClibc and BusyBox, along with a range of GNU utilities, such as `awk` and `sed`.

Setting Up the Environment: Cross-Compiling GDB

In order to run GDB on our embedded platform, we will make use

FEATURE Debugging Embedded Linux Platforms with GDB and Python

Listing 1. C Source Code for hello_world

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>

#define THREAD_COUNT      32

/* String output data */
static const char    *string = "Hello World!\n";
static int            cursor = 0;
pthread_mutex_t      print_lock = PTHREAD_MUTEX_INITIALIZER;

/* Runtime statistics */
static int            chars_printed = 0;
pthread_mutex_t      statistics_lock = PTHREAD_MUTEX_INITIALIZER;

/* Print one character of the string "Hello World!" to stdout */
/* Returns a pointer to the character printed */
static char *say_hello(void)
{
    char *printed_letter = NULL;

    printf("%c", string[cursor]);
    if (++cursor > strlen(string)) {
        cursor = 0;
        fflush(stdout);
    }

    printed_letter = (char *) malloc(1);
    if (printed_letter) {
        *printed_letter = string[cursor];
    }

    return printed_letter;
}

/* A "bug-free" printer function */
static void *good_printer(void *data)
{
    char *c = NULL;

    while(1) {
        c = NULL;
        pthread_mutex_lock(&print_lock);
        pthread_mutex_lock(&statistics_lock);
        c = say_hello();
        if (c) free(c);
        chars_printed++;
        pthread_mutex_unlock(&statistics_lock);
        pthread_mutex_unlock(&print_lock);
    }
    return NULL;
}

/* A buggy printer function */
static void *bad_printer(void *data)
{
    while(1) {
        pthread_mutex_lock(&statistics_lock);
        pthread_mutex_lock(&print_lock);
        say_hello();
        chars_printed++;
        pthread_mutex_unlock(&print_lock);
        pthread_mutex_unlock(&statistics_lock);
    }
    return NULL;
}

int main (int argc, char **argv)
{
    pthread_t threads[THREAD_COUNT];
    int i;

    /* Spawn many good children threads */
    for (i = 1; i < THREAD_COUNT; i++) {
        if (0 != pthread_create(&threads[i], NULL, good_printer, NULL)) {
            perror("pthread_create");
            exit(EXIT_FAILURE);
        }
    }

    /* Spawn one bad child thread */
    if (0 != pthread_create(&threads[0], NULL, bad_printer, NULL)) {
        perror("pthread_create");
        exit(EXIT_FAILURE);
    }

    pthread_join(threads[0], NULL);

    return EXIT_SUCCESS;
}
```

of the gdbserver tool for remote debugging. This allows us to run GDB on a Linux PC, connecting to the embedded target using Ethernet. The protocol GDB uses to communicate with gdbserver is compatible across releases, so we can update the GDB installation on our host PC without needing to install a new version of gdbserver on the target.

Because most distributions do not package GDB with MIPS architecture support, we need to compile GDB from source. This is accomplished easily using the instructions in the source tarball,

which can be downloaded from the GDB Web site. If you get stuck with cross compilation or with the GDB/gdbserver configuration, plenty of good references exist on-line that will help; the Resources section for this article lists a few.

Initial Debugging

Now that we have GDB cross-compiled and installed, let's take a look at debugging the deadlock on the embedded target.

First, run gdbserver on the target and attach to the

deadlocked process:

```
gdbserver :5555 --attach <pid of process>
```

Now, fire up GDB on the host PC:

```
mipsel-linux-uclibc-gdb
```

Once GDB is running, point it at the target's root filesystem and at the file to debug:

```
(gdb) set solib-absolute-prefix /export/shared/rootfs  
(gdb) file hello_world  
(gdb)
```

Finally, tell GDB to attach to the process running on the target via gdbserver:

```
(gdb) target remote 10.0.0.6:5555  
(gdb)
```

If all goes well, now you should be able to explore the running process a little to see what is going on. Given that the process has deadlocked, examining the state of the threads in the process is a good first port of call:

```
(gdb) info threads  
Id  Target Id  Frame  
33  Thread 737  0x2aac1068 in __lll_lock_wait from libpthread.so.0  
32  Thread 738  0x2aac1068 in __lll_lock_wait from libpthread.so.0  
31  Thread 739  0x2aac1068 in __lll_lock_wait from libpthread.so.0  
....  
3  Thread 767  0x2aac1068 in __lll_lock_wait from libpthread.so.0  
2  Thread 768  0x2aac1068 in __lll_lock_wait from libpthread.so.0  
1  Thread 736  0x2aab953c in pthread_join from libpthread.so.0  
(gdb)
```

The omitted threads in the GDB output are all similarly blocking in __lll_lock_wait(), somewhere in the depths of libpthread.so. Clearly, some of these threads must be waiting for a mutex that another thread has not given up—but which threads, and which mutex?

Some examination of the libpthread source in the uClibc tree shows us that __lll_lock_wait() is a low-level wrapper around the Linux futex system call. The prototype for this function is:

```
void __lll_lock_wait (int *futex, int private);
```

On MIPS, the a0 register typically is used for the first argument to a function. So if we examine a0 for each thread that is blocked in __lll_lock_wait(), we should get a good idea of which threads are waiting on which mutexes. That's a good start, but ideally we want to find out which thread currently owns each mutex. How can we manage that?

Going back to the uClibc sources, we can see that __lll_lock_wait() is called from pthread_mutex_lock(). The integer pointer provided to __lll_lock_wait() is actually a pointer to the pthread_mutex_t structure:

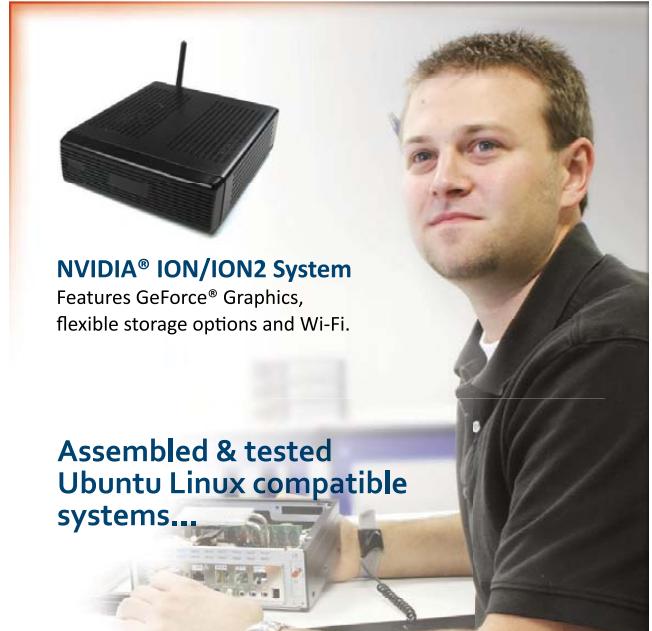
```
typedef union  
{
```

```
struct __pthread_mutex_s  
{  
    int __lock;  
    unsigned int __count;  
    int __owner;  
    int __kind;  
    unsigned int __nusers;  
    __extension__ union  
    {  
        int __spins;  
        __pthread_slist_t __list;  
    };  
    } __data;  
    char __size[ __SIZEOF_PTHREAD_MUTEX_T ];  
    long int __align;  
} pthread_mutex_t;
```

The __owner field looks interesting, and on further investigation it seems that __owner is set to the thread ID (TID) of the thread that is currently holding the mutex.

By combining these two pieces of information (namely the mutex pointer provided to __lll_lock_wait(); and the __owner field two integers on in that structure), we should be able to find out which threads are blocking on which mutexes.

The trouble is that this would be very tedious to iterate



NVIDIA® ION/ION2 System

Features GeForce® Graphics,
flexible storage options and Wi-Fi.

**Assembled & tested
Ubuntu Linux compatible
systems...**

Genuine expertise. **LOGIC**
SUPPLY®
www.logicsupply.com/linux

© 2011 Logic Supply, Inc. All products and company names listed are trademarks or trade names of their respective companies.

through by hand. Each thread that is blocking in `_lll_lock_wait()` will need to be selected. Then the contents of register `a0` must be queried for the appropriate stack frame of each thread, and the memory at the location pointed to by `a0` examined to discover which thread owns the mutex that the thread is waiting for. Even for this trivial program, we have some 32 threads to look at, which is a lot of manual work.

Putting Python into Practice

Rather than driving the debugger by hand, let's instead look at how we can automate the task described above using the GDB Python API. First, we need to be able to iterate over each thread in the process under debugging (the "inferior", in GDB terminology). To do this, we can use the `threads()` method of the `gdb.Inferior` class:

```
for process in gdb.inferiors():
    for thread in process.threads():
        print thread
```

That was easy. Now we need to look at the currently executing stack frame for each thread and figure out whether it is waiting on a mutex. We can do this using the `gdb` module function `selected_frame()` and the `name()` method of the `gdb.Frame` class:

```
for process in gdb.inferiors():
    for thread in process.threads():
        thread.switch()
        frame = gdb.selected_frame()
        if frame.name() == "__lll_lock_wait":
            print "Thread is blocking in __lll_lock_wait"
```

So far, so good. Now that we have a method for programmatically finding each thread that is waiting on a mutex, we need to examine the contents of the `a0` register for each of those threads. This should extract a pointer to the mutex structure that the thread is waiting on. Happily, GDB provides a convenience variable, `$a0`, which we can use to access the `a0` register. The `gdb` module function `parse_and_eval()` provides API access to convenience variables, amongst other things:

```
for process in gdb.inferiors():
    for thread in process.threads():
        thread.switch()
        frame = gdb.selected_frame()
        if frame.name() == "__lll_lock_wait":
            print "Thread is blocking in __lll_lock_wait"
            a0 = gdb.parse_and_eval("$a0")
```

The last piece of information we need to extract from GDB is the contents of memory at the pointer in the `a0` register so that we can determine the `__owner` field for each mutex in play. Although it's probably cheating to do so, we can fall back on the `gdb` module function `execute()` to pass the `x` command to the GDB command-line interface. This will print the contents of memory to a string that we can parse to find the required information:

```
for process in gdb.inferiors():
    for thread in process.threads():
        thread.switch()
        frame = gdb.selected_frame()
```

```
if frame.name() == "__lll_lock_wait":
    print "Thread is blocking in __lll_lock_wait"
    a0 = gdb.parse_and_eval("$a0")
    s = gdb.execute("x/4d $a0", to_string=True).split()
    s.reverse()
    owner = int(s[1])
```

It's not particularly pretty to look at, but it works. This code splits the string returned from the `x` command into a whitespace-delimited list. Because GDB may alter the labels used at the start of the output depending on what symbolic information it can extract from the application binary, we then

Listing 2. Python Code for GDB Mutex Debugging

```
from collections import defaultdict

# A dictionary of mutex:owner
mutexOwners = {}

# A dictionary of blocking mutex:(threads..)
threadBlockers = defaultdict(list)

# Print the threads
print "Process threads : "
gdb.execute("info threads")

print "Analysing mutexes..."
# Step through processes running under gdb
for process in gdb.inferiors():

    # Step through each thread in the process
    for thread in process.threads():

        # Examine the thread -- is it blocking on a mutex?
        thread.switch()
        frame = gdb.selected_frame()
        if frame.name() == "__lll_lock_wait":

            # a0 is the first argument passed to the function
            a0 = gdb.parse_and_eval("$a0")
            mutex = int(a0)

            # Make a note of which thread blocks on which mutex
            threadBlockers[mutex].append(thread)

            # Make a note of which thread owns this mutex
            if not mutex in mutexOwners:
                s = gdb.execute("x/4d $a0", to_string=True).split()
                s.reverse()
                mutexOwners[mutex] = int(s[1])

        # Print the results of the analysis
        for mutex in mutexOwners:
            print " Mutex 0x%lx :" % mutex
            print "     -> held by thread : %d" % mutexOwners[mutex]
            s = ["%d" % t.ptid[2] for t in threadBlockers[mutex]]
            print "     -> blocks threads : %s" % ''.join(s)
```

reverse the list and pull out the second-to-last value. This yields the third integer value in the structure, which in this case is the `_owner` field of `pthread_mutex_t`.

All that remains to do now is to plug all of these pieces of data together to provide some useful information. Listing 2 shows the full Python code to do this. Putting it all together:

```
(gdb) source mutex_check.py
Process threads :
Id Target Id Frame
33 Thread 737 0x2aac1068 in __lll_lock_wait from libpthread.so.0
32 Thread 738 0x2aac1068 in __lll_lock_wait from libpthread.so.0
....
3 Thread 767 0x2aac1068 in __lll_lock_wait from libpthread.so.0
2 Thread 768 0x2aac1068 in __lll_lock_wait from libpthread.so.0
1 Thread 736 0x2aab953c in pthread_join from libpthread.so.0
Analysing mutexes...
Mutex 0x401cf0 :
-> held by thread : 740
-> blocks threads : 737 738 739 741 742 743 744 745 746 747
    748 749 750 751 752 753 754 755 756 757
    758 759 760 761 762 763 764 765 766 767
    768

Mutex 0x401d08 :
-> held by thread : 768
-> blocks threads : 740
(gdb)
```

The deadlock now becomes very clear. Thread 740 is waiting for a mutex currently owned by thread 768, and thread 768 in turn is waiting for the mutex that thread 740 currently owns. Neither thread can run until the mutex owned by the other becomes available. Returning to the GDB prompt, we can generate backtraces for both threads to gain more insight:

```
(gdb) t 30
[Switching to thread 30 (Thread 740)]
#0 0x2aac1068 in __lll_lock_wait ()
(gdb) bt
#0 0x2aac1068 in __lll_lock_wait ()
#1 0x2aab568 in pthread_mutex_lock ()
#2 0x00400970 in good_printer (data=0x0) at hello_world.c:45
#3 0x2aab7f9c in start_thread ()
#4 0x2aac2200 in __thread_start ()
Backtrace stopped: frame did not save the PC
(gdb) t 2
[Switching to thread 2 (Thread 768)]
#0 0x2aac1068 in __lll_lock_wait ()
(gdb) bt
#0 0x2aac1068 in __lll_lock_wait ()
#1 0x2aab568 in pthread_mutex_lock ()
#2 0x00400a04 in bad_printer (data=0x0) at hello_world.c:60
#3 0x2aab7f9c in start_thread ()
#4 0x2aac2200 in __thread_start ()
Backtrace stopped: frame did not save the PC
(gdb)
```

As the backtraces show, the two threads have followed two different code paths to end up in the deadlock situation. Reviewing the code for `hello_world` in light of this information

MIPS Registers

The **MIPS** architecture has 32 general-purpose integer registers. Of these, the hardware architecture specifies that registers 0 and 31 are used for the value zero and the function return address, respectively. The usage of the rest of the registers is entirely defined by the software toolchain.

By convention, however, the use of the general-purpose **MIPS** registers is quite firmly set to allow software interoperability. For example, registers 4 to 7 are used to pass the first four non-floating-point arguments to functions and are given the names `a0` to `a3`.

should allow us to find the bug: `bad_printer()` is taking the print and statistics locks in the wrong order.

Conclusion

Adding a Python API to GDB provides another capable weapon in the Linux debugging arsenal. For embedded systems, where other debugging tools may not be so widely available, a powerful programmatic interface to GDB can make the difference between hours of painstaking debugging and minutes of enjoyable scripting.

Astute readers will have noted that the bug we have discovered in this article is not the only bug in `hello_world.c`. The task of finding and fixing the remaining bugs is left as an exercise for readers to tackle with their new-found GDB Python knowledge. Have fun! ■

Tom Parkin (tom.parkin@gmail.com) has been working with Linux and embedded systems for ten years and is still finding new things to get excited about. When not in front of a computer, he enjoys 10k runs and Real Ale, although not in combination.

Resources

GDB: www.gnu.org/software/gdb

GDB/Python Wiki: sourceware.org/gdb/wiki/PythonGdb

Tom Tromey's Excellent Blog Posts about GDB and Python: tromey.com/blog/?cat=17

OpenWrt's GDB Cross-Compilation Makefile:
<https://dev.openwrt.org/browser/trunk/toolchain/gdb/Makefile>

A How-To for GDB/gdbserver Usage:
www.linux.com/archive/feature/121735

uClibc Project: uclibc.org

Linux Futex Information: kernel.org/doc/man-pages/online/pages/man2/futex.2.html

BREAKING FREE THE GUMSTIX DSP

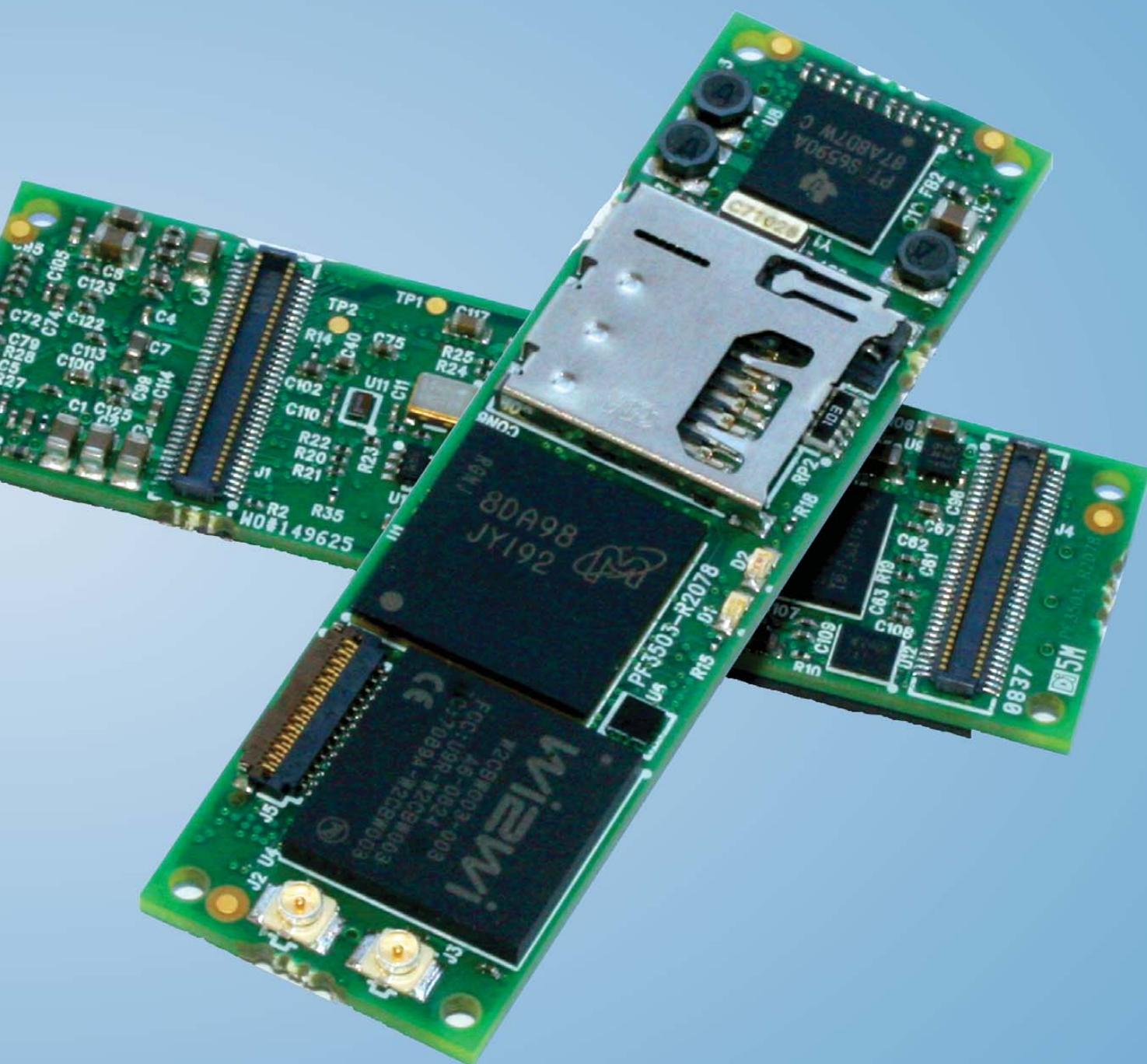


**Compiling a Linux 2.6.33 kernel for the
Gumstix Overo Fire with DSP support.**

JAMES MCCOLL

For a senior design project, I was part of a team that needed to process Fast Fourier Transforms on an embedded computer. Specifically, I chose to use the Gumstix Overo Fire because of the integrated Digital Signal Processor (DSP), which has Fast Fourier Transforms as part of the Texas Instruments API. While researching this project, I found the activation of the DSP on the Gumstix to be a relatively unexplored area. So using a combination of methods from the DSPBridge on the BeagleBoard, the Gumstix Developer and the PIXHAWK Project Web sites, I arrived at this method for compiling the Linux-2.6.33 kernel for the Gumstix Overo Fire with DSP support.

The Overo Fire comes with three processors: an ARM Cortex-A8 CPU, POWERVR SGX and C64x+ Digital Signal Processor core. While the ARM provides for the general-purpose processing, the Digital Signal Processor can be used to perform more mathematically intensive calculations. For this reason, the purpose of this article is to describe how to compile a Linux 2.6.33 kernel for the Gumstix Overo Fire to leverage the extra processing power of the DSP for future projects.



Pre-Compilation Package Installation

Go to the DSPBridge Project Wiki (www.omappedia.org/wiki/DSPBridge_Project), and follow the instructions for retrieving both the kernel-dspbridge and the userspace-dspbridge packages from the DSPbridge git repository, but do not issue any make command as mentioned in the tutorial on the Web page.

Download and install the Code Sourcery tool suite (specifically CodeSourcery G++ Lite 2008q3-72) and both bios_SetupLinux_5_33_04.bin and ti_cgt_c6000_6.0.7_Setup_Linux_x86.bin packages from the Texas Instruments Web site. The Texas Instruments tools require a Texas Instruments user account in order to download. Installing both of the Texas Instrument Tools in the same directory will help with configuring the \$DEPOT variable later.

An additional tool needed for compiling the DSP image for the Gumstix is u-boot from Steve's Beagles U-boot git repository. Use the following commands:

```
git clone git://gitorious.org/u-boot-omap3/mainline.git u-boot-omap3
cd u-boot-omap3
git checkout --track -b omap3-dev origin/omap3-dev
```

Or, go to www.sakoman.com/cgi-bin/gitweb.cgi?p=u-boot.git;a=summary and download the packages there.

Next, change into the newly installed u-boot directory and issue the make tools command in order to compile mkimage, which later will be used to create a uImage. Ensure that the OpenEmbedded and bitbake programming suites are installed for the Overo Gumstix profile. Use the default config for the Linux 2.6.33 kernel as the starting config of the new DSP-enabled kernel. Installation instructions for OpenEmbedded and bitbake can be found at www.gumstix.org/software-development/open-embedded/61-using-the-open-embedded-build-system.html.

Finally, create an SDRAM card with a bootable Gumstix image. Instructions for creating a bootable SDRAM card can be found at www.gumstix.org/create-a-bootable-microsd-card.html.

Setting Up the Compilation Environment

To set up the compilation environment, point the CROSS_COMPILE variable at the Code Sourcery tool suite using the command:

```
export CROSS_COMPILE= <path to tool suite>/CodeSourcery/
➥ Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-
```

An additional tool needed for compiling the DSP image for the Gumstix is u-boot from Steve's Beagles U-boot git repository.

and the DEPOT variable with the command:

```
export DEPOT=<path to TI tools>/
```

Additionally, set the architecture of the device to ARM using the command export ARCH=arm, and add the u-boot directory and cross compilers to the path using the commands:

```
PATH=<path to u-boot directory>/u-boot-omap3/
➥ tools:<path to tool suite>/CodeSourcery/
Sourcery_G++_Lite/bin:$PATH
```

and:

```
export PATH
```

Compiling the Kernel

Before compiling the kernel, reset the git repository from the current branch to the Linux 2.6.33 kernel to make it compatible with the Gumstix configuration file. (At the time of this writing, the default DSPbridge is set to Linux 2.6.37.) First, issue the git branch command and check out the current DSPbridge branch. Now, issue:

```
git reset - -hard 85343cd5491260881b34ab7bb7cdc8fdeef078e4
```

After the branch is reset, check the Makefile for the proper Linux information at the top. In order to keep the kernel size to a minimum, build the kernel in an output directory, which will be referred to as the production directory for the remainder of this

```
Low-level debug console UART
1.UART1(OMAP_LL_DEBUG_UART1) (NEW)
2.UART2(OMAP_LL_DEBUG_UART2) (NEW)
3.UART3(OMAP_LL_DEBUG_UART3) (NEW)
>4.None(OMAP_LL_DEBUG_NONE) (NEW)
choice[1-4]:4
...
OMAP34xx Based System (ARCH_OMAP34XX) [N/y] (NEW)y
OMAP3430 support (ARCH_OMAP3430) [Y/n]y
*
*OMAP Board Type
*
OMAP3 BEAGLE board (MACH_OMAP3_BEAGLE) [N/y]n
OMAP3 LDP board (MACH_OMAP_LDP) [N/y]n
Gumstix Overo board (MACH_OVERO) [Y/n]y
OMAP3530 EVM board (MACH_OMAP3EVM) [N/y]n
OMAP3517/AM3517EVM board (MACH_OMAP3517EVM) [N/y]n
OMAP3 Pandora (MACH_OMAP3_PANDORA) [N/y]n
OMAP3 Touch Book (MACH_OMAP3_TOUCHBOOK) [N/y]n
OMAP 3430 SDP board (MACH_OMAP_3430SDP) [N/y]n
Nokia RX-51 board (MACH_NOKIA_RX51) [N/y]n
OMAP3 Zoom2 board (MACH_OMAP_ZOOM2) [N/y]n
Compulab CM-T35 module (MACH_CM_T35) [N/y]n
IGEP0020 (MACH_IGEP0020) [Y/n]y
OMAP3630 SDP board (MACH_OMAP_3630SDP) [N/y]n
OMAP3 debugging peripherals (OMAP3_EMU) [N/y/?]n
Enable SDRC AC timing register changes (OMAP3_SDRC_AC_TIMING) [N/y/?]n
...
*
*DSP Bridge driver
*
DSP Bridge driver (MPU_BRIDGE) [N/m/y/?] (NEW)y
Physical memory pool size (Byte) (BRIDGE_MEMPOOL_SIZE) [0x600000] (NEW)
DSP Bridge Debug Support (BRIDGE_DEBUG) [N/y/?] (NEW)
DSP Recovery Support (BRIDGE_RECOVERY) [N/y/?] (NEW)
Check buffers to be 128 byte aligned (BRIDGE_CACHE_LINE_CHECK) [N/y/?] (NEW)
Enable WDT3 interruptions (BRIDGE_WDT3) [N/y/?] (NEW)
*
*Bridge Notifications
*
Notify DSP Power Error (BRIDGE_NTFY_PWERR) [N/y/?] (NEW)
```

Figure 1. Changes Made during make oldconfig

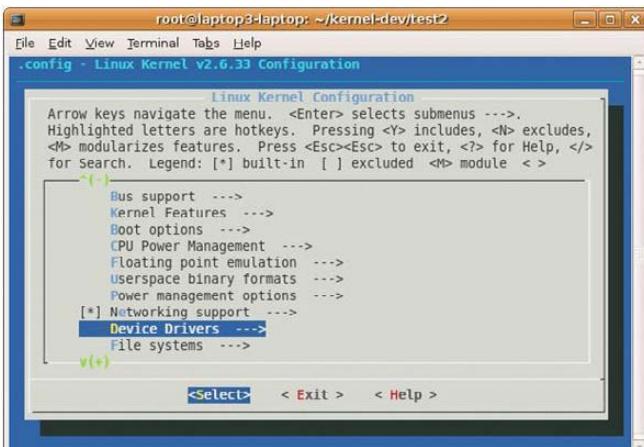


Figure 2. Navigating to Device Drivers

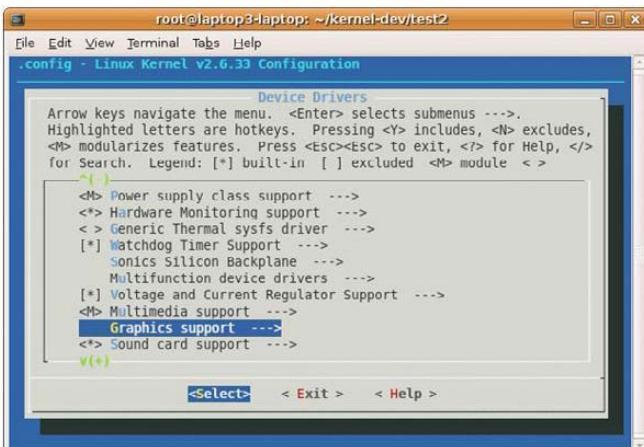


Figure 3. Navigating to Graphics Support

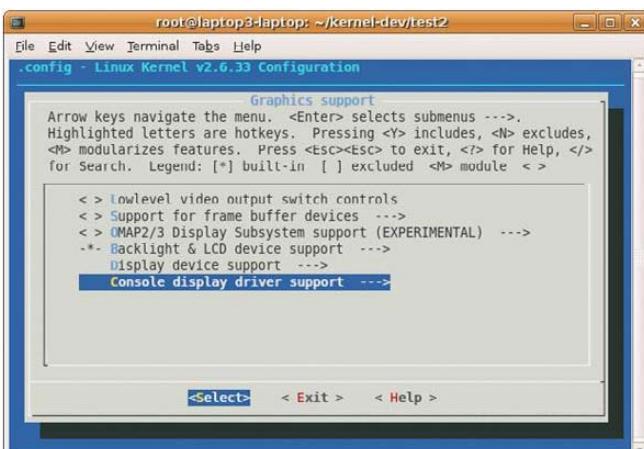


Figure 4. Deselecting the Graphics Support Options

article. Now, copy the default config for the Gumstix from OpenEmbedded into the production directory as the .config file. The purpose of this is to configure the basic structure of the Gumstix before adding the DSPBridge libraries. The command should look something like this:

```
cp <path to openembedded>/org.openembedded.dev/recipes/
  ➔ linux/linux-omap3-2.6.33/Overo/defconfig
<path to "production" directory>/.config
```

Next, issue the `make oldconfig` command, which is used to update the current .config file with new options. Generally, the default responses to the questions are acceptable, except in the highlighted cases shown in Figure 1.

Note that the ellipses in Figure 1 are used to indicate breaks in the output for display purposes. Now, issue the `make menuconfig` command, which will launch a configuration interface in the terminal, such as the one shown in Figure 2.

Navigate to the device drivers option (also shown in Figure 2), press Enter, and navigate to the Graphics Support option (Figure 3), and press Enter.

Now, deselect all the options in Graphics support by highlighting the option and pressing the N key. (The Backlight and LCD Driver support cannot be disabled, but all other options can be, as shown in Figure 4.)

These drivers, if left selected, will cause a compilation error while making the ulimage in the core directory of the kernel. By disabling them, the kernel should compile.

Next, issue the `make uImage` command and wait for the end of the compilation. This compiles a specially wrapped zImage that is used by u-boot during the booting processes for the Gumstix.

Then, issue the `make modules` command to compile all the kernel modules. This completes the compilation of all the necessary libraries to activate the DSP on the kernel side.

Now, compress the production directory with the compiled ulimage and the kernel modules in order to deploy on the Gumstix. A recursive copy will not work, because it will break several links inside the newly compiled ulimage and modules. The preferred method is to tarball the directory using `tar cvjf <production directory>.tar.bz2 <production directory>`.

Compiling the Userspace Files

Following the instructions on the DSPBridge Project (located at www.omappedia.org/wiki/DSPBridge_Project) for compiling the userspace files will work as long as the \$DEPOT, \$CROSS_COMPILE and the modified \$PATH variables are set in the terminal as mentioned previously. Issue the `make all` command to get the full DSPBridge project samples, MPU API and DSPBridge library.

Deploying the New Kernel to the SDRAM Card

The Gumstix bootable SDRAM card should be split into two partitions: one containing the MLO, U-BOOT.bin and ulimage, and the other containing the root filesystem (rootfs).

Using a bootable SDRAM card configured for the Gumstix Overo Fire, reformat the section with the current rootfs. Next, go to cumulus.gumstix.org/images/angstrom/factory/ and download the rootfs-booted-Overo-201004270808.tar.bz2

package for the Gumstix from the factory images. Now, uncompress rootfs-booted-Overo-201004270808.tar.bz2 into the rootfs partition on the SDRAM card.

After the rootfs-booted-Overo-201004270808.tar.bz2 package has been fully uncompressed, uncompress the production directory into the rootfs partition. If uncompressing the production directory into the partition created a new directory, move all the files out of the new directory into the same directory as the files with rootfs-booted-Overo-201004270808.tar.bz2. This should leave two directories inside the production directory: the usr and lib directories. Recursively copy the contents of these directories into the usr and lib directory of the uncompressed rootfs-booted-Overo-201004270808.tar.bz2. Now remove the ulmage-2.6.33 in the boot/ directory of rootfs-booted-Overo-201004270808.tar.bz2 and copy the new ulimage from the arch/arm/boot directory. Finally, change directory into the SDRAM card's bootable partition. Following the Gumstix site tutorial, this directory should be named FAT. Remove the ulimage and copy the new ulimage from the arch/arm/boot directory. Now unmount the SDRAM card and place it in the Gumstix and boot from the SDRAM card.

Deploying the DSPBridge Userspace Files

Once the new kernel has completed the boot sequence, add a password to the root user and secure copy the contents of the target/ directory of the userspace DSPBridge directories to the Gumstix. (The directories should be dspbridge/ and lib/.) Next, recursively copy the contents of the lib/ directory to the root /lib/ directory of the Gumstix and reboot the Gumstix. After the Gumstix reboots, enter the dspbridge/ directory and issue the ./ping.out command to receive the error shown in Figure 5.

This means the libraries have installed properly and the DSP device is detected. Now, load the base image for the DSP by issuing the ./cexec.out ddspbase_tiomap3430.dof64P and the ./cexec.out dynbase_tiomap3430.dof64P command. (To learn the difference, read the description at the bottom of the DSPBridge Project Wiki.)

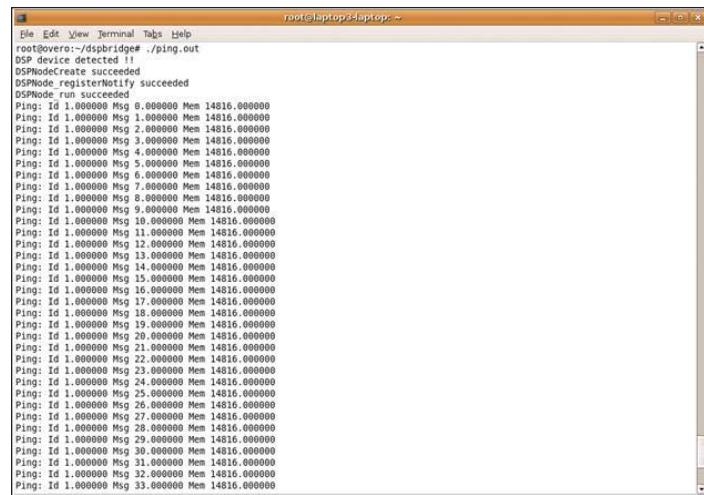
Next, register the base image with ./dynreg.out -r <sample>dyn_3430.dll64P for the test you want to run. For example, register the ping sample DSP program using the ./dynreg.out -r pingdyn_3430.dll64P command, and

This output proves the kernel has been compiled successfully with userspace files for the Gumstix Overo Fire and that the libraries have been integrated into the rootfs.



```
root@laptop3-laptop: ~
File Edit View Terminal Tabs Help
root@overo:~/dspbridge# ./ping.out
DSP device detected !!
node_create: failed to load create code: 0xfffffffac
DSPNode_Create failed: 0xfffffffacprocwrap_detach: deprecated dspbridge ioctl
c
root@overo:~/dspbridge#
```

Figure 5. DSP Error Message



```
root@laptop3-laptop: ~
File Edit View Terminal Tabs Help
root@overo:~/dspbridge# ./ping.out
DSP device detected !!
DSPNodeCreate succeeded
DSPNode_registerNotify succeeded
DSPNode_run succeeded
Ping: Id 1.000000 Msg 0.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 1.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 2.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 3.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 4.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 5.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 6.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 7.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 8.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 9.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 10.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 11.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 12.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 13.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 14.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 15.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 16.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 17.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 18.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 19.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 20.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 21.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 22.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 23.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 24.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 25.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 26.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 27.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 28.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 29.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 30.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 31.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 32.000000 Mem 14816.000000
Ping: Id 1.000000 Msg 33.000000 Mem 14816.000000
```

Figure 6. Results of ping.out

execute it with ./ping.out. The output should look like Figure 6.

This output proves the kernel has been compiled successfully with userspace files for the Gumstix Overo Fire and that the libraries have been integrated into the rootfs. Finally, this proves that the DSP is operational because of the response from the DSP to the ping program. The Gumstix with the newly compiled kernel now can be used as a test bed for projects involving the DSP. I hope this tutorial has helped in understanding and implementing the compilation of the Gumstix Overo Fire's Linux 2.6.33 with support for the DSP.■

James McCall is a Cadet majoring in Computer Science at the United States Military Academy at West Point, which was the best decision he made in college, and he's never looked back. Please direct comments to jim.mccoll11@gmail.com.

Resources

BeagleBoard/DSP Howto:
elinux.org/BeagleBoard/DSP_Howto

Gumstix Developer Center: gumstix.org

DSPBridge Project:
www.omappedia.org/wiki/DSPBridge_Project

ETH PIXHAWK: MAV Computer Vision Wiki Tutorials:
pixhawk.ethz.ch/wiki/tutorials/start



SHARE THE
PASSION

ELEVATE
YOUR GAME



SELF2011
SPARTANBURG SC
JUNE 10-12, 2011
SOUTHEASTLINUXFEST.ORG

Speech I/O for Embedded Applications

Is the world ready for speech-enabled embedded devices? Now the technology is here for usable speech recognition and synthesis. **See how you can use it in your own embedded applications.**

RICK ROGERS

Speech user interfaces are like the holy grail for computing. We talk to each other to communicate, and sci-fi stories—from HAL in *2001: A Space Odyssey* to the ship's computer in *Star Trek*—point to talking computers as the inevitable future. But, creating speech interfaces that are natural and that people will use has proven to be difficult. Too often speech technology is provided, or even preinstalled (as with Microsoft Windows Speech Recognition), and never used, but there are glimmers of hope. The technology to do “decent” speech recognition and speech synthesis has existed for a while now, and users are trying it out, at least in some application categories.

It feels like the opportunity is ripe for someone to get the speech interface right. Maybe you’re the one to invent a speech interface that makes your embedded application as cool and unique as the iPhone touch interface was when it first came out.

In some ways, embedded applications are particularly well suited for speech. An embedded device often is physically small and may not have a rich user interface. Almost by

definition, embedded applications are not general-purpose, so it’s okay if a speech interface has a limited vocabulary. Speech may be the only user interface provided, or it may augment a display and keyboard.

Mobile phones are one class of embedded applications where speech works as a user interface. Voice dialing (“dial home”) is almost a trivial interface that works very well on phones. If you’re driving and want to send a text message, it’s difficult (and in many places illegal) to use the phone’s soft keyboard to enter the message and its destination. Speech recognition is good enough, and mobile phones are powerful enough computers, that sending text messages by voice is a valid use case people are starting to employ.

In this article, I examine technologies for speech synthesis and recognition and see how they fit with today’s embedded devices. As an example application, and in step with the re-discovery of checklists as productivity tools (thanks to Atul Gawande’s best-seller *The Checklist Manifesto*), we’ll build a simple vocal checklist that you can use the next time you do surgery, like Dr Gawande (kids don’t try this at home).

Speech Technologies

As with any other user interface, a speech interface has two components: input and output (or recognition and synthesis). The two technologies are closely related, sharing techniques, algorithms and data models. As mentioned, speech has been a very popular computing research topic, and I can't cover all the work here, but I take a quick look at some different approaches, investigate some open-source implementations and settle on input and output packages that seem well suited for embedded applications. You don't have to be a computerized speech expert (I certainly don't claim to be) to speech-enable your embedded application.

Speech Synthesis or Text-to-Speech (TTS)

Naïvely, you might think "What's so hard about speech synthesis?" You envision a hashmap with English words as the keys and speech utterances as the values. But, it's not that easy. Any nontrivial TTS system needs to be able to understand things like dates and numbers that are embedded in the text and utter them properly. And, as any first-grader can tell you, English is full of words whose pronunciation is context-dependent (should "lead" be pronounced as rhyming with "reed" or "red"?). We also vary the pitch of our voices as we come to the end of a sentence or question, and we pause between clauses and sentences (called the prosody of the speech).

A lot of smart people have thought this over and have come up with a basic architecture for TTS:

1. A front end to analyze the text, replace dates, numbers and abbreviations with words, and emit a stream of phonemes and prosodic units that describes the utterance.
2. A back end, or synthesizer, that takes the utterance stream and converts it to sounds.

The front end, sometimes called text normalization, is not an easy problem. It's one of those pattern things that humans do easily and computers have a difficult time mimicking. The algorithms used vary from simple (word substitution) to complex (statistical hidden Markov models). For applications where the text to be spoken is relatively fixed (like our checklist), most TTS systems provide a way of marking up the text to give the normalizer hints about how it should be spoken (and, there is a standard Speech Synthesis Markup Language to do so; see Resources).

A variety of schemes have been developed to build speech synthesizers. The two most popular seem to be formant synthesis and concatenation.

Formant synthesizers can be quite small, because they don't actually store any digitized voice. Instead, they model speech with a set of rules and store time-based parameters for models of each phoneme. The prosodic aspects of speech are relatively easy to introduce into the models, so formant synthesizers are noted for their ability to mimic emotions. They also are noted for sounding "robotic", but very intelligible. For our chosen application, intelligibility is more important than "naturalness".

Concatenative synthesizers have a database of speech snippets that are strung together to create the final sound stream. The snippets can be anything from a single phoneme to a complete sentence. They are known for natural-sounding speech, although the technique can produce speech with distracting glitches, which can interfere with intelligibility, particularly at higher speeds. They

also are typically larger than formant synthesizers, due to the large database required for a large vocabulary. The database can be minimized if the TTS is for a domain-specific application, but, of course, that limits its usefulness.

Speech Recognition

In contrast to TTS, there is one dominant algorithm for speech recognition, hidden Markov models (HMMs). If you haven't run into HMMs before, don't expect me to explain the math in detail here, because, frankly, I don't completely understand it. I do understand the idea behind HMMs, and that's more than what we need to know to use an HMM-based recognizer.

If you sample a speech waveform (say every 10ms) and do some fancy math on the resulting waveform that extracts frequency and amplitude components, you can end up with a vector of cepstral coefficients. You then can model connected speech as a series of these cepstral vectors. A Markov model is like a state machine where the probability of a particular state transition is dependent only on the current state. In our case, each state of the Markov model corresponds to a particular vector, and as a Markov model moves probabilistically through its states, a series of cepstral vectors and sounds are generated. A hidden Markov model is one where you can't see the details of the state transitions, you just see the output vectors.

The trick is to create a bunch of these HMMs, each trained to mimic the sounds from a bunch of human-generated speech samples. Again, the math is beyond me here, but the process is to expose a training algorithm to a *lot* of speech samples for the language desired. As the sea of HMMs is trained, they take on the ability to reproduce the sounds they "hear" in the training samples.

To use the HMMs to recognize speech, we use one last bit of mathematical wizardry. For appropriate sets of HMMs, there are algorithms that, given a waveform (that is hopefully speech), can tell you: 1) which sequences of HMMs might have generated that waveform and 2) the probability for each of those sequences.

So, HMMs won't give us a definitive answer of what words the speech represents, but they'll give us a list from which to choose and tell us which is most likely and by how much. How cool is that?

Open-Source TTS

Many commercial TTS packages are available, but they don't concern us here. On the open-source side, there are still many candidates, with a few that seem more popular:

- eSpeak is the TTS package that comes with Ubuntu and several other Linux distros. It is of the formant flavor and, therefore, small (~1.4MB), with the usual robotic formant voice. The eSpeak normalizer also can be used with a diphone synthesizer (MBROLA) if desired, but we won't take advantage of that for the checklist example here.
- Flite is the embedded version of Festival, which is an open-source speech synthesis package originating at University of Edinburgh, with Flite done at Carnegie Mellon University. It is diphone-based concatenative, and as you would expect, it has a more natural voice. CMU also offers a set of scripts and tools for developing new voices, called FestVox.

Open-Source Speech Recognition

Most speech recognition packages are commercial software

for Windows and Mac OS X. I looked at two open-source speech recognition packages, both from the Sphinx group at Carnegie Mellon:

- Sphinx-4 is a speech recognizer and framework that can use multiple recognition approaches, written in Java. It is intended primarily for server applications and for research.
- PocketSphinx is a speech recognizer derived from Sphinx and written in C. As such, it is much smaller than Sphinx (but still around 20MB for a moderate vocabulary), and it runs in real time on small processors, even those with no floating-point hardware.

PocketSphinx is the obvious choice between the two implementations, so that's what we'll use here.

Atul—a Speech Checklist Application

In the interest of flexibility and speed, I've chosen a rather high-end embedded platform for the example program. The Genesi LX is a nettop with a rather generous configuration for an embedded device:

1. Freescale i.MX515 (ARM Cortex-A8 800MHz).
2. 3-D graphics processing unit.
3. WXGA display support (HDMI).
4. Multi-format HD video decoder and D1 video encoder.
5. 512MB of RAM.
6. 8GB internal SSD.
7. 10/100Mbit/s Ethernet.
8. 802.11 b/g/n Wi-Fi.

9. SDHC card reader.

10. 2x USB 2.0 ports.

11. Audio jacks for headset.

12. Built-in speaker.

On top of all that, there is a version of the Ubuntu 10.10 (Meerkat) distro available to load and run on the system, which makes installation and testing a lot easier. The download for Ubuntu and installation instructions are on the Genesi Web site. Installation from an SD card is straightforward through the U-boot bootloader.

Installing eSpeak

The eSpeak TTS system originally was developed for the Acorn RISC Machine (can you say "full circle"?), comes with Ubuntu and is included in the version for Genesi, so we get a pass there. That may not be true for your embedded system, but the installation procedure for eSpeak is straightforward and is given in the README of the download on the eSpeak site (see Resources). Of course, you'll need to do the install in the context of Scratchbox, or whatever native build environment you're using for your embedded Linux.

Installing PocketSphinx

To install PocketSphinx, you first need to install sphinxbase, which also is available on the PocketSphinx site. Both are tarballs, and the installation instructions are given in the READMEs. On systems like the Genesi, you can download and use the tarball to build the package. I did have to set LD_LIBRARY_PATH, so ld could find the libraries:

```
export LD_LIBRARY_PATH=/usr/local/lib
```

On smaller embedded systems, you might have to use a cross-compiler or Scratchbox.

The Atul Program

We want to create a general-purpose spoken checklist program along the lines of the checklists discussed in Dr Gawande's book. As an example, let's use part of the World Health Organization's Surgical Safety Checklist.

Let's create a speech checklist program that reads a checklist and listens to a reply for each item. We'll just match the reply with some valid ones now and record it in a file, but this could be a springboard for your own innovative speech user-interface ideas.

PocketSphinx comes with an application called pocketsphinx_continuous that will do basic continuous speech recognition and print the results to stdout, along with a lot of information about how it performed the recognition. We'll create a small C program, atul.c, that uses the libespeak library to speak the checklist items. We will have piped pocketsphinx_continuous to atul, so atul can listen to the replies on its stdin.

The compilation command for atul will vary



Figure 1. Genesi EPIKA MX Smarttop

Before induction of anaesthesia

(with at least nurse and anaesthetist)

Has the patient confirmed his/her identity, site, procedure, and consent?
<input type="checkbox"/> Yes
Is the site marked?
<input type="checkbox"/> Yes
<input type="checkbox"/> Not applicable
Is the anaesthesia machine and medication check complete?
<input type="checkbox"/> Yes
Is the pulse oximeter on the patient and functioning?
<input type="checkbox"/> Yes
Does the patient have a:
Known allergy?
<input type="checkbox"/> No
<input type="checkbox"/> Yes
Difficult airway or aspiration risk?
<input type="checkbox"/> No
<input type="checkbox"/> Yes, and equipment/assistance available
Risk of >500ml blood loss (7ml/kg in children)?
<input type="checkbox"/> No
<input type="checkbox"/> Yes, and two IVs/central access and fluids planned

Figure 2. Surgical Safety Checklist, Part I

Listing 1. SafeSurgery.ckl

```
This is the Surgical Safety Checklist.
#
Before induction of anesthesia.
#
Has the patient confirmed his or her identity, site,
procedure and consent?
# yes | no
Is the site marked?
# yes | notapplicable
Is the anesthesia machine and medication check complete?
# yes
Is the pulse oximeter on the patient and functioning?
# yes
Does the patient have a known allergy?
# yes | no
Does the patient have a difficult airway or aspiration risk?
# no | yes
Is there a risk of more than 500 milliliters of blood loss?
# no | yes
Thank you, that completes this portion of the checklist.
```

If You Use Linux, You Should Be Reading **LINUX JOURNAL**™



Get *Linux Journal* delivered to your door monthly for 1 year for only \$29.50! Plus, you will receive a free gift with your subscription.

SUBSCRIBE NOW AT:
WWW.LINUXJOURNAL.COM/SUBSCRIBE

Offer valid in US only. Newsstand price per issue is \$5.99 USD; Canada/Mexico annual price is \$39.50 USD; International annual price is \$69.50. Free gift valued at \$5.99. Prepaid in US funds. First issue will arrive in 4-6 weeks. Sign up for, renew, or manage your subscription on-line, www.linuxjournal.com/subscribe.

Listing 2. atul.c

```

/*
 atul - a simple speech checklist for embedded systems
*/
#include <string.h>
#include <malloc.h>
#include <stdio.h>
#include <speak_lib.h>

espeak_POSITION_TYPE position_type;
espeak_AUDIO_OUTPUT output;
char *path=NULL;
int BuffLen=500, Options=0;
void* user_data;
t_espeak_callback *SynthCallback;
espeak_PARAMETER Parm;
FILE *ckfp; /* Checklist file pointer */
char *ckBuf; /* Checklist item buffer */
char *mtchBuf; /* Checklist expected response buffer */
char *srBuf; /* Speech rec buffer */
char *reply; /* Trimmed reply */
int bsize=100; /* buffer length for all buffers */
int next; /* flag - true if should go to next prompt */

char Voice[] = {"default"};
unsigned int size, position=0, end_position=0,
flags=espeakCHARS_AUTO|espeakENDPAUSE, *unique_identifier;

void recordreply(){
/* read lines from stdin, which are piped in
 * from pocketsphinx_continuous.
 * Valid responses look like:
 * <9 digits>: reply (7 or 8 digits)
 * Returns a trimmed reply as char *reply
 * no spaces in return
 */
int i, j;
    while (!feof(stdin)) {
        getline (&srBuf, &bsize, stdin);
        if (srBuf[9]!=':') continue;
        j=0;
        for (i=0; i<strlen(srBuf); i++) {
            if (isdigit(srBuf[i])) continue;
            if (srBuf[i]=='-') continue;
            if (srBuf[i]==':') continue;
            if (isspace(srBuf[i])) continue;
            if (srBuf[i]=='(') continue;
            if (srBuf[i]==')') continue;
            reply[j++] = srBuf[i];
        }
        reply[j] = '\0';
        break;
    }

int checkreply(){
/* returns true if reply matches
 * false if no match (try again)
 */
char *tryagain="Please answer ";
char *ans, *spBuf;

/* if template blank, just sleep 2 sec */
if (strlen(mtchBuf)==2) {
    sleep(2);
    return(1);
}
/* see if reply matches template */
recordreply();
printf("reply: '%s'\n", reply);
if (strstr(mtchBuf, reply)==NULL){
    /* no match - tell user what we're looking for */
    spBuf = (char *) malloc (bsize+1);
    strcpy(spBuf, tryagain);
    if (ans=strtok(mtchBuf+2, "|")){
        free(spBuf);
        return(0);
    }
}
}

```

depending on your development environment. The invocation is:

`pocketsphinx_continuous | ./atul SafeSurgery.ckl`

Let's keep the application simple by reading checklist items and commands from a text file, whose name we'll pass as an argument to the program. Let's mark commands with a # at the beginning of a line. If the # is followed by a number, let's pause that number of seconds (up to 9). We will record each item and the replies as text to stdout.

The espeak library depends on two development packages you'll need to load into your target development environment. Both are readily available as rpm or deb packages: portaudio-devel and espeak-devel.

The Safe Surgery Checklist file is shown in Listing 1, and Listing 2 shows the source code for atul.c.

The code isn't very complex, although in retrospect, it might

have been clearer in Python or some other language that is better than C at string manipulation. The main routine initializes the TTS subsystem and makes sure that phoenix_continuous is ready to catch replies and forward them to us. It then just cycles through the checklist file, reading prompts and comparing the replies with the acceptable ones it finds in the checklist file. If it doesn't find a match, it tells the user what it's looking for and asks again. One thing to note, the string trimming routine in recordreply() throws out all spaces, so if your checklist is looking for a multiword response, be sure to concatenate the words in the list (like "notapplicable" in our checklist above). Everything of note is recorded in stdout, which you might want to redirect to a log file.

From Here

We've barely scratched the surface of speech user interfaces, even for a checklist application. Depending on your embedded system,

```

        strcat(spBuf, ans);
    }
    ans = strtok(NULL, "|");
    while (ans!=NULL){
        strcat(spBuf, " or ");
        strcat(spBuf, ans);
        ans = strtok(NULL, "|");
    }
    espeak_Synth( spBuf, size, position,
                  position_type, end_position, flags,
                  unique_identifier, user_data );
    espeak_Synchronize();
    free(spBuf);
    return(0); /* repeat last prompt */
} else return(1); /* go to next prompt */

}

int main(int argc, char* argv[])
{
    printf("atul started.\n");

    /* allocate needed buffers */
    ckBuf = (char *) malloc (bsize+1);
    srBuf = (char *) malloc (bsize+1);
    mtchBuf = (char *) malloc (bsize+1);
    reply = (char *) malloc (bsize+1);

    /* open the checklist file */
    if (argc < 2) {
        printf("Usage: atul <checklist filename>\n",
               argc);
        return 0;
    }
    ckfp = fopen(argv[1], "r");
    if (ckfp == NULL) {
        printf("Unable to open checklist file: %s\n",
               argv[1]);
        return 0;
    }

    /* Initialize the TTS subsystem */
    output = AUDIO_OUTPUT_PLAYBACK;
    espeak_Initialize(output, BuffLen, path, Options);
    espeak_SetVoiceByName(Voice);

    /* Initialize speech recognition
     * piped in from pocketsphinx_continuous */
    while (!feof(stdin)) {
        getline (&srBuf, &bsize, stdin);
        if (strncmp(srBuf, "READY...", 8)==0) break;
    }

    /* Go through the checklist */
    next = 1; /* advance to next prompt */
    while (!feof(ckfp)) {
        if (next) {
            getline (&ckBuf, &bsize, ckfp);
            getline (&mtchBuf, &bsize, ckfp);
        }
        size = strlen(ckBuf)+1;
        espeak_Synth( ckBuf, size, position,
                      position_type, end_position, flags,
                      unique_identifier, user_data );
        espeak_Synchronize();
        fputs(ckBuf, stdout);
        next = checkreply();
    }
    fclose(ckfp);

    free(ckBuf);
    free(srBuf);
    free(mtchBuf);
    return 0;
}

```

you'd have to give the user a way to start and end the checklist app, and ideally you'd have a way of signaling to the user when the app is listening. PocketSphinx prompts with "Listening..." and is supposed to terminate on saying "goodbye" (that doesn't work for me—maybe it's my Texas accent?). The source code for PocketSphinx (labeled continuous.c) comes with the package, so you can experiment with it. There are many, many optimizations you could make to both speech recognition and synthesis, using restricted vocabularies, different voice databases and just tuning the parameters.

And, what about a more general, practical speech user interface for embedded devices? The tools are available—how creative can you be? ■

Rick Rogers has been a professional embedded developer for more than 30 years. Now specializing in mobile application software, when Rick isn't writing software for a living, he's writing books and magazine articles like this one. He welcomes feedback on this article at portmobileapps@gmail.com.

Resources

eSpeak: espeak.sourceforge.net/index.html

Carnegie Mellon University's Sphinx Group:
cmusphinx.sourceforge.net

Carnegie Mellon's Flite Page: www.speech.cs.cmu.edu/flite

Genesi EFIKA MX: www.genesi-usa.com/products/efika

World Health Organization Safe Surgery Checklist:
www.who.int/patientsafety/safesurgery/ss_checklist/en/index.html

CyanogenMod 7.0— Gingerbread in the House

CyanogenMod is a full-blown Android distro you can install on your phone, whether your cell-phone provider wants you to or not. SHAWN POWERS

In order to run the latest version of Android, users normally have to be using the “flagship” Google handset. Unfortunately, that often requires using a different cell-phone provider, and it limits purchasing options significantly. In fact, it limits options to a specific handful of devices like the Nexus One and Nexus S. Because the Nexus devices don’t have a slider keyboard and come on the T-Mobile network, they’re not a viable option for me.

Enter CyanogenMod.

For users comfortable with rooting their phones, CyanogenMod offers many, many more features than the stock ROMs available for supported handsets. I have an original Motorola Droid from Verizon, and if I stick with the stock ROM, I will be stuck with Android 2.2. The current version of CyanogenMod, version 7, includes the Android 2.3 operating system. Code-named “Gingerbread”, Android 2.3 includes all the latest bells and whistles from Google that normally would be available only on the Nexus S. Thanks to the CyanogenMod team, we can have those features now—features like:

- New on-screen keyboard with better responsiveness and a cleaner layout.
- Easier selection tools for copying and pasting.
- A new Marketplace application with a Web-based companion for installing apps from your desktop browser.
- Integrated VoIP calling.

Although the new 2.3 features are great, the real beauty behind CyanogenMod is the customization options. Some of the same functionality can be added to the stock ROM for your phone by adding a replacement launcher, but CyanogenMod includes most of the bells and whistles by default. Some of the more exciting features include:

- The ability to change the lock screen’s layout.
- Highly customizable ADWLauncher installed by default.
- Improved pull-down status bar with power options.
- Visual improvements like screen snapping shut when powering off, customizable virtual desktops, resizable widgets and more.

CyanogenMod is so customizable, it’s often frustrating to

Got Root?

The first step to installing a custom ROM like CyanogenMod is to “root” your phone. Rooting doesn’t change the way your phone behaves; instead it gives you superuser (sudo-like) access to the system. Once your phone is rooted, you can do things like install a custom ROM, overclock your CPU, set up wireless tethering (which may violate your cell-phone contract) and even do simple things that shouldn’t require root, like take screenshots.

Rooting an Android phone is usually extremely simple. Typically, all it takes is a simple search on Google for the model of Android phone you have along with the word “root”. For instructions on how to root several different model phones, sites like www.droid-life.com can be helpful as well.

Remember, “rooting” your phone doesn’t change very much on its own. It just gives you the ability to change things. With great power comes great responsibility, however, so be careful with your rooted phone. Just like with a desktop version of Linux, the power of root can get you into trouble!

show off, because it can look so drastically different from install to install. To answer the question, “What does CyanogenMod 7 look like?”, the best answer is truly, “However you want it to look!”

If you want to have the latest version of Android on your phone, but you don’t want to wait for the cell-phone provider to release an update, or if you have an older handset (like my original Droid) that likely never will see an update beyond Android 2.2, CyanogenMod is the tool for you. Most phones are supported after rooting, and even older phones perform well with Android 2.3, especially if you overclock them. For more details or to download the latest version of CyanogenMod, check out www.cyanogenmod.com. The easiest way to install it on your rooted phone is with ROM Manager, however. It’s a simple download from the Marketplace, and the free version includes support for the stable version of CyanogenMod.

Steve Kondik: the Man behind the Mod

While I explored CyanogenMod version 7, I got really excited about how quickly development in the Android world was progressing. I decided to contact Steve Kondik, CyanogenMod’s creator, and ask him a few questions. (Thanks to my programmer friend Russ Ryba for helping me come up with some of the more



Figure 1. This is the default home screen in CyanogenMod 7. The background is my own, from NASA's Astronomy Pic of the Day Web site.



Figure 2. The lock screen is extremely customizable, and it can be themed to look like SenseUI, Honeycomb or completely custom with added functionality.

programming-centric questions.)

SP: CyanogenMod is one of, if not the, most popular custom ROMs for Android phones. What motivated you to start the project?

SK: I've always tried to customize desktop Linux, trying to make things smoother and faster. When I learned about how the G1 worked, and that I was able to change anything I wanted, I started off with rebuilding the kernel with different tweaks. Then I realized that I could rebuild the whole system from source and flash it to the phone with everything working perfectly—that's when the doors opened up.

SP: Installing a custom ROM requires a rooted phone. Rooting your phone is something that cell-phone companies generally frown upon. Have you gotten any unfriendly correspondence from either a cell-phone provider or Google?

SK: Yeah, and it's still a gray area with regard to firmware and proprietary drivers. When CM started to get big, Google made me very aware of the line between the open-source code and the closed portions (like the Google apps, such as Gmail), so I had to stop including those. CM doesn't ship with those parts now, so we leave it in the users' hands on how to get that functionality back.

It seems like carriers and manufacturers are starting to realize that these are more than just phones, and some even are embracing the community by releasing their own code as open source. T-Mobile recently open-sourced its theme engine, which we are using in CM7. Qualcomm and TI release code for their reference boards. Now we are finally starting to see more devices that are unlockable out of the box.

SP: You support a huge number of devices; I'm curious to know how much of CyanogenMod needs to be ported specifically to an individual handset, and how much is generic across the board. Does the wide variety of devices make releases difficult?

SK: A lot of devices are similar, but there always are subtle differences. Usually it has to do with the secondary processors on these devices and offloading media encoding/decoding, or special hardware like cameras and GPS. In some cases, we've had to reverse-engineer various parts, but the major manufacturers have

Advertiser Index

CHECK OUT OUR BUYER'S GUIDE ON-LINE.

Go to www.linuxjournal.com/buyersguide where you can learn more about our advertisers or link directly to their Web sites.

Thank you as always for supporting our advertisers by buying their products!

Advertiser	URL	Page #
1&1 INTERNET INC.	www.oneanddone.com	70, 71, 73
ABERDEEN, LLC	www.aberdeencinc.com	C3
ALL ABOUT THE CLOUD	sii.a.net/aatc/2011	9
ARCHIE MCPHEE	www.mcphee.com	79
CLOUD COMPUTING CONFERENCE	www.opalevents.org/print/LJ	5
DIGI-KEY CORPORATION	www.digi-key.com	79
EMAC, INC.	www.emacinc.com	25
GENSTOR SYSTEMS, INC.	www.genstor.com	27
IXSYSTEMS, INC.	www.ixsystems.com	3
LINODE, LLC	www.linode.com	33
LINUX JOURNAL STORE	www.linuxjournalstore.com	7
LOGIC SUPPLY, INC.	www.logicsupply.com	29, 49
ULLABOT	www.lullabot.com	1
MICROWAY, INC.	www.microway.com	C2, C4
O'REILLY VELOCITY	velocityconf.com	13
RACKMOUNTPRO	www.rackmountpro.com	19
SAINt ARNOLD BREWING COMPANY	www.saintarnold.com	79
SILICON MECHANICS	www.siliconmechanics.com	21, 37
SOUTHEAST LINUXFEST	southeastlinuxfest.org	57
TECHNOLOGIC SYSTEMS	www.embeddedx86.com	39
USENIX ANNUAL TECHNICAL CONFERENCE	www.usenix.org/fedweek11	31
UTILIKILTS	www.utilikilts.com	79

ATTENTION ADVERTISERS

September 2011 Issue #209 Deadlines

Space Close: June 27; Material Close: July 5

Theme: Programming

BONUS DISTRIBUTIONS: USENIX Security, FOSE 2011

Contact Joseph Krack, +1-713-344-1956 ext. 118,
joseph@linuxjournal.com

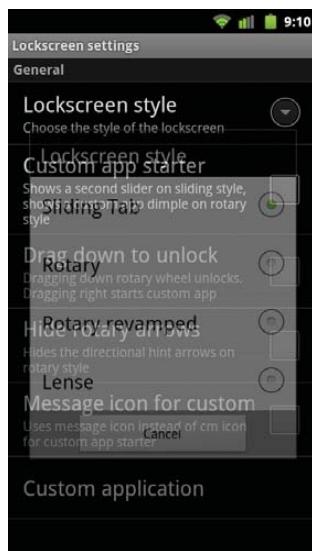


Figure 3. CyanogenMod has so many customizable options, it's easy to become overwhelmed. Thankfully, the default is beautiful.

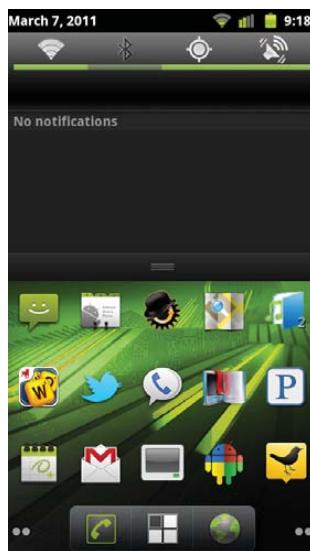


Figure 4. The pull-down menu has power modification tools available. Wi-Fi, Bluetooth, GPS and so on can be toggled quickly to save battery life.



Figure 5. With a few simple tweaks, the CyanogenMod home screen has a new theme, tightly packed icons, removed labels and more. If the included themes aren't to your liking, more are available on-line.

been providing open-source code that we can learn from or use outright, like Qualcomm's CodeAurora Project.

With CM, each device has a "maintainer", which is one or more people who work with that specific device and handle the issues.

SP: Once a handset is rooted, is installing custom ROMs simple, or do manufacturers try to prevent you from running custom code in other ways?

SK: It depends on the device. Some are wide open, and some are "rooted" but only in userspace—nothing can be changed in the boot process so things like custom kernels can't be used easily.

SP: How does the development process work with so many different developers and different handsets? Are there any unique frustrations with working on a project like yours?

SK: Keeping device-specific changes from breaking things on other devices is the tricky part. Most of this is handled by device "overlays" that customize the build for the hardware. Android has done a good job with making most of this relatively easy with its abstraction layers, but it also evolves rapidly. The extensible parts are where vendors often add their proprietary code and Android doesn't do anything for backward compatibility. In CM, we've actually had to re-add support for some of these older drivers in order to move to newer versions of Android.

There's also the issue of the way Google does development internally. It releases the source code for new versions after doing all the development behind the doors, so when it gets into our hands, it takes a lot of time to figure out what's changed and how to re-integrate our custom features back into it.

SP: Would any of what you do be possible without the GPL requiring Google to release its source code?

SK: Android isn't GPL; it's Apache-licensed. The only code Google's required to release is the Linux kernel. But no, without the source code, the kind of things we are doing wouldn't be possible.

SP: What are your most favorite and least favorite hardware devices to work with? What makes them awesome or horrible?

SK: The Motorola devices have been the most difficult because of the huge amount of proprietary code they include, not to mention the signed bootloader. The Samsung phones also have been a challenge because they've done a lot of things differently. HTC devices are what we have the best support for, partially because they are all similar to devices that Google has worked on directly (G1, Nexus One). The HTC Evo has been my personal favorite to work on, because the hardware is unique and it was the first to include things nothing else had like dual cameras and HDMI video.

SP: How much and what kind of experience do people need to begin creating their own custom ROM?

SK: If you want to build CM yourself for a device, we provide detailed documentation. You just check out the source code and with a few simple commands, you'll have a flashable ROM. That was the goal—to ease the barrier to entry. If you have an idea for a feature, it's very easy to dive in and start coding.

SP: I'm not a programmer at all, but I love CyanogenMod, and I would like to help. How can someone like me contribute?

SK: Helping out on our forums, writing documentation and tutorials on our wiki, and telling everyone you know about it.

SP: What makes CyanogenMod different from the other custom ROMs?

SK: I can't even call it a "custom ROM" anymore; it's become more like an Android distribution. We have a whole infrastructure for submitting code and a great support network. I want it to be as easy as possible for people to get involved and add their own ideas. CM is really about the community. Although we do have a core team of developers, many of the best parts in CM just came from out of nowhere because somebody thought it would be a neat idea, and we've made it easy to run the code on your own device.

SP: Do you plan to expand CyanogenMod to tablet computers as they become more common?

SK: We're already supporting a few tablets like the Nook and Viewsonic G-Tablet. As more become available and Android 3.0 is released, you can be sure that we will be trying to make them better.■

Shawn Powers is the Associate Editor for *Linux Journal*. He's also the Gadget Guy for *LinuxJournal.com*, and he has an interesting collection of vintage Garfield coffee mugs. Don't let his silly hairdo fool you, he's a pretty ordinary guy and can be reached via e-mail at shawn@linuxjournal.com. Or, swing by the #linuxjournal IRC channel on Freenode.net.

For more details on CyanogenMod, and how you can contribute, visit www.cyanogenmod.com.

Tiny Core Linux

If you want to go back to those great old days of really lightweight Linux, give Tiny Core Linux a try and relive the joy of a bare-bones system. JOEY BERNARD

Several projects exist that purport to be small, run-in-memory distributions. The most popular probably is Puppy Linux. Puppy has spawned several variations, and I have used it several times myself on older machines. But, I have discovered one that bowled me over completely—Tiny Core Linux. This distribution is a totally different beast and fills what I think is as of yet an unfilled category.

To start, Tiny Core is tiny—*really* tiny. The full desktop version weighs in at approximately 10MB—this is for a full graphical desktop. Not many other options can deliver something like this. People of a certain age may remember projects like Tom's root/boot, or muLinux. Tiny Core fits somewhere in between those older floppy-based projects and “heavier” small distributions like Puppy.

Along with this full version, there is an even more stripped-down version called Micro Core, which weighs in at less than 7MB. This version provides a command-line interface for all of you text aficionados. Tiny Core is designed to be run completely, or partially, from RAM. This means the system can be very fast and responsive. You also can set up the system so that it is loaded fresh on every boot, which reduces the probability of cruft working itself into your system dramatically.

To get Tiny Core, download it as an ISO image, which can be burned to a CD or copied to a USB device. Basically, you can put it on anything bootable. When you boot it up, you get the full desktop in a matter of a few seconds—in a virtual machine on my Mac, it takes less than five seconds (Figure 1).

The default gives you a window manager (flwm, the Fast Light Window Manager), a set of custom tools and a terminal (aterm). Everything else is available as an installable package, using its own custom package system called the AppBrowser (Figure 2). At the



Figure 1. You are greeted with a nice, clean desktop on bootup.

time of this writing, 3,170 packages are available. Packages are being added constantly, and there are very clear instructions on how to create and add your own packages.

When you boot Tiny Core, you initially are dumped at a boot prompt (Figure 3). If you don't do anything, it times out and places you on the desktop. However, you can use boot codes, which have the form of `tinycore option1 option2 ...`. Some of these boot codes include:

- `tce={hda1|sda1}` — specify restore TCE apps directory.
- `waitusb=X` — wait X seconds for slow USB devices.
- `swapfile{=hda1}` — scan for or specify a swap partition.
- `base` — skip TCE and load only the base system.
- `xsetup` — prompt user for Xvesa setup.
- `text` — start up in text mode.
- `{cron|syslog}` — start various daemons at boot time.
- `host=XXXX` — set hostname to XXXX.
- `noautologin` — skip automatic login.
- `desktop=xyz` — use alternate window manager.

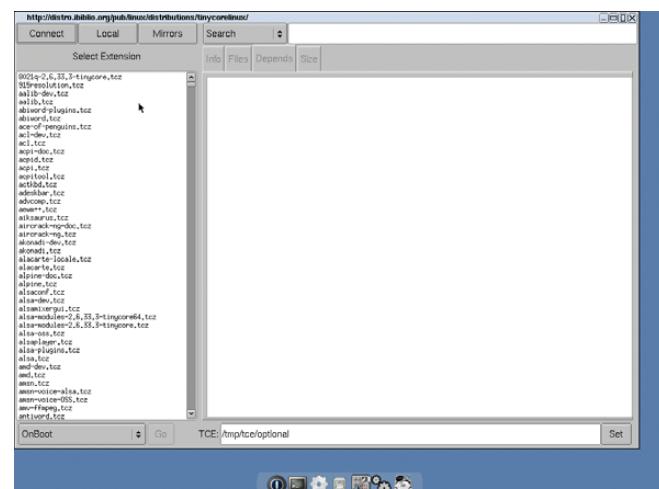


Figure 2. The packages available to you are listed after clicking on Connect.

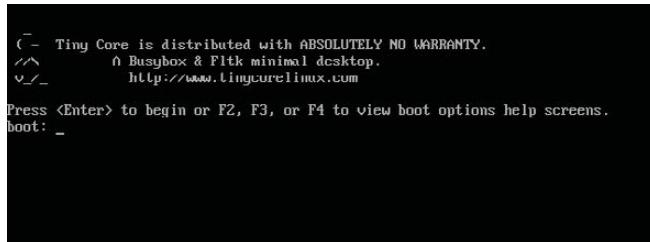


Figure 3. On bootup, you are greeted with a prompt where you can enter options to control your system setup.

Many other options are available. You can find them on the Tiny Core Wiki or list them during bootup. By default, you're logged in as user tc automatically and end up at the desktop with flwm as the window manager.

One of Tiny Core's features is that you get a fresh system on every boot. But, what if you want to save settings over a reboot? What are your options? In Tiny Core, you have the option to back up any necessary files at shutdown and have them be recovered automatically on boot. These files are saved to the file mydata.tgz. By default, the system saves all the files and directories that exist under /home/tc.

You can control what's actually backed up and what's ignored by using the files /opt/.filetool.lst and /opt/.xfiletool.lst. In .filetool.lst, you can add any files you want included in the backup. The file .xfiletool.lst contains a list of files to exclude from the backup. This backed-up home directory resides in RAM, so if you have a lot of files in your home directory, they will take up precious RAM. Also, as your home directory gets bigger and bigger, the startup and shutdown times grow as those files are being backed up and restored.

Another option is to create a persistent home directory. You can tell Tiny Core where to find this with the boot code home=xxx, where xxx is the device partition storing your home directory (for example, sda1 for the first partition on the first drive). If you want to put the home directory inside a subdirectory, you can hand this in with:

```
home=xxx/yyy
```

where yyy is the subdirectory name.

This gives you a really fast basic desktop, which is fine for everyday use. But, what if you want to adjust the distribution for some special case? To figure out how you can personalize it, let's take a step back and look at how Tiny Core is put together and how it works. Then you'll see how to change the system to suit your application.

The core part of the system is stored in a compressed filesystem that gets copied to RAM. Any extra applications are mounted from wherever they are stored as loopback devices, by default. They can be selected to be actually copied to RAM along with the core system, if you prefer. The advantage of this "run from RAM" system is that once the system has finished booting, you can remove the storage media.

In the first case, let's assume you have the system booting from a USB device that you don't need to remove. Then, you have two options on how to set up the system. The first, mount mode, is to create a directory called tce on the USB device. In this directory, you can dump packages for all the applications you want to have

available. These then are mounted as loopback devices. You can use a utility called appsaudit to maintain those packages. You have the choice of having those packages mounted at boot time, or you can have them mounted "on demand" (Figure 4). The other option is called copy mode. In copy mode, Tiny Core actually takes the contents of the package files and copies them all into RAM. This costs a bit more in boot time, but then the entire system is, again, running from RAM, so you get the expected speed-up once everything boots. You actually can control which packages are copied into RAM on an individual basis through the configuration file copy2fs.lst. The system can use this file to decide what is copied and what is mounted.

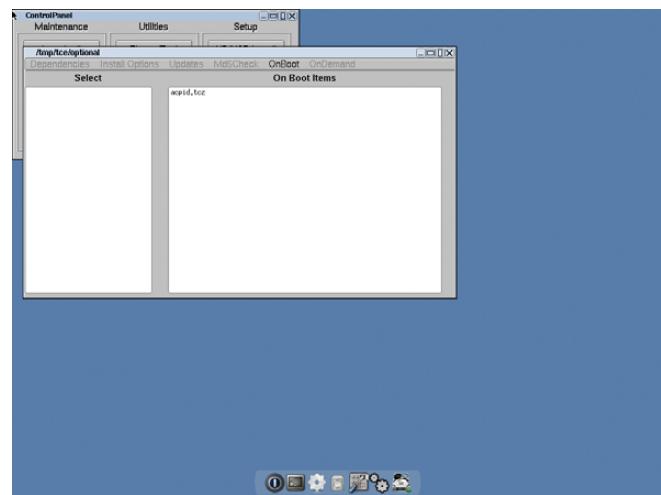


Figure 4. Maintaining your installed packages is made simpler with a GUI application.

These aren't the only methods available if you want to make a tailored distribution. Because Tiny Core is under the GPL, you can grab the source code and mess around as much as you please. You actually can just remaster the ISO to add in any extra packages you need for your application. In the ISO, there is a gzipped cpio archive named tinycore.gz. This file contains the core filesystem that is mounted in RAM when Tiny Core boots. You can do this work on any Linux box or even from within Tiny Core. If you want to do it in Tiny Core, you need to install a few extra packages before you start: advcomp.tcz and mkisofs-tools.tcz. Once you have all the tools you'll need, you can mount the ISO image:

```
sudo mount tinycore.iso /mnt -o loop,ro
```

where /mnt is the directory to which you want to mount. You also need a directory into which you can extract the Tiny Core filesystem, which for this piece, let's call it /temp/extract. To get the files, you need to execute the following:

```
cp -a /mnt/boot /temp
cd /temp/extract
zcat /temp/boot/tinycore.gz | sudo cpio -i -H newc -d
```

Once this command is done, you can go ahead and change files, add new ones or delete others. This way, you can add extra

binaries (such as for a point-of-sale application) directly as part of the system. If you want to be able to handle special hardware, where you'll need a new kernel module, you can add it to the filesystem. But, then you'll need to run this:

```
sudo depmod -b /temp/extract 2.6.29.1-tinycore
```

You also may need to add new shared libraries to provide support for any new binaries you install. If you do, run this:

```
sudo ldconfig -r /temp/extract
```

Once you've finished creating a personalized filesystem for Tiny Core, you need to get it ready to use. The first step is to pack the filesystem back up into a gzipped cpio archive. Run the commands:

```
cd /temp/extract find | sudo cpio -o -H newc |  
→ gzip -2 > /temp/tinycore.gz  
cd .. advdef -z4 tinycore.gz
```

This will give you a brand-spanking-new core file. If you are using a system other than a CD from which to boot (like some form of hard drive), you simply need to copy tinycore.gz and the kernel to that device.

If you want to create a new ISO image that you can use over and over again, execute the following commands:

```
cd /temp  
mv tinycore.gz boot  
mkdir newiso  
mv boot newiso  
mkisofs -l -J -R -V TC-custom -no-emul-boot \  
-boot-load-size 4 -boot-info-table \  
-b boot/isolinux/isolinux.bin \  
-c boot/isolinux/boot.cat -o TC-remastered.iso newiso  
rm -rf newiso
```

You now have a nice new ISO that you can put to work.

This kind of task happens often enough that the Tiny Core team has put together a GUI application that helps simplify these steps called ezremaster. Install it using the AppBrowser. This way, all the required dependencies also will be installed. You also need either to have the ISO available or a CD mounted. Once you've done all of these steps, open up a terminal and run ezremaster from the command line, and you should see what's shown in Figure 5. Here you can point it to the locations it needs, and you should end up with what's shown in Figure 6. From there, you can set all kinds of options to customize your ISO image. The sections available are:

- Boot codes.
- Display settings and mydata.tgz backup.
- Security settings and users.
- Which services would you like enabled?

- Network settings.
- Window manager, core elements and Xorg settings.
- ISOLINUX settings and 64-bit kernel.
- Startup and shutdown scripts.
- Extension installation.

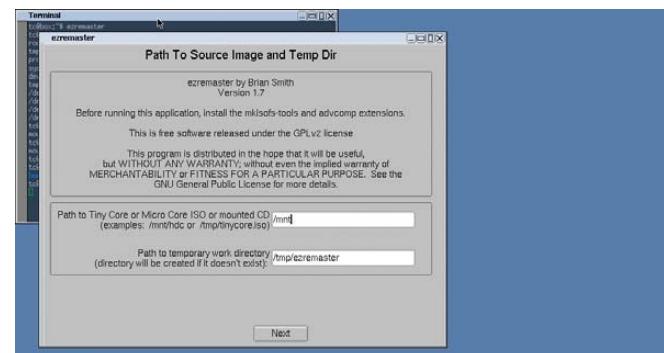


Figure 5. The first step when using ezremaster is setting paths for the source files and a working directory.

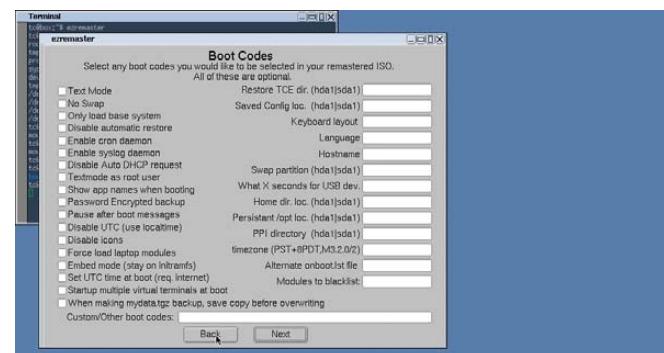


Figure 6. You can set default boot codes to save you extra typing on each boot.

Once you have finished all of these steps, move on to "Remaster step #1", where the filesystem for the new ISO is created. Once that step is done, move on to the last step, "Remaster step #2", where the actual ISO image is created. It ends up in the temporary directory you defined in the first screen. Now you're ready to deploy your awesome customized Linux on the world.

Be sure to check out the Tiny Core Web site and Wiki for more information (www.tinycorelinux.com). There is also a very active forum at the main site where people always are happy to answer questions. Hopefully, this project can give you a quick start for generating your own custom distributions for your smaller projects. ■

When Joey Bernard isn't debugging scientific code or sorting out problems on the clusters at the university, he's working on refinishing furniture or playing with his boys in the backyard. He might call himself a renaissance man, but that sounds a little too presumptuous. Just call him a well-rounded geek.

WEB HOSTING. TWICE AS SECURE.

1&1 DUAL H

Double Security, Double Reputability:

No one can afford downtime of their website...

1&1 is now offering dual hosting for the ultimate security of your website.

Your website is hosted in two different locations in our data center.

If the first location is unexpectedly interrupted, your site will automatically continue running in the second location – without any data loss.



MEMBER OF



NEW!

HOSTING

No other web host offers as much expertise, know-how and quality as 1&1: 1&1 combines over 20 years of web hosting experience with the latest technology in our high-speed and high-performance American data center. More than 1,000 IT professionals will continue to develop our top performance web solutions for years to come.

NEW: 1&1 is pleased to offer double security for your website with 1&1 Dual Hosting! All at unbeatably low prices!



Double Security: 1&1 Dual Hosting



Fast Global Network Connectivity: 210 GBit/s Connection



Top Performance: High-end Servers



Environmentally Responsible: 100% Renewable Energy



Solid Technical Foundation: Over 1,000 In-house Developers

1&1 DUAL UNLIMITED

- 3 FREE Domains
- FREE Private Domain Registration
- **UNLIMITED** Web Space
- **UNLIMITED** Traffic
- **UNLIMITED** FTP Accounts
- **UNLIMITED** E-mail Accounts (2 GB)
- **UNLIMITED** Mailing Lists
- 20 Microsoft® SQL Databases
- ASP, .NET, AJAX, LINQ, PHP, Perl, SSI
- GeoTrust® Dedicated SSL Certificate
- **NEW!** 1&1 SiteAnalytics
- 99.99% Uptime
- 24/7 Toll-free Customer Support

1&1 DUAL UNLIMITED

\$9.99

per month*
(36 month term)

\$11.99/month (24 month term)

\$12.99/month (12 month term)

\$13.99/month (3 month term)

Please see following page for more
1&1 DUAL HOSTING packages.

1&1



1-877-GO-1AND1

www.1and1.com



1-855-CA-1AND1

www.1and1.ca

Roll Your Own Embedded Linux System with Buildroot

The time between getting a new piece of hardware and seeing a first shell prompt can be one of the most frustrating experiences for embedded Linux developers. Buildroot can help reduce your frustration.

ALEXANDER SIROTKIN

It all started when I ordered an ARM-based development board for my FemtoLinux project, which is a Linux flavor specifically designed for ultra-small systems. Initially, I played with the idea of simply using a Linksys WRT router supported by an OpenWrt open-source project for development. But eventually, I decided that because it is a commercial project and development time is important, I was going to spend an extra \$100–\$200 for a real development board with official Linux support, which would come with everything that an embedded Linux developer would need: cross-compiler toolchain, Linux sources and embedded Linux distribution (at least, that's what I thought I would be getting). If you're on a budget and looking for a cheap embedded board for your hobby project, using a Linksys WRT router is not such a bad idea.

Choosing the right embedded Linux development board deserves an article of its own, but for now, suffice it to say that when you decide to use WRT, you should be prepared to build your software development environment yourself and expect to get support from the community. With a commercial board, I was expecting to receive it from the vendor, but I didn't. The vendor's idea of Linux support turned out to be just a list of kernel patches, forcing me to evaluate, choose and configure an embedded Linux development environment for this board by myself, which turned out to be quite an interesting and educational experience.

Embedded Linux Distributions

First, let's start with some basic terminology. An embedded Linux distribution is quite different from the PC distributions you are used to, such as Ubuntu or Fedora Core. It typically includes at least the following components:

- Cross-compiler toolchain for your target architecture, which is at least gcc, g++ and ld. It usually runs on one architecture, but produces binaries for a different architecture—x86 and ARM, respectively, in my case, hence the term cross-compiler toolchain.
- Kernel sources with a BSP (Board Support Package) for your board.
- Filesystem skeleton—that is, /bin, /etc with all the standard configuration files, such as /etc/fstab, /etc/inittab and so on.
- Applications—init and shell as a bare minimum, but most people will need more in order to do something useful.

Currently, the two most widely used embedded Linux distributions are OpenEmbedded and Buildroot. This article is about Buildroot, as that's the one I am most familiar with and naturally the one I used in my project. Buildroot's biggest advantage is its simplicity and flexibility, which are important if you are going to do some kernel hacking or other low-level development. If, on the other hand, you are an embedded application developer, OpenEmbedded certainly is a viable choice as well.

Buildroot

Even though you may not have heard of Buildroot before, it's actually not a new project. It has been around for many years, most of the time under the name of uClinux. Initially, uClinux was an effort to port the Linux kernel to processors without an MMU, such as the Motorola MC68328. However, it eventually expanded beyond that, adding support for more processors, a binary format for MMU-less systems and more userland capabilities, including a libc flavor specifically designed for low memory systems—uClibc. Eventually, it evolved into one of the more-advanced and easy-to-use embedded Linux distributions.

This is where the confusion started, as people used the name uClinux to refer both to the MMU-less CPU kernel support and the embedded distribution, which were two quite different things. The fact that many MMU-less patches (the whole armmmu architecture support, for instance) eventually were included in the standard kernel tree added to the confusion as well. Finally, the embedded Linux distribution part was split into a different project

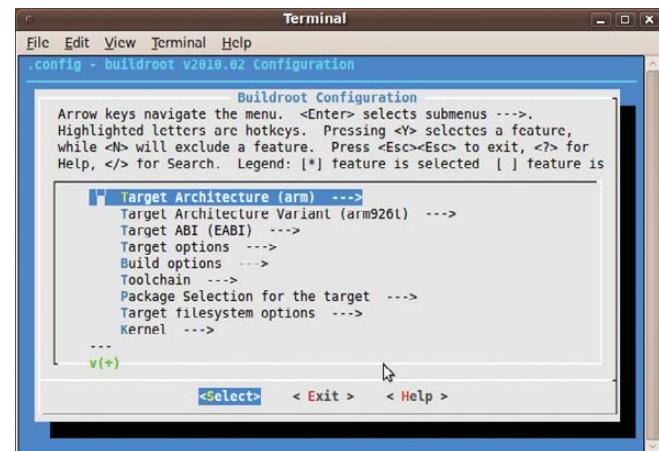


Figure 1. Buildroot Main Menu

1&1 DUAL HOSTING

SECURE & SAVE!

called Buildroot. uClibc development continued separately, and the parent uClinux Project somewhat lost its momentum.

From the Buildroot Web site: "Buildroot is a set of Makefiles and patches that makes it easy to generate a cross-compilation toolchain and root filesystem for your target Linux system using the uClibc C library." This is not entirely correct, as it also supports (to some extent, as you will see later) other libc flavors, such as glibc. It works with many embedded CPUs, including ARM, MIPS and PowerPC.

If you want to get started with Buildroot, download the tarball, extract it and run `make help` from its root directory. If this all looks familiar to you, wait till you run `make menuconfig`.

As you already may have guessed, Buildroot uses the same Makefile infrastructure as the Linux kernel to configure and build everything, including applications and libraries. The usual sequence of commands is:

```
make clean  
make menuconfig  
make
```

The first one is important if you are going to change some configuration parameters—incremental building may or may not work in this case. Initially, I was going to recommend that you start working with some default configuration, by running, for instance, `make integrator926_defconfig`, which should configure Buildroot for the Integrator ARM reference board. However, it turns out that as Buildroot development moved forward, most of the default configurations somehow lagged behind and currently do not work out of the box. I suggest you run `make menuconfig`, and choose the following options manually:

- Target architecture: arm.
- Target Architecture Variant: arm926t.
- Kernel: same version as Linux headers.

And, go over the other parameters and check for others that you may want or need to modify. Be careful when you do so, and always save your latest working configuration (the .config file). It is very easy to end up with a nonworking configuration.

Buildroot configuration options can be divided roughly into hardware-, build-process- and software-related, while software-related options can be divided further into kernel, toolchain and packages.

Hardware options are the "Target Architecture" that defines your CPU core (ARM, MIPS and so on). "Target Architecture Variant" defines the exact CPU you are using, and "Target Options" defines board-related parameters, such as UART baud rate and so on. You hopefully should know your hardware parameters, and there is not much to add here, except that for the ARM architecture, I suggest using EABI and making sure you use the same ABI convention everywhere.

If you are running Buildroot for the first time, you probably should avoid changing the "Build options". These options probably are okay the way they are; the only thing you may want to

1&1 DUAL BASIC

- 1 FREE Domain
- 10 GB Web Space
- UNLIMITED Traffic
- NEW! 5 FTP Accounts
- NEW! 1&1 SiteAnalytics

\$2.99
per month*
(36 month term)

\$3.99/month (24 month term)
\$4.99/month (12 month term)
\$6.99/month (3 month term)

1&1 DUAL ADVANCED

- 2 FREE Domains
- 150 GB Web Space
- UNLIMITED Traffic
- NEW! 50 FTP Accounts
- NEW! 1&1 SiteAnalytics

\$4.99
per month*
(36 month term)

\$5.99/month (24 month term)
\$6.99/month (12 month term)
\$8.99/month (3 month term)

.com

Now only \$7.99/first year*

.biz

Now only \$3.99/first year*

1&1 DOMAINS

Starting at

\$3.99
first year*

FREE Private Registration!

*Monthly dual hosting prices based on 36 month upfront billing term, for a total of \$107.64 for Basic and \$179.64 for Advanced package. No refunds. Domain offers valid first year only. After first year, standard pricing applies. Visit www.1and1.com for full promotional offer details. Program and pricing specifications and availability subject to change without notice.

More special offers
available online.



www.1and1.com



www.1and1.ca

1&1

®

BSP

BSP stands for Board Support Package. The term is somehow associated with RTOSes, such as VxWorks. Therefore, some people prefer the more “politically correct” LSP (Linux Support Package). Anyhow, the BSP is a set of usually small kernel and bootloader modifications specific to your hardware. Intel x86 developers take for granted that all x86 systems have the same basic hardware and peripheral interface, which is not the case on embedded systems. BSP development usually includes fixing memory mappings, configuring interrupt controllers and development of at least the following basic drivers: serial (for console), network and Flash.

change is the “Number of jobs to run simultaneously” if your build PC is a multicore system. Also, choose “build packages with debugging symbols” if you want to debug some of the pre-installed applications.

Remember, in order to build the kernel and software packages, Buildroot first needs to build the cross-compiler toolchain for your hardware. The Toolchain menu allows you to choose the gcc

You can start by creating a filesystem with just BusyBox. It contains everything you need in order to boot and verify that your system is working.

version and other toolchain-related parameters. The wrong toolchain configuration can lead to some very weird errors, so be careful. By default, Buildroot builds its own toolchain and works with uClibc. There is an option to work with an external toolchain, which can be glibc-based, but that’s beyond the scope of this article, so you should set “Toolchain type” to “Buildroot toolchain”. You can change the gcc, binutils, uClibc and kernel headers (but not the kernel itself) versions from this menu. You also can decide to compile the C++ (g++) compiler and gdb support (gdbserver for the target and gdb client for the host or a standalone gdb for the target), which is probably something you are going to need. All the other options are better left alone at this stage.

“Package selection for the target” is where you get to choose what software components you want as part of your embedded filesystem image. This is where you can experiment relatively freely—even if you select an application that’s not supported on your hardware or with the particular Linux and gcc versions that you chose, it’s easy to find the problematic application and disable it.

First, there is BusyBox. It deserves an article of its own, but basically, it’s a collection of standard Linux utilities (such as shell and init), optimized for low memory footprint systems. You can start by creating a filesystem with just BusyBox. It contains everything you need in order to boot and verify that your system is working. Later, you can add more packages, ranging from the MySQL or SQLite databases to the VLC and MPlayer media players, as well as Perl, Python and many others.

The “Target filesystem options” allow you to choose the type of filesystem image. Pretty much all the commonly used (in the

embedded world) filesystems are supported, including: cramfs, squashfs, jffs2, romfs and ext2.

If you just want to experiment or prefer to create the filesystem image manually (if you are using some rare unsupported filesystem, such as yaffs2), you can choose the “tar the root filesystem” option, which will create a tar archive with your filesystem. For some unknown reason, bootloader configuration also is found under this menu (only Das U-Boot is supported for now), but I’ll skip this one, assuming you have a working bootloader already.

The last menu is “Kernel”, which is optional. In case you are interested only in application development, choosing the right kernel headers (see above) is enough. If you decide to modify the kernel, remember to keep the kernel version and the kernel headers version (in the Toolchain menu) in sync.

When you are finished, exit menuconfig, and run `make`. Buildroot automatically will download everything it needs, compile it and eventually create the filesystem image in the output/images/ directory. If you want to modify something in the filesystem image, for example, to change the IP address of your system, you can modify the filesystem skeleton directory tree, which is usually located in `target/generic/target_busybox_skeleton`. Note that if you are not using BusyBox, or if your hardware platform has its own filesystem tree skeleton, this location can be different.

uClibc, BusyBox and Kernel Configuration

When you gain enough experience with Buildroot and decide you are brave enough to modify some of the uClibc, BusyBox and/or kernel parameters, the way to do it is to compile Buildroot with default settings for all three, and after that, run the following commands to modify the parameters and eventually recompile everything:

```
make uclibc-menuconfig
make busybox-menuconfig
make linux26-menuconfig
```

Note that the last one will work only after you enable the Linux kernel option in the main Buildroot configuration menu. Chances are that you already know how to configure the kernel, and uClibc configuration rarely requires tweaking, unless you want to compile out some functionality in order to save memory, so I’m going to look at BusyBox configuration only.

The BusyBox menu can be divided into settings and applets. I concentrate on the latter, as that’s probably what you would want to modify first. Applets are applications in BusyBox parlance, with one small difference. In order to save space, BusyBox usually is installed as a single binary that includes all the utilities you decided to compile: shell, ping, gzip and so on. You can launch an individual applet either by giving its name as an argument to BusyBox—`busybox ping`, for instance—or you can create a symbolic link, `ln -s /bin/ping /bin/busybox`, and BusyBox will choose the correct applet automatically, depending on the link from which it was executed. BusyBox installation automatically creates links for all

ARM ABI

An Application Binary Interface (ABI) describes the low-level interface between an application and an operating system and hardware. ARM Linux supports Old ABI (OABI) and Embedded ABI (EABI). OABI is deprecated, and it is recommended that you use EABI. As this parameter affects the kernel, the compiler and the standard libraries, it is important to use the same ABI everywhere, even though mixing ABIs may be supported. Compared to OABI, EABI defines a more-efficient system call convention, improves floating-point performance, changes structure packing, removes the minimal four-byte size limitation and some other minor improvements.

the compiled applets. If you are curious, you can run it without any parameters to see what applets were compiled in. You should have no difficulty in choosing the right set of applets for your project. The only thing worth mentioning is the shell. BusyBox does not support standard shells such as bash or tcsh; instead, you get to choose between ash, hush and msh with ash being the closest to bash and the one I always work with. Note that even though standard bash is not part of BusyBox, it is supported by Buildroot if you need it.

When you are finished configuring your embedded system, run make to compile everything. Now you are ready to program your newly compiled kernel and filesystem images to your board and boot. Actual Flash programming depends on your system, bootloader, type of Flash and so on, and it is beyond the scope of this article.

If you want to compile your own applications, you can (and should) use the toolchain created by Buildroot. You can get (or build) a different toolchain, but if it is not based on uClibc or if it was compiled with different kernel headers, it may not work. All you have to do in order to use the Buildroot toolchain is add the output/staging/usr/bin/ directory to your path and then simply run arm-linux-uclibcgnueabi-gcc.

The important point to remember is that Buildroot is not fool-proof in the sense that it is relatively easy to create a configuration that won't work or even compile. You should not expect every

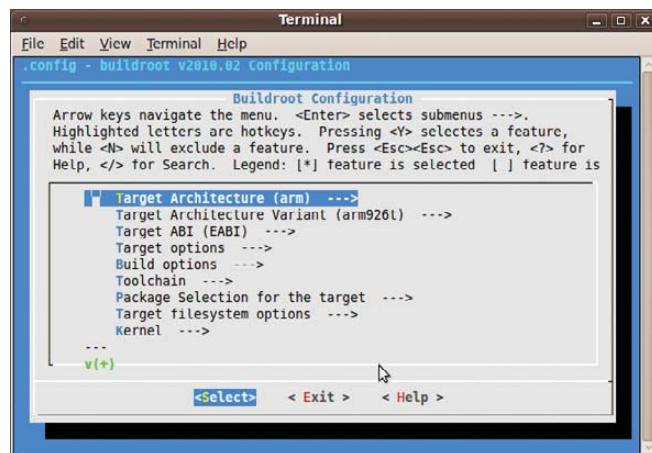


Figure 2. BusyBox Configuration Menu

parameter combination to work, and always keep your last working configuration file. The upside is that there is a large and active community behind this project, which will be happy to help.■

Alexander (Sasha) Sirotkin has been an active Linux user and developer for more than 15 years. One of the projects he's worked on is FemtoLinux, which improves performance on low-end embedded systems and eases porting from legacy RTOSes. He lives in Tel-Aviv, Israel, and can be reached at "sasha AT femtolinux.com".

Resources

FemtoLinux: femtolinux.com

uClinux: www.uclinux.org

uClibc: www.uclibc.org

Buildroot: buildroot.uclibc.org

OpenEmbedded: www.openembedded.org

Linux News and Headlines Delivered To You

Linux Journal topical RSS feeds available



http://www.linuxjournal.com/rss_feeds

A Primer to the OAuth Protocol

OAuth uses digital signatures rather than the “Basic” authentication method used by the HTTP protocol. ADRIAN HANNAH

During the past several decades, Web pages have changed from being static, mostly informational tools to full-blown applications. Coinciding with this development, Web developers have created interfaces to their Web applications so that other developers could develop applications to work with the Web application. For instance, think of any application on your phone for a Web service. This is possible only because of the application programming interface (API) constructed by the Web service's developers.

An API allows developers to give others access to certain functionality of their service without losing control of their service or how it behaves. With the development of these APIs arose the issue of user authentication and security. Every time you want to do something with the service, you have to send your user credentials (typically a user ID and password). This exposes the user to interested parties and makes the authentication untrustworthy. The application used by the user also could store the password and allow another application or person access to the user's account.

Enter OAuth.

OAuth is intended to be a simple, secure way to authenticate users without exposing their secret credentials to anyone who shouldn't have access to them. It was started in November 2006 by Blaine Cook, who was working on an OpenID implementation for Twitter. While working on it, Blaine realized that the Twitter API couldn't deal with a user who had authenticated with an OpenID. He got in touch with Chris Messina in order to find a way to use OpenID with the Twitter API. After several conversations with a few other people later, OAuth was born. In December of that year, OAuth Core 1.0 was finalized.

You can think of OAuth like an ATM card. Your bank account (the Web service) has a load of services associated with it, and you can use all of them, provided you put your card in the ATM and enter your PIN. Ultimately, anyone who has your card and PIN has full access to your account and can use all those neat services to do whatever he or she wants to your account. However, you can use your card as a credit card as well, and in that case, replace your knowledge of the PIN with a signature. In this capacity, the cardholder can do only very limited transactions, namely make charges against the

Since August 31, 2010, all third-party Twitter applications are required to use OAuth.

balance of the account.

If someone were to try to use your signature to charge something to your account without your card, it wouldn't work. If you had the card but not the signature, the same result would occur (theoretically). OAuth works in a similar manner. If an application has your signature, it can make API calls on your behalf to the Web service, but that signature works only with that application. Allowing one party to access someone else's resources on his or her behalf is the core of the OAuth protocol.

An Example of OAuth

Consider user Jane, a member of a photo-sharing site, photosharingexample.com (Service Provider), where she keeps all her pictures. For Christmas, she decides to give her mother some nice prints of her family, so she signs up for an account with another site called photoprintingexample.com (Consumer). The new site, photoprintingexample.com, has a feature that allows Jane to select pictures stored in her photosharingexample.com account and transfer them to her photoprintingexample.com account to be printed.

Photoprintingexample.com already has registered for a Consumer Key and Consumer Secret from photosharingexample.com:

Consumer Key: `dpf43f3p214k3103`
Consumer Secret: `kd94hf93k423kf44`

Jane elects to use this service. When photoprintingexample.com tries to retrieve Jane's pictures from photosharingexample.com, it receives an HTTP 401 Unauthorized error, indicating those photos are private. This is expected, because Jane hasn't authorized photoprintingexample.com access to her photosharingexample.com account yet. The Consumer sends the following request to the Service Provider:

```
https://photosharingexample.com/request_token?
→ oauth_consumer_key=dpf43f3p214k3103&oauth_
→ signature_method=PLAINTEXT&oauth_signature=
→ kd94hf93k423kf44%26&oauth_timestamp=
→ 1191242090&oauth_nonce=hsu94j3884jdops1&oauth_version=1.0
```

Using nonces can be very costly for Service Providers, as they demand persistent storage of all nonce values ever received. To make server implementations less costly, OAuth adds a timestamp value to each request, which allows the Service Provider to keep nonce values only for a limited time. When a request comes in

Nonce

The term nonce means “number used once” and is a unique and usually random string that is meant to identify each signed request uniquely.

with a timestamp that is older than the retained time frame, it is rejected, because the Service Provider no longer has nonces from that time period.

The Service Provider checks the signature of the request and replies with an unauthorized request token:

```
oauth_token=hh5s93j4hdidpola&oauth_token_secret=hdhd0244k9j7ao03
```

The Consumer redirects Jane's browser to the Service Provider User Authorization URL:

```
http://photosharingexample.com/authorize?oauth_token=
↪ hh5s93j4hdidpola&oauth_callback=
↪ http%3A%2F%photoprintingexample.com%2Frequest_token_ready
```

If Jane is logged in to photosharingexample.com, this page will ask her whether she authorizes photoprintingexample.com to have access to her account. If Jane authorizes the request, her browser will be redirected back to http://photoprintingexample.com/request_token_ready?oauth_token=hh5s93j4hdidpola, telling the consumer that the request token has been authorized. The Consumer then will exchange the Request Token for an Access Token using the following address:

```
https://photosharingexample.com/access_token?
↪ oauth_consumer_key=dfpf43f3p214k3l03&oauth_token=
↪ hh5s93j4hdidpola&oauth_signature_method=PLAINTEXT&
↪ oauth_signature=kd49hf93k423kf44%26hdhd0244k9j7ao03&
↪ oauth_timestamp=1191242092&oauth_nonce=
↪ dji430splmx33448&oauth_version=1.0
```

which will return the Access Token in the response:

```
oauth_token=nncch734d00s12jdk&oauth_token_secret=pfkdkdi9sl3r4s00
```

This exchange will happen only the first time Jane tries to access her photosharingexample.com photos from photoprintingexample.com. Any time afterward, only the



SUBSCRIBE TODAY!
WWW.LINUXJOURNAL.COM/SUBSCRIBE

following will happen.

Now, the Consumer is equipped properly to access Jane's photos. First, the Consumer needs to generate the request signature. The initial step is to create the Signature Base String. This is a combination of the following elements:

```
oauth_consumer_key: dpf43f3p214k3l03
oauth_token: nnch734d00s12jdk
oauth_signature_method: HMAC-SHA1
oauth_timestamp: 1191242096
oauth_nonce: kllo9940pd9333jh
oauth_version: 1.0
file: family.jpg
size: original
```

Ultimately, you end up with the string:

```
GET&http%3A%2F%2Fphotosharingexample.com%2Fphotos&
➥file%3Dfamily.jpg%26oauth_consumer_key%
➥3Ddpf43f3p214k3l03%26oauth_nonce%3Dkllo9940pd9333jh%
➥26oauth_signature_method%3DHMAC-SHA1%26oauth_timestamp%
➥3D1191242096%26oauth_token%3Dnnch734d00s12jdk%
➥26oauth_version%3D1.0%26size%3Doriginal"
```

If your request is being transmitted through SSL, the request can be in plain text. However, a vast majority of Web sites do not use SSL, so the signature string must be encoded.

Traditionally, the HTTP protocol uses an authentication method it calls "Basic" in which users provide their user names and passwords in order to gain access to the protected resource. The major flaw in that procedure is that those credentials are passed in plain text, clear for any people listening to read and store as they wish. In order to protect users' credentials, OAuth uses digital signatures instead of sending credentials with each request.

This digital signature is used to verify that the request being made is legitimate and hasn't been tampered with. A hashing algorithm is used to make that work. In order to allow the recipient to verify that the request came from the claimed sender, the hash algorithm is combined with a shared secret. If both sides agree on a secret known only to both parties, they can add it to the content being hashed. This can be done by simply appending the secret to the content, or by using a more sophisticated algorithm with a built-in mechanism for secrets, such as HMAC.

For this example, let's say the Service Provider allows HMAC-SHA1 signatures. Thus, the encoded signature string becomes:

```
tR3+Ty81lMeYAr/Fid0kMTYa/WM=
```

Hash Algorithm

The process of taking data (of any size) and condensing it to a much smaller value (digest) in a fully reproducible (one-way) manner. Using the same hash algorithm on the same data always will produce the same smaller value.

All together, the Consumer request for the photo is:

```
http://photosharingexample.com/photos?file=vacation.jpg&size=
➥original&oauth_consumer_key=dfp43f3p214k3l03&
➥oauth_token=nnch734d00s12jdk&oauth_signature_method=
➥HMAC-SHA1&oauth_signature=tR3%2BTy81lMeYAr%2FFid0kMTYa%
➥2FWM%3D&oauth_timestamp=1191242096&oauth_nonce=
➥kllo9940pd9333jh&oauth_version=1.0
```

The Service Provider performs the same work flow to calculate the signature of the request that the Consumer performs. It then compares its calculated signature to the provided signature. If the two match, the recipient can be confident that the request has not been modified in transit. The Service Provider then responds with the requested pictures.

This process can be daunting to deal with programmatically. There are a number of libraries written for both the OAuth server and client for quite a few programming languages.

Security Issues

The shared secret used to verify the request signature is called the Consumer Secret. Because it is vital to the integrity of this transaction, it is imperative that this piece of data be kept secret. In the case of a Web-based Consumer, such as a Web service, it is easy to keep the Consumer Secret safe. If the Consumer is a client-side application, the Consumer Secret must be hard-coded in each copy of the application. This means the Consumer Secret potentially is discoverable, which compromises the integrity of any desktop application.

There is a known session fixation attack vulnerability found in the OAuth 1.0 protocol that allows an attacker to gain access to a target's account. The attacker logs in to the Consumer site and initiates the OAuth authorization process. The attacker saves the authorization request page instead of clicking submit. This stores the request token and secret. The attacker sends a link to a victim, which, if clicked, will continue the authorization process as started by the attacker. Once completed, the attacker will have access to the victim's protected resources via the Consumer used.■

NOTE:
OAuth defines
three signature
methods used
to sign and
verify requests:
PLAINTEXT,
HMAC-SHA1
and **RSA-SHA1.**

Adrian Hannah is a lifelong system administrator, trying to find a nice place to finally settle down. He is currently working for the federal government in Indiana.

American made Utility Kilts for Everyday Wear
UTILIKILTS.com

Archie McPhee

Supplying the world with impractical weirdness for over 25 years!
mcphee.com

INNOVATION ON THE GO
 ORDER YOUR BEAGLE BOARD FROM DIGIKEY.COM

AVAILABLE EXCLUSIVELY AT DIGI-KEY
beagleboard

LOW-COST, NO FAN,
 SINGLE-BOARD
 COMPUTER

Digi-Key
 CORPORATION

www.digikey.com

SAINTARNOLD

FINE HANDCRAFTED BEERS & ALES

TOURS
 EVERY
 SATURDAY
 AT 1PM

WWW.SAINTARNOLD.COM

TEXAS'
 OLDEST
 CRAFT
 BREWERY

Whatever Sinks Your Boat

Why is our Internet slow today? DOC SEARLS AND DAVE TÄHT

Lately I've been urged by friends in the Linux community to write here about a topic dear to the infrastructure we share. So, rather than give away that topic in this intro, I'll turn the floor over to one of those worthy others: my old friend Dave Täht, who now will treat us to a guest EOF. Take it away, Dave.—Doc.

Do your movies stutter when you stream them? Does your kid get fragged when you fire off a Flickr upload? Can you not make a VOIP call while surfing the Web? These common problems may have one common cause: bufferbloat.

Although bufferbloat masquerades as inadequate network provisioning, it's actually a result of mis-design. To prevent packet loss, manufacturers have been putting vastly overlarge—bloated—buffers for data everywhere in the Internet: routers, switches, cable modems, wireless access points and other devices. This has badly worsened both average latency and latency under load—what you should think of as “speed” on the Internet. As a result, even after large increases in bandwidth, we often find we can't share a connection anymore. Movies stutter, calls drop, uploads interfere with gaming and so on.

In a string of painstaking and now well-publicized experiments, Jim Gettys has outlined (en.wordpress.com/tag/bufferbloat) the breathtaking, almost Y2K scope (mirrors.bufferbloat.net/Talks/BellLabs01192011), of the problem. He also coined bufferbloat (gettys.wordpress.com/what-is-bufferbloat-anyway) as the name for the pain. (Some of you may recall that Jim also originated the Unobtainium handheld.)

Jim's experiments showed, clearly, that even on high-speed 10–50Mbit lines, operations that should take 1/100th of a second might now take seconds. He also showed how the core protocol of the Web, TCP/IP, is now misbehaving, thanks to bufferbloat. The consequences might include widespread problems similar to the NSFnet collapse in 1986. And there

are a lot more people on the Net now than there were then.

The *RMS Titanic* hit an iceberg and sank because it was unable to turn fast enough to avoid disaster. The Tesla sports car carries two, goes from 0 to 60 in less than four seconds and turns on a dime. Which would you rather drive?

The *Titanic* is actually a more fitting analogy than you might think. The *Titanic*, like the Internet, was built during a major shift in technology. Steel was replacing iron. Nobody knew for sure what worked and what didn't. Bolting a giant ship together took advanced skills and advanced rivets. But, in the rush to launch that ship, essential risks were misunderstood and under-tested technology was pushed too far.

The Internet we've built has the carrying capacity and the turning speed of the *Titanic*. The great big bloated buffers we've built in to all the newest (and supposedly fastest) kit have been breaking the Net. Bufferbloat is the risk we now understand, and it's being tested now under increasing stress.

Jim Gettys is also no longer alone on the bufferbloat case. Since he sounded the alarm in November 2010, Robert Cringely, Slashdot and LWN have all covered the problem. Vint Cerf—a father of TCP/IP—put out a call for help at LCA as well.

Since then, many members of the Open Source and Internet engineering communities have leaped forward to help beat the bloat. As I write this (in early March 2011), more than 180 people have joined the bloat mailing list. In less than two months, we've also produced a new (debloat-testing) Linux kernel that puts many of the core ideas for fixes in one place.

Bufferbloat is a subtle bug that has been bugging everybody for a very long time, only we didn't know it. Now it has a name, plus a bunch of highly motivated people working on fixing it, from top to bottom, across multiple operating systems.

Much work and testing remain. There are already simple solutions for home routers out there, and more fixes for wireless and



other devices are on their way. Unfortunately, some problems still only have theoretical solutions (gettys.wordpress.com/2010/12/17/red-in-a-different-light).

While the network neutrality debate is over regulatory fixes to the threat of carrier favoritism toward certain kinds of traffic, a technological solution to the bufferbloat problem may turn down the heat a bit. It may be possible for your son's game, your wife's Facebook, your Flickr upload and your business calls all to co-exist happily on one network, and for content to travel much more smoothly through ISPs once good fixes for bufferbloat appear.

Best of all, fixing bufferbloat from end to end will make new edge applications feasible, from immersive video-conferencing to VRM.

Meanwhile, the bufferbloat problem remains huge, largely unrecognized, and it's all around us. Hundreds of millions of bloated products are in the field, and hundreds of millions more are in the pipeline. Fortunately, fixing new designs is fairly simple. Unfortunately, fixing already-deployed hardware is complicated and often expensive.

Can we turn our *Titanic* back into a Tesla, with a little trunk space? I'm betting: Yes! But we need all the help we can get.

There's still a shortage of good rivets, and good riveters.

If you design network software or hardware, use VoIP, upload/download video, play games, run a Web site, administer a network, purchase hardware, or merely care about the future of new, innovative applications on the Net, please don't stop reading here. Go to bufferbloat.net. Then read on, pass on, and apply what you learn.

The network you save may be your own. ■

Doc Searls is Senior Editor of *Linux Journal* and a fellow with the Center for Information Technology and Society at UC Santa Barbara.

Dave Täht is an IPv6 and mesh networking researcher who surfs (literally) on the side.



ANYONE INTERESTED IN SAVING MONEY?

Looks like these guys are comfortable overpaying for enterprise storage. Are You?

"Hewlett-Packard Co. agreed to buy 3Par Inc. for \$2.35 billion" — *Bloomberg.com*

"EMC to Buy Isilon Systems Inc. for \$2.25 Billion" — *Wall Street Journal*

"Dell to Buy Compellent for \$960 Million" — *CNBC*

So what "benefit" will you see by this spending spree, other than higher costs?

The AberSAN Z-Series scalable unified storage platform, featuring the Intel® Xeon® processor 5600 series, brings the simplicity of network attached storage (NAS) to the SAN environment by utilizing the innovative ZFS file system. The AberSAN Z20 is easily found starting under \$20,000.

Who gives you the best bang for the buck?

	3Par InServ F200	Compellent Storage Center Series 30	Isilon NL-Series	Aberdeen AberSAN Z20
Storage Scale-Out	✓	✓	✓	✓
Thin Provisioning	✓	✓	✓	✓
HA Clustering	✓	✓	✓	✓
VMware® Ready Certified	✓	✓	✓	✓
Async / Synchronous Replication	✓	✓	✓	✓
iSCSI / Fibre Channel Target	✓	✓	iSCSI Only	✓
Unlimited Snapshots	✗	✓	✓	✓
Native Unified Storage: NFS, CIFS	✗	✗	✓	✓
Virtualized SAN	✗	✗	✗	✓
Deduplication	✗	✗	✗	✓
Native File System	none	none	OneFS	ZFS 128-bit
RAID Level Support	5 and 6	5 and 6	Up to N+4	5, 6 and 10
Raw Array Capacity (max)	128TB	1280TB	2304TB	Unlimited
Warranty	3 Years	5 Years	3 Years	5 Years
Online Configurator with Pricing	Not Available	Not Available	Not Available	Available



Powerful.
Intelligent.

Above specific configurations obtained from the respective websites on Feb. 1, 2011. Intel, Intel Logo, Intel Inside, Intel Inside Logo, Pentium, Xeon, and Xeon Inside are trademarks of Intel Corporation or its subsidiaries in the United States and other countries. All other marks and names mentioned may be trademarks or registered trademarks of their respective holders. All rights reserved. For terms and conditions, please see www.aberdeeninc.com/lj038.

888-297-7409
www.aberdeeninc.com/lj038

Cut Execution Time by >50% with WhisperStation-GPU

Delivered ready to run new GPU-enabled applications:



WhisperStation with 4 Tesla Fermi GPUs

Design	Simulation	BioTech
3ds Max Bunkspeed Shot Adobe CS5	ANSYS Mechanical Autodesk Moldflow Mathematica	MATLAB ACUSIM AcuSolve Tech-X GPUlib

Integrating the latest CPUs with NVIDIA Tesla Fermi GPUs, Microway's WhisperStation-GPU delivers 2x-100x the performance of standard workstations. Providing explosive performance, yet quiet, it's custom designed for the power hungry applications you use. Take advantage of existing GPU applications or enable high performance with CUDA C/C++, PGI CUDA FORTRAN, or OpenCL compute kernels.

- ▶ Up to Four Tesla Fermi GPUs, each with: 448 cores, 6 GB GDDR5, 1 TFLOP single and 515 GFLOP double precision performance
- ▶ Up to 24 cores with the newest Intel and AMD Processors, 128 GB memory, 80 PLUS® certified power supply, and eight hard drives
- ▶ Nvidia Quadro for state of the art professional graphics and visualization
- ▶ Ultra-quiet fans, strategically placed baffles, and internal sound-proofing
- ▶ New: Microway CL-IDE™ for OpenCL programming on CPUs and GPUs

Microway's Latest Servers for Dense Clustering

- ▶ 4P, 1U nodes with 48 CPU cores, 512 GB and QDR InfiniBand
- ▶ 2P, 1U nodes with 24 CPU cores, 2 Tesla GPUs and QDR InfiniBand
- ▶ 2U Twin² with 4 Hot-Swap MBs, each with 2 Processors + 256 GB
- ▶ 1U S2050 servers with 4 Tesla Fermi GPUs

Microway Puts YOU on the Cutting Edge

Design your next custom configuration with techs who speak HPC. Rely on our integration expertise for complete and thorough testing of your workstations, turnkey clusters and servers. Whether you need Linux or Windows, CUDA or OpenCL, we've been resolving the complicated issues – so you don't have to – since 1982.



OctoPuter™ 4U Server with up to 8 GPUs and 144 GB memory

Configure your next WhisperStation or Cluster today!

microway.com/quickquote or call 508-746-7341

Sign up for technical newsletters and special GPU promotions at microway.com/newsletter

2U Twin² Node with 4 Hot-Swap Motherboards
Each with 2 CPUs and 256 GB



GSA Schedule
Contract Number:
GS-35F-0431N

Microway
Technology you can count on™