AUUGN

The Journal of AUUG Inc.
Volume 24 • Number 1
March 2003

Features:

News:

Regulars:

ISSN 1035-7521

Print post approved by Australia Post - PP2391500002

AUUG Membership and General Correspondence
The AUUG Secretary
AUUG Inc
PO Box 7071
Baulkham Hills BC NSW 2153
Telephone: 02 8824 9511
or 1800 625 655 (Toll-Free)
Facsimile: 02 8824 9522
Email: auug@auug.org.au

AUUG Management Committee
Email: auugexec@auug.org.au

President
Greg Lehey
PO Box 460
Echunga, SA, 5153
Bus. Tel (08) 8388 8286, Mobile 0418 838 708, Fax (08) 8388 8725
<Greg.Lehey@auug.org.au>

Immediate Past President
David Purdue
Sun Microsystems
Level 6, 476 St Kilda Road
Melbourne, Victoria, 3004
Phone: +61 3 9869 6412, Fax: +61 3 9869 6288
<David.Purdue@auug.org.au>

Vice-president
Malcolm Caldwell
Bus. Tel (08) 8946 6631, Fax (08) 8946 6630
<Malcolm.Caldwell@ntu.edu.au>

Secretary
David Bullock
0402 901 228
<David.Bullock@auug.org.au>

Treasurer
Gordon Hubbard
Custom Technology Australia Pty Ltd
Level 22, 259 George Street, Sydney NSW 2000
Bus Tel: 02 9659 9590, Bus Fax: 02 9659 9510
<Gordon.Hubbard@auug.org.au>

Committee Members
Sarah Bolderoff
FourSticks
Suite 2, 259 Glen Osmond Rd,
Frewville, South Australia, 5065
<Sarah.Bolderoff@auug.org.au>

Adrian Close
Mobile: +61 412 385 201, <adrian@auug.org.au>

Stephen Rothwell
IBM Australia, Linux Technology Center
8 Brisbane Ave, Barton ACT 2600
Business phone: 02 62121169
<Stephen.Rothwell@auug.org.au>

Andrew Rutherford
Iagu Networks, 244 Pirie St
Adelaide, SA, 5000
Bus. Tel (08) 8425 2201, Bus. Fax (08) 8425 2299
<Andrew.Rutherford@auug.org.au>

Mark White
apviva technology partners
P. O. Box 1870, Toowong QLD 4066
Bus Tel 07 3876 8779, Mobile 04 3890 0880
<Mark.White@auug.org.au>

AUUG Business Manager
Elizabeth Carroll
AUUG Inc
PO Box 7071
Baulkham Hills BC NSW 2153
<busmgr@auug.org.au>

Editorial

Con Zymaris <auugn@auug.org.au>

To those of us ensconced within the technical realms of our industry, economics, or the 'dismal science' as it's often belittled, nary merits a moment's thought during a day filled with debugging shell-scripts and scanning log files. However, as I'd like to demonstrate briefly, this is often to our detriment in understanding a large part of what makes us use the tools we use and what shapes the jobs we will be pursuing in years to come.

The IT industry is quickly reaching a maturation point whereby most of the hardware and most of the system-software and tool stacks will be commodities. Some of this works in the Unix community's favour, some does not. Here's the upside: 23 years ago, after an aborted mission to meet with DRI's Gary Kildall, (of CP/M fame) IBM rolled up to a meeting with Bill Gates to discuss the licencing of MS Basic for IBM's forthcoming entry into the PC industry. Gates, upon hearing that IBM was snubbed by DRI, offered a replacement OS for the new IBM PC. Gates & Allen then legally purloined Q-DOS (Quick and Dirty Operating System) from Tim Patterson of Seattle Computer Products, for the bargain price of $60,000.

With this acquisition, Microsoft was able to build an OS hegemony wherein Microsoft increasingly provided the *Intellectual Product* while the hardware stack was abstracted to near irrelevancy. It mattered not from whom you purchased your underlying hardware, as long as you purchase your OS from Microsoft. This process of hardware stack commoditisation has been relentless, and greatly benefits Microsoft as much as it usurps all hardware vendors belabouring under its yoke. How does this benefit our community? Simple, Linux and Open Source Unix platforms and tools are now doing to Microsoft's *Intellectual Product* stack what it had in turn done to the hardware. Linux *et al* are completely commoditising the system software and tools stacks. With time, this will likely have two major effects: it may squeeze Microsoft out of the mainstream OS platforms space (as too expensive) and provide respite to the hardware fraternity by allowing them to build more esoteric and specialised hardware systems, but which can still interoperate and run the same apps, as they all rely on the same core Open Source Unix system and tool stacks.

The first effect of economic commoditisation of the software stack looks like becoming a reality. A recent IDC report claims that this year, Microsoft will see its first real reduction in platform market domination, *ever*. The second effect is also becoming reality; numerous, formerly niche, players are now bringing out capable entries in specific market segments and winning business due to their adoption of Open Source Unix system and tool stacks. Two that I've seen introduced in recent months include Apple's X-Serve and SGI's Altix3000. Bull market for our sector!

The downside of this incursion of economics? Decreasing margins for those who choose not to play in this new reality. Cheers, Con

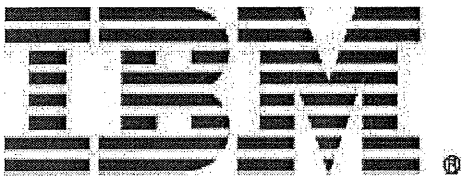
Contribution Deadlines for AUUGN in 2002

Volume 24 • Number 2 – June 2003: May 15th, 2003

Volume 24 • Number 3 – September 2003: August 15th, 2003

Volume 24 • Number 4 – December 2003: November 15th, 2003

AUUG Incorporated gratefully acknowledges the support of its corporate sponsor:



AUUGN Editorial Committee

The AUUGN Editorial Committee can be reached by sending email to: auugn@auug.org.au

Or to the following address:
AUUG Inc
PO Box 7071
Baulkham Hills BC NSW 2153

Editor:
Con Zymaris

Sub-Editors:
Frank Crawford, Mark White

Contributors:
This issue would not have happened without the transcription and editorial efforts of Gary R. Schmidt" <grschmidt@acm.org>, Rik Harris <rik@kawaja.net>, Raymond Smith <zzrasmit@uqconnect.net>, David Lloyd <lloy0076@adam.com.au>, Steve Jenkin <sjenkin@pcug.org.au>, Cameron Strom <c.strom@statscout.com>

Public Relations and Marketing:
Elizabeth Carroll

AUUGN Submission Guidelines

Submission guidelines for AUUGN contributions can be obtained from the AUUG World Wide Web site at:

www.auug.org.au

Alternately, send email to the above correspondence address, requesting a copy.

AUUGN Back Issues

A variety of back issues of AUUGN are still available. For price and availability please contact the AUUG Secretariat, or write to:
AUUG Inc
PO Box 7071
Baulkham Hills BC NSW 2153

Conference Proceedings
A limited number of copies of the Conference Proceedings from previous AUUG Conferences are still available. Contact the AUUG Secretariat for details.

Mailing Lists

Enquiries regarding the purchase of the AUUGN mailing list should be directed to the AUUG Secretariat.

Disclaimer

Opinions expressed by the authors and reviewers are not necessarily those of AUUG Inc., its Journal, or its editorial committee.

Copyright Information

AUUGN is the journal of AUUG Inc., an organisation with the aim of promoting knowledge and understanding of Open Systems, including, but not restricted to, the UNIX® operating system, user interfaces, graphics, networking, programming and development environments and related standards.

President's Column

Greg Lehey <Greg.Lehey@auug.org.au>

The last quarter has certainly been an interesting time for AUUG: we have managed to greatly increase our visibility in the press and the community, and for the first time since I have been with AUUG, we have noted an increase in memberships. Things are looking good.

The big story, of course, was our presentation to NOIE, the National Office of the Information Economy, on 22 February. I'll go into more detail about it elsewhere in this issue, but I'd like to take this opportunity to thank the "Open Computing in Government" committee, but particularly Gordon Hubbard and Con Zymaris, for their untiring effort in putting together the presentation for this event.

Apart from that, we also put in a submission to the House of Representatives Joint Committee of Public Accounts and Audit Inquiry into the Management and Integrity of Electronic Information in the Commonwealth. We don't have an official reply yet, but informal feedback has been that the submission has been well received. By the time you read this the submission should be on the web site. Thanks to Michael Paddon and his helpers for getting this submission out in a surprisingly short space of time.

Talking of the web server, the old and somewhat asthmatic machine has now been upgraded, which allows us to migrate more functions to it. In particular, we're working towards getting the membership database on the machine, which should allow you to update your membership details more easily. We're still considering the security implications, but we expect things to happen fairly soon.

It's not a coincidence that AUUG is doing well lately. As I've mentioned, we have a number of dedicated members who are helping to get and keep things moving. The number is still too small: AUUG will be as successful as its members make it. Why don't you consider becoming more active? There are plenty of things to do:

- The AUUG Board of Directors determines the direction of the organization. Along with this issue you should find an election nomination form. We're electing four officers (President, Vice-President, Secretary and Treasurer) and five "ordinary" board members. We're planning to have "job descriptions" on the web site by the time you read this, but in general as an ordinary board member you can expect to have to spend one day a quarter at a board meeting somewhere in Australia, and respond to mail messages on a regular basis. It's a lot of fun, and we'd like to see more applicants for the positions.

- In addition, we run a number of other events over the year. All of these events need people to help organize them, particularly people local to the event. The next one for this year is the systems administration symposium, which will take place in Melbourne on 9 April. By the time you read this, there shouldn't be much left to do for the systems administration symposium, though helpers on the day would be welcome. Contact Enno Davids <*enno.davids@metva.com.au*> if you'd like to help.

- We will hold the fifth Australian Open Source Symposium in Brisbane in July. Contact Mark White <*mark.white@auug.org.au*> if you'd like to help with this.

- We had also planned a security symposium in Adelaide for May, but unfortunately, we had to postpone it: we discovered that it clashed with the AusCERT symposium in the same week. In view of that, we've decided to postpone it to November. It's not clear that the people who were originally available at this time will still be able to do it in November, so we're looking for a number of people to help, including potentially the programme chair. Contact me if you're interested in helping here.

- Membership figures may be recovering, but the decline in AUUG chapter activity in recent years has not yet been reversed. The ACT and SA chapters have shown that this isn't necessary, and they're getting a lot done. The SA chapter held an Installfest in December, and the ACT chapter has held two seminars in this time, most recently the "Open Source in Government" symposium on 1 March. On the other hand, some other chapters are very quiet. They need people to get things moving again. If you know your local chapter committee, talk to them and offer to help. If you don't know them, contact Liz Carroll <*busmgr@auug.org.au*> or myself and we'll do what we can to help.

- The last issue of AUUGN included a call for papers for AUUG 2003, our annual conference. You can also find it at http://www.auug.org.au/events/2003/auug2003/cfp.html. The success of the conference depends greatly on the quality of the presentations. If you have a good idea for a paper, please send it to the programme committee <*auug2003prog@auug.org.au*>. If you're not sure, discuss it with the committee.

- Finally, this magazine needs contributors. Con Zymaris has done a fine job of finding interesting content, but we could do with more contributions from members. Do you have a good article in you, or do you think you might, or can you help putting the magazine together? If so, please contact Con <conz@auug.org.au> and discuss it with him.

/var/spool/mail/auugn

Editor: Con Zymaris <auugn@auug.org.au>

Where do you go for your Certificate Authority (SSL) requirements? I've found this place, which offers free certs: http://www.cacert.org/ AUUG members discussed others on the mailing list. Remember, to join in on such discussions, talk to the *mailman*: http://www.auug.org.au/mailman/listinfo/talk

From: Joel Sing <joel@ionix.com.au>
Subject: [Talk] SSL Certificates

Hi All,

Just a quick query - we're looking at acquiring an SSL certificate for use with a client's website and although Thawte and Verisign are 'reputable', they are *way* over priced for our client's needs.

InstantSSL (http://www.instantssl.com/), a spin off of Comodo Networks and QualitySSL (http://www.qualityssl.com/) both seem to be far better priced (starting at $49USD per year for a certificate) - anyone had experience with certificates from these companies? Any other companies that you would suggest checking out?

Thanks,

Joel

From: Conrad Parker <conrad@vergenet.net>
Subject: Re: [Talk] SSL Certificates

On Wed, Feb 12, 2003 at 04:19:42PM +1100, Ben Elliston wrote:
> I've often wondered why there isn't a volunteer-run
> grass roots SSL certificate authority. While not
> completely without flaws, it could work if there
> was a wide enough group of trustworthy individuals
> to perform various levels of identity checking. The
> certificate for the
> certificate authority could be included in major
> open source browsers
> like Mozilla and Konqueror.

ok, how would an organisation (like AUUG, Debian, or a local LUG) go about implementing this? ie. assuming we know how to issue certs, and we know how to sign GPG keys, what levels of identity checking would be required, how would the CA be managed, etc.?

if multiple such groups did this, could they form a web of trust between them in order to minimise the number of central/top-level CAs that browsers would be required to know?

Conrad.

From: Ben Elliston <bje@air.net.au>
Subject: Re: [Talk] SSL Certificates

On Thu, Feb 13, 2003 at 02:07:14PM +1030, Greg 'groggy' Lehey wrote:

> FWIW, this is one of the membership benefits that
> we have been talking about in the past. I think the
> board is agreed that this is a good idea, but we
> haven't found anybody to address the details,
> notably the trust issues you mention. There will be
> a board meeting next Saturday (22 February). If
> some volunteer could come up with a concrete
> proposal by then, we could talk about implementing
> it pretty quickly.

A CA is not ideally useful unless their cert is installed in browsers by default. Sure, we could send the CA cert to AUUG members on a CD by postal mail (probably good enough, IMHO), but this would only be limited to trusting AUUG and other AUUG members.

The problem I see is that open source browser projects are only going to be willing to include a handful of CA certs and if AUUG's only serves a couple of hundred users in the antipodes, there will be strong reluctance to include it.

Cheers, Ben

From: Duane <duane@sydneywireless.com>
Subject: Re: [Talk] SSL Certificates

Hello Andrew,

Currently I haven't tried to get the root cert included into mozilla, as others have tried and were asked for trading histories etc etc etc

I'm still trying to get time to finish the site off, with the users verifying users thing happening... it's been mentioned on slashdot numerous times and I've lost count of copies of the root cert downloaded in excess of 30,000 times last time I checked, and so far there has been almost 250 certificates issued to date...

I created the site because I was sick of paying for them, even for personal sites, and with blunders from verisign and verisign jacking the price of thawte certs, I sat down and replicated their system more or less as much as I could... Plans for faxed verification in the pipeline, but there would still be some sort of free certificates issued based on email verification (which is what some of the commercial certification issuers only check, apart from credit card payments)

--
Best regards,
Duane

mailto:duane@sydneywireless.com

This quarter's CD-R

Greg Lehey <Greg.Lehey@auug.org.au>

This quarter your AUUGN comes with OpenOffice (http://www.openoffice.org/) version 1.0.2. OpenOffice is an open source Microsoft-like office suite derived from StarOffice. As you can see from the label, it includes binaries for FreeBSD, Linux, MacOS X and Solaris, and also for Microsoft. We don't intend to continue distributing binaries for Microsoft--after all, we are a UNIX organization--but this time we made an exception so that we could distribute the same CD-Rs at the NOIE seminar described elsewhere in this AUUGN. In that connection, I'd like to quote from the ``Open Source'' column in The Australian of 25 February 2003, describing the NOIE seminar:

> *Despite the lack of a Microsoft-versus-Linux showdown, however, Open Source (no relation) also heard that ``certain people'' were not impressed with copies of OpenOffice being handed out to everyone who attended.*

I have it on good authority that ``certain people'' included at least Steve Vamos, Managing Director of Microsoft Pty Ltd.

Installing OpenOffice is relatively simple. There are briefing structions on the CD itself, and more detailed instructions in the top-level directory. Bring much space: it emulates Microsoft in its appetite for disk and memory as well.

NOIE's Open Source Seminar, February 2003

Greg Lehey <Greg.Lehey@auug.org.au>

On 18 February 2003, the National Office for the Information Economy held a seminar for the CIOs and CTOs of Government agencies to discuss the issues involved with the deployment of open source technology, notably Linux, in the government. As Australia's largest organization representing UNIX and open source software, AUUG was invited to present. Gordon Hubbard and Con Zymaris presented some slides, which you can find at http://www.auug.org.au/ocg/noie-feb2003.pdf, and I participated in the panel discussion.

There's been a lot of press coverage of this event; see the links at the end of the article. This article reflects my own personal view, little of which corresponds with the press releases.

First up was Mary Ann Fisher of IBM, who presented a paper which looked so much like ours that you could almost have thought that we had collaborated. In fact, it was no longer as similar as our first attempts. I have to concede to Con and Gordon that my objections to many of the slides weren't as relevant as I thought, since Mary Ann presented much of the material that I had asked to have removed. Bottom line was ``open source is good because it helps to implement open standards'', pretty much the same as what we said later.

Next came Maggie Wilderotter of Microsoft, who put up a good presentation saying not very much of substance except that Microsoft was committed to open standards.

Next were Gordon and Con, who put up our position pretty well. It's interesting to see how people see things differently. Con used Gordon's laptop to demonstrate the ``desktop'' environment, and in the process had some minor difficulties. While he did it, I looked at the attendees and decided that there wasn't enough interest to repeat this part of the talk next time. Later I was talking to Hugh Blemings, and he volunteered his appreciation for the demonstration and the way everybody watched with spellbound interest. Obviously my aversion to ``desktops'' is showing.

During lunch, spoke to Maggie Wilderotter and Steve Vamos, the latter Managing Director of Microsoft Australia. I commended them on their intention to pursue open standards (which we had defined in our talk as being maintained by industry consensus and available from more than one vendor), and asked when they proposed to start. Steve said ``Ah, but we have started. Do you know about .NET?''. sigh. I told him that we weren't too happy with .NET (in fact, it was in our presentation as an example of a non-standard system, but Steve had apparently not noticed that). At this point, Maggie disappeared and I carried on talking to Steve about it for a while. I was left with the distinct impression that Microsoft don't completely understand why we don't consider .NET to be an open standard. Still, they're interested in putting their point of view, so we may ask them to do so at the AUUG 2003 conference.

After lunch, a presentation by Peter Gigliotti of the Bureau of Meteorology, also an AUUG member, about what they're doing with Open Source. It's interesting, but not encouraging: much of what they're doing has been replacing proprietary UNIX, so the open standards issue is not touched, and they're still planning to migrate to Microsoft, though it's possible that might change.

Robin Simpson from Gartner came on next with a lively discussion full of numbers and probabilities, but without hard copies of the slides. One of the most interesting things he said was that, with 0.8 probability, the GPL will prove to be too restrictive by the end of 2004. I'll be interested to see if he's right there. He made some other more unlikely claims, including that forked projects can rejoin at a later date as long as license issues don't intervene. That sounds very unlikely to me. Also, he claims that there isn't enough information about TCO over a five year period to be certain that open source software would be cheaper than Microsoft. That sounds rather unlikely as well.

Then another talk about a government project, this time by Tony Ablong of the Department of Veteran's Affairs, describing how they propose to replace their

current IBM infrastructure with a zSeries 8000 running Linux under VM in LPARs. It's hasn't been implemented yet, unfortunately, which rather detracted from the message.

Then the panel discussion, in which I took part. This was also the only part where the press was allowed in. The questions were interesting in the sense that they showed where the participants were coming from, and also because they came mainly from non-Government people. The summary, which was also printed in ``The Australian'', was the uncertainty about TCO, quite a shame really. I don't see much change happening as a result of this seminar; if there's more uptake of open source, probably IBM will be the real motivator.

Another Perspective on the NOIE's Event

Jonathon Coombes <jon@cybersite.com.au>

The day started with most of the guests in attendance. The NOIE people were very informed on what was happening on the day and managed the proceedings very well. Talking with some NOIE members after the day they also showed great interest in pursuing further similar events.

The first speaker was from IBM (Mary Ann Fisher), and she spoke well and directed regarding IBM and there approach to open source. It worked in well with the AUUG presentation as Mary Ann's approach was more on the managerial side than the advocacy or technical side.

Next came Microsoft's speaker (Maggie Wilderotter), flown out from the US for this event. Many people expressed dis-belief initially, and then amazement at the approach Maggie took with her talk. Unlike previous seminars from Microsoft, Maggie had a very balanced approach and let out much of the marketing and promotion that is often associated with similar talks. Points presented included Microsoft's support for access to code, for open protocols and data format standards. Much of what Maggie presented was done in a way that presented Microsoft as supporting the open source community. The main area of disagreement was that of intellectual property and licensing restrictions. Maggie pushed the Shared Source Initiative as a viable alternative to the GPL and proposed that some secrecy was necessary for commercial benefit. The government security program was to allow the proposed government full access to the code, protocols and standards, although this is yet to qualified by other Microsoft team members.

I had the fortune to sit with some people from the government solicitor's office during the seminar, and the lack of legal responsibilities or quirks were a concern. I had the opportunity to talk with Maggie and questioned the liability associated with the Government Shared source initiative. Her response was that the code is kept very secure and that clauses were in the contract to account for liability. However, the extent of the liability was greatly varied depending upon how the source code was leaked, who was responsible etc. Maggie referred the question to Steve Vamos to follow up.

The AUUG presentation was well formulated, but had quite a lot of information to present. The option of the demonstration was good, but did really suite the situation as it turned out. The option was made for people to test-run the software after the talks. The actual content of the talk was concise and directly to the point. The aim of AUUG was to present the benefits of the open source community both in support and association, but also in business economics and management. Overall a well presented talk.

In the afternoon, following the light lunch, a number of case studies were presented as a practical representation of what Linux/OSS could do for the government. Although this was good in theory, unfortunately, it did not really live up to its potential. The first speaker was from the Bureau of Meteorology who gave a good rundown on what they used Linux for at the bureau, and how it had helped with one particular aspect of their work. The other case study was from the Dept of VA, which was based more around what benefits they expected to get from Linux, as they were still very much in the planning stage of implementing Linux. Both talks were more along management issues than benefits of open source, either technically or monetary.

The last main presenter of the day was Robin Simpson from the Gartner group. This speaker was presenting Linux/OSS from a market research prospective and it was obvious in his speech. Any major point was made with an associated probability. Generally though, his points were good for open source, although maybe more long term than the community hopes. I questioned Robin on some points he made regarding OpenOffice.org including its lack of support for templates, incompatibility with MS Office and lack of options. After talking with Robin, it turns out he was still in the process of testing the package, so hopefully we can see a better response after the testing is complete.

The final event of the day was with the panel. The panel consisted of all the speakers with Greg Lehey standing in for AUUG instead of Gordon or Con. The panel was of excellent make, but unfortunately the questions given them were not. Due to time restrictions there were only a few questions asked, and these were more around marketing and planning approaches than the open source benefit to government. I would have preferred to see the panel given more time, but unfortunately it was not to be so.

All in all, a great day and definitely of benefit to all people present. The only disadvantage was the lack of time as not all possible areas were covered, and some could have been covered better. Hopefully this seminar will be the start of many similar events to come that will help to raise the awareness of open source and what benefit it has for government and business alike.

Public Notices

Upcoming Conferences & Events

AUUG System Administration Symposium
April 9, 2003
Melbourne

USENIX '03
USENIX Annual Technical Conference
June 9-14
San Antonio, TX

AUUG Australian Open Source Symposium
July 2003
Brisbane

AUUG 2003
September 2003
Sydney

AUUGN Bookreviews

Section Editor: Mark White <mark.white@auug.org.au>

Tuning and Customizing a Linux System

by Daniel L. Morril. Apress 2002 (ISBN 1-893115-27-5)

Reviewed by Michael Still <mikal@stillhq.com>

I wasn't expecting much when I received my copy of Tuning and Customizing a Linux System to review. I had read the sole review on amazon.com, and it was very short of being complimentary. Having now actually looked at the book, I think that reviewer has completely misinterpreted the intention of the book. Part of the problem is the title of the book. Many people I suspect will read the title and think "Ah ha! A book to help me configure product X to be able to handle more load / users / gronks per hour". This isn't what Tuning and Customizing a Linux System is about at all -- the book is aimed at introductory to intermediate Linux users who are interested in exploring other distributions, and perhaps understanding a little more about how the operating system functions.

It is also worth noting that this is the first book of it's type I have seen where the author has taken the time to devote the first 25 pages of the book to attempting to explain the history of Linux, and the GNU philosophy. It's a reasonable attempt as well.

The book then goes on to talk about three distributions in detail: Red Hat Linux 7.3; Slackware Linux 8.0; and Debian GNU/Linux 3.0. The book discusses issues which apply to all Linux users, such as the kernel numbering conventions, the File system Hierarchy Standard (see http://www.pathname.com/fhs/ for more details), and the Linux Standards Base (http://www.linuxbase.org/).

As a long time Red Hat user, the chapter on Red Hat 7.3 contains very few surprises, and didn't contain any glaring inaccuracies which would cause me to wince. It also covered all the things which I would expect a new user to be told -- for instance how Red Hat goes about deciding what makes it into a given release, how to find packages, and how to install packages once you have found them. It also covers some basic configuration tasks, such as adding your own run level scripts. If you are a Red Hat power user, then there is nothing new to you here. You should also note that the book does assume that you've already managed to install the system, and don't need to be told how to do things like change directory at a shell prompt.

The book also covers similar details for Slackware 8.0, and Debian 3.0 Beta (the latest version which was available at the time of poublication). Again, if your a power user, then these sections are of limited use, but might serve as a good reference. If your a newer user however, they serve as a very useful introduction to your chosen distribution. Overall, Daniel L. Morril devotes 140 pages to discussions of these three

distributions.

Tuning and Customizing a Linux System then goes on to discuss how to install other software on your Linux system. It discusses the pros and cons of installing source downloaded from the Internet versus installing a package from your distribution vendor. It also discusses how to build source packages of various forms, as well as how to use autoconf and Makefiles. It even goes as far as describing briefly how to roll your own packages for your chosen distribution using downloaded source.

The next hundred or so pages of the book cover how to install several software packages: OpenSSH, Pluggable Authentication Modules, the Dante SOCKS library, Apache, CVS, and Sun's Java Development Kit.

Daniel L. Morril then finishes up Tuning and Customizing a Linux System with three case studies: how to build a desktop Linux machine; how to build a software development machine (although it doesn't match my personal beliefs about what a development machine should look like); and how to build a network firewalling machine. All of these case studies are based on Red Hat 7.3, and are little more than lists of what software you should install if you want to play MPEG movies, surf the Internet, or a variety of other common tasks. As an example, two pages are devoted to instructions on what software to install to be able to play those MPEG movies. There is also discussion of required hardware for the named task -- for example installing network cards in your firewall machine. The reality is that a user is probably better off consulting a HOWTO for this sort of information.

In summary, this is the wrong book to go out and buy if you're expecting to be told how to install a Linux system from scratch. It is also the wrong book to buy if you're a power user looking for high end configuration and optimization tips. On the other hand, if you're new to Linux, and want some hints on how to actually use your new Linux machine once you have managed to install it, then this book is for you. It's also an acceptable reference for what packages to install to perform common tasks.

Online backup using SDS

Author: Joseph Gan <joseph.gan@abs.gov.au>

When backing up in Solaris, the file system must be inactive; otherwise, the output may be inconsistent. A file system is inactive when it is unmounted or it is write locked by OS. Although fssnap utility in Solaris can do online backup, it will fail if the file system could not be write locked.

SDS on the other hand can be used as a "bridge" to transfer the data of a file system across to another partition on the fly. Then the data can be backed up at anytime.

Using a file system "test" as an example:

```
   # df -k /test
   Filesystem                kbytes    used
avail capacity  Mounted on
   /dev/dsk/c0t2d1s3    50700783 39746386 10530839
80%    /test
```

1. Creating a metadevice named d101 on the test file system when it is mounted.

```
   # metainit -f d101 1 1 c0t2d1s3
```

2. Initialising an one-way mirror metadevice named d100 with the submirror d101 created above.

```
   # metainit d100 -m d101
   # metastat d100
   d100: Mirror
       Submirror 0: d101
         State: Okay
       Pass: 1
       Read option: roundrobin (default)
       Write option: parallel (default)
       Size: 102961152 blocks

   d101: Submirror of d100
       State: Okay
       Size: 102961152 blocks
       Stripe 0:
           Device              Start Block  Dbase
State        Hot Spare
           c0t2d1s3                    0     No
Okay
```

3. Creating another metadevice d102 on the new location c0t2d1s7. The size of the new partition should be the same as d101.

```
   # metainit d102 1 1 c0t2d1s7
   d102: Concat/Stripe is setup

   # metastat d102
   d102: Concat/Stripe
       Size: 102975488 blocks
       Stripe 0:
           Device              Start Block  Dbase
           c0t2d1s7                    0     No
```

4. Adding the metadevice d102 as the second submirror to d100, resync would automatically take place.

```
   # metattach d100 d102
```

5. After the resync has completed successfully, you should get the following two way mirrors:

```
   # metastat d100
   d100: Mirror
       Submirror 0: d101
         State: Okay
       Submirror 1: d102
         State: Okay
       Pass: 1
       Read option: roundrobin (default)
       Write option: parallel (default)
       Size: 102961152 blocks

   d101: Submirror of d100
       State: Okay
       Size: 102961152 blocks
       Stripe 0:
           Device              Start Block  Dbase
State        Hot Spare
           c0t2d1s3                    0     No
Okay
```

```
   d102: Submirror of d100
       State: Okay
       Size: 102961152 blocks
       Stripe 0:
            Device              Start Block  Dbase
State        Hot Spare
            c0t2d1s7                    0     No
Okay
```

6. Now you have a file system ready to be backed up at any time.

7. You can schedule to backup the file system whenever you decide. Before you start though, you have to detach the second metadevice d102 or bring it offline:

```
      # metadetach d100 d102  or
      # metaoffline d102
```

8. Remember you can only backup the new partition c0t2d1s7 which includes all the data at the point of detach.

9. After backup has done, you could clean up all the metadevices, or you could leave them for using next time. Before the next backup, all you have to do is

```
   # metattach d100 d102  or
   # metaonline d102
```

Then wait until resyc has done before starting backup. The difference between the two is the first one will take longer to sync than the second one. But the second one was read only until you bring it online.

10. For file systems need to be backed up consistent at the same point, all you have to do is detach all the second metadevics at the same time. Then they can be backed up one by one.

Latest Thinking: Open Source Insight

Author: AUUGN Editor <auugn@auug.org.au>

I recently came across an interesting publication which I thought may be of interest obliquely to AUUGN's traditional technical audience, but possibly of real interest to management and policy staffers within your organisations.

Open Source Insight is published by an outfit called Latest Thinking, coming out of Sydney. It's a weekly newsletter which discusses in full detail many of the issues which relate to the adoption of open source and Unix-related technologies by business and government.

You can obtain a free trial of the newsletter from here: http://www.latestthinking.com/ or by emailing: info@latestthinking.com

AUUG Corporate Members

as at 1st October 2002

- ac3
- Accenture Australia Ltd
- ADFA
- ANSTO
- ANU
- Australian Centre for Remote Sensing
- Australian Bureau of Statistics
- Australian Defence Force Academy
- Australian Taxation Office
- Bradken
- British Aerospace Australia
- Bureau of Meteorology
- C.I.S.R.A.
- Cape Grim B.A.P.S
- Centrelink
- CITEC
- Corinthian Industries (Holdings) Pty Ltd
- Cray Australia
- CSC Australia Pty Ltd
- CSIRO Manufacturing Science and Technology
- Curtin University of Technology
- Cybersource Pty Ltd
- Deakin University
- Department of Land & Water Conservation
- Department of Premier & Cabinet
- Energex
- Everything Linux & Linux Help
- Fulcrum Consulting Group
- IBM
- ING
- Land and Property Information, NSW
- LPINSW
- Macquarie University
- Multibase WebAustralis Pty Ltd
- Namadgi Systems Pty Ltd
- National Australia Bank
- National Library of Australia
- NSW Public Works & Services, Information Services
- Peter Harding & Associates Pty Ltd
- Rinbina Pty Ltd
- Security Mailing Services Pty Ltd
- St John of God Health Care Inc
- St Vincent's Private Hospital
- Sun Microsystems Australia
- The University of Western Australia
- Thiess Pty Ltd
- Tower Technology Pty Ltd
- Uniq Advances Pty Ltd
- UniTAB Limited
- University of Melbourne
- University of New England
- University of New South Wales
- University of New South Wales, Comp. Sci & Eng.
- University of Sydney
- University of Technology, Sydney
- Victoria University of Technology
- Workcover Queensland

DNSTRACER - Exploring the DNS infrastructure

Author: Edwin Groothuis <edwin@mavetju.org>

Quick DNS intro

The Domain Name Server system is a globally replicated and distributed database which primary translate hostnames (www.auug.org.au) into IP addresses (150.101.248.57), route mail (@auug.org.au) to mail-hubs (www.auug.org.au) and converts IP addresses (66.216.68.159) into hostnames (www.auug.org.au). Without it, we would have to use remember the IP addresses of the servers we want to connect to (telnet 131.155.132.36 4000) and it would be very hard to send emails as easy as it goes today (mcvax!moskvax!kremvax!chernenko).

Normally you don't have to worry about DNS, you just get the settings for the nameserver you have to use via PPP when dialing into an ISP or via DHCP when connecting to a LAN at a company. They make sure that their nameservers know where to get the rest of their data, which are initially the root-nameservers.

The root-nameservers are the 13 (13 logical, but physical more) most important nameservers on the Internet. They know where the rest of the DNS servers can be found.

Furthermore you have master and slave servers for a domain: the data for a domain is only manually changed at the master, the slaves transfer the data via the internal DNS mechanics.

Quick DNS example

If you're requesting the IP address of www.auug.org.au your nameserver will ask one of the root-servers for it. It will reply that it doesn't know it, but that the answer can be found at the DNS servers for .au and supplies a list with them and their IP addresses (The list is known as Authority Data, the IP addresses are known as Additional Data). Your server will ask the question again at one of the servers responsible for .au and get a similar answer: it doesn't know it, but it hands you a list of servers for .org.au and their IP addresses. This goes on until you're at the servers which are responsible for auug.org.au, in which case you get the IP address of www.auug.org.au (Answer Data).

Your server now caches the data for .au, .org.au, .auug.org.au and www.auug.org.au for a short time (the Time To Live) so that following requests for that data doesn't need to explore so much, it just can do a quick lookup of in it's own cache and returns the answer.

SPOF?

The DNS system is not really a SPOF, it is designed as a globally replicated and distributed database which means that if you can't reach one of the servers, you can try it at a different one. As there are 13 root-servers which know where to find the rest, there are 6 servers for the .au domain (6 logical with a total of at least 8 IP addresses), there are 9 servers for the .org.au domain and four servers for the auug.org.au domain. The location of the servers on the Internet and replication is used to overcome connectivity problems. Regarding the network, there isn't much which can go wrong. Regarding the administrative side of it, that's where things go wrong.

APOF!

When you register a new domain, you are asked what the nameservers are and if necessary also the IP addresses. Furthermore, these nameservers have to be configured to answer requests for that new domain and to exchange information between them. And actually data has to be served on that domain. Five places for things to go wrong!

Unknown nameservers

At the time of writing, one of the domains of a nameserver for .org.au has expired (for people interested: optus.net has expired at December 16th 2002 and after half a month it still hasn't been re-registered). That means that the IP address of the nameserver audns01.syd.optus.net can't be found and that this server will never be queried (after all, if you don't know an IP address you can't connect to it)

Wrong IP addresses

Changing the IP address of a nameserver is a pain and often it will be forgotten on one or two machines (Remember that switch in the cupboard which got installed a long time ago? Yes, that one too has the IP address of the DNS server hardcoded). Or that the registrar makes it impossible to change the IP address of the nameserver via their website because of all kind of internal checks.

Lame and stealth servers

Lame servers are servers which are mentioned in the NS records for a domain but are not authoritative for that domain. This can happen because of a typo in the IP address or a change which has never been fully finished (new server added while it wasn't ready or old server data removed but never from the NS records).

Stealth servers are servers which are not mentioned in the NS records but are authoritative for that domains. For example servers which have been removed from the NS records but the configuration of the server never updated.

Old data on a server

When data is changed on the master server, the slaves will have to transfer it from there. But sometimes they can't because the master has disabled it for some reason. In that case the data on the slaves will get more and more obsolete.

Wrong data on a server

DNS server software has strange habits and one of them is often that if you end a name without a dot, it will add the current domainname to it. So if you see a zonefile with www.auug.org.au.auug.org.au, you know that they forgot to end it with a dot at the end.

Now what is dnstracer?

Remember the traceroute(8) utility? It shows the path an IP packet takes when you send it to its destination IP address. Remember ntptrace(8)? It shows the path of NTP servers which your NTP client is syncing on. Dnstracer is something similar, it shows you where a DNS server will go for its information. So if you want to know the path to www.auug.org.au:

```
[~] edwin@k7>dnstracer -s . -o www.auug.org.au
Tracing to www.auug.org.au via A.ROOT-SERVERS.NET, timeout 15
seconds
A.ROOT-SERVERS.NET [.] (198.41.0.4)
 |\___ SEC3.APNIC.NET [au] (202.12.28.140)
 |     |\___ audns01.syd.optus.net [org.au] (No IP address)
 |     |\___ au21d.csiro.au [org.au] (130.116.2.21)
 |     |     |\___ ns.aone.net.au [auug.org.au] (203.2.192.95) Got
authoritative
+answer
 |     |     |\___ supreme.canb.auug.org.au [auug.org.au]
(203.10.76.34) Got
+authoritative answer
 |     |     |\___ dns0.optus.net.au [auug.org.au] (202.139.83.3)
Got
+authoritative answer
 |     |      \___ ns.auug.org.au [auug.org.au] (150.101.248.57)
Got
+authoritative answer
 |     |\___ dns1.telstra.net [org.au] (203.50.5.200)
 |     |     |\___ ns.auug.org.au [auug.org.au] (150.101.248.57)
(cached)
 |     |     |\___ ns.aone.net.au [auug.org.au] (203.2.192.95)
(cached)
 |     |     |\___ supreme.canb.auug.org.au [auug.org.au]
(203.10.76.34)
+(cached)
 |     |      \___ dns0.optus.net.au [auug.org.au] (202.139.83.3)
(cached)
 |     |\___ box2.aunic.net [org.au] (203.202.150.20)
 |     |     |\___ supreme.canb.auug.org.au [auug.org.au]
(203.10.76.34)
+(cached)
 |     |     |\___ dns0.optus.net.au [auug.org.au] (202.139.83.3)
(cached)
 |     |     |\___ ns.auug.org.au [auug.org.au] (150.101.248.57)
(cached)
 |     |      \___ ns.aone.net.au [auug.org.au] (203.2.192.95)
(cached)
 |     |\___ ns4.ausregistry.net [org.au] (210.8.15.253)
 |     |     |\___ dns0.optus.net.au [auug.org.au] (202.139.83.3)
(cached)
 |     |     |\___ ns.auug.org.au [auug.org.au] (150.101.248.57)
(cached)
 |     |     |\___ ns.aone.net.au [auug.org.au] (203.2.192.95)
(cached)
 |     |      \___ supreme.canb.auug.org.au [auug.org.au]
(203.10.76.34)
+(cached)
 |     |\___ ns3.melbourneit.com [org.au] (203.27.227.10) *
 |     |     |\___ ns.aone.net.au [auug.org.au] (203.2.192.95)
(cached)
 |     |     |\___ supreme.canb.auug.org.au [auug.org.au]
(203.10.76.34)
+(cached)
 |     |     |\___ dns0.optus.net.au [auug.org.au] (202.139.83.3)
(cached)
 |     |      \___ ns.auug.org.au [auug.org.au] (150.101.248.57)
(cached)
 |     |\___ ns3.ausregistry.net [org.au] (203.18.56.43)
 |     |     |\___ dns0.optus.net.au [auug.org.au] (202.139.83.3)
(cached)
 |     |     |\___ ns.auug.org.au [auug.org.au] (150.101.248.57)
(cached)
 |     |     |\___ ns.aone.net.au [auug.org.au] (203.2.192.95)
(cached)
 |     |      \___ supreme.canb.auug.org.au [auug.org.au]
(203.10.76.34)
+(cached)
 |     |\___ ns2.ausregistry.net [org.au] (203.18.56.42)
 |     |     |\___ supreme.canb.auug.org.au [auug.org.au]
(203.10.76.34)
+(cached)
 |     |     |\___ dns0.optus.net.au [auug.org.au] (202.139.83.3)
(cached)
 |     |     |\___ ns.auug.org.au [auug.org.au] (150.101.248.57)
(cached)
```

```
|     |       \___ ns.aone.net.au [auug.org.au] (203.2.192.95)
(cached)
|       \___ ns1.ausregistry.net [org.au] (203.18.56.41)
|           |\___ ns.auug.org.au [auug.org.au] (150.101.248.57)
(cached)
|           |\___ ns.aone.net.au [auug.org.au] (203.2.192.95)
(cached)
|           |\___ supreme.canb.auug.org.au [auug.org.au]
(203.10.76.34)
+(cached)
|           \___ dns0.optus.net.au [auug.org.au] (202.139.83.3)
(cached)
 |\___ SEC1.APNIC.NET [au] (202.12.29.59)
 |     |\___ au21d.csiro.au [org.au] (130.116.2.21) (cached)
 |     |\___ dns1.telstra.net [org.au] (203.50.5.200) (cached)
 |     |\___ box2.aunic.net [org.au] (203.202.150.20) (cached)
[...]
supreme.canb.auug.org.au (203.10.76.34) www.auug.org.au ->
150.101.248.57
ns.aone.net.au (203.2.192.95)           www.auug.org.au ->
150.101.248.57
ns.auug.org.au (150.101.248.57)         www.auug.org.au ->
150.101.248.57
dns0.optus.net.au (202.139.83.3)        www.auug.org.au ->
150.101.248.57
```

Just like expected: the server goes to a root-server, the servers for the .au domain, the servers for the .org.au domain and the servers of the .auug.org.au domains. The answers received are printed at the end and they all seem to agree on it.

Sometimes it will go wrong, for example when a server is unreachable or when a lame server is detected:

```
[~] edwin@k7>dnstracer -o -s RELAY-1.FTEL.CO.UK
fataldimensions.nl.eu.org
Tracing to fataldimensions.nl.eu.org via RELAY-1.FTEL.CO.UK,
timeout 15 seconds
RELAY-1.FTEL.CO.UK (192.65.220.24)
 |\___ ns.cistron.nl [nl.eu.org] (62.216.31.55) Got answer
 |\___ ns.lf.net [nl.eu.org] (212.9.160.1) Got answer
 |\___ ns.eu.org [nl.eu.org] (137.194.2.218) Lame server
 |\___ ns2.ispi.net [nl.eu.org] (206.131.193.15) Got authoritative
answer
 |\___ ns.patriots.net [nl.eu.org] (206.131.200.40) * * *
  \___ auth1.dns.elm.net [nl.eu.org] (81.17.34.251) Got
authoritative answer
[...]
```

The difference between "Got answer" and "Got authoritative answer" is that the first one can be a cached answer, while the second one is one from a server which admits that its responsible for that domain.

What can you see with dnstracer?

Dnstracer shows you the path from the root DNS servers to the DNS servers responsible for a domain. It shows if there are unreachable servers, lame servers but doesn't show servers which aren't configured for that domain. It will query for MX records, SOA records, NS records, normal A/AAAA and PTR records (and other ones). And at the end, it will print the results received. But it will not interpret the results for you.

More information

See http://www.mavetju.org/unix/dnstracer.php for more information about the dnstracer utility and how to obtain it. For FreeBSD and OpenBSD, it is in the ports-collection. For Linux, there is an RPM for it. Otherwise, just grab the tarball and compile it.

'Busy Tone' for CGI Web Applications

Author: Steve Jenkin <sjenkin@canb.auug.org.au>

This was implemented for the Australian Government's first large on-line transaction - Business Number (ABN) registrations. The site served 20 times its normal load for two days and maintained reasonable service levels for connected users.

Techniques for handling caching or overload of static pages are well understood. This is a method for handling CGI based applications.

Busy Tone

The telepone system answers the question, "How do you economically handle extraordinarily high traffic loads, such as Christmas day"? With a 'busy tone' – "come back later, we're busy just now".

These traffic peaks are characterised as unpredictable, unrestrained demand of short duration, with no tolerance for increased pricing. It is uneconomic for the service provider to install and maintain sufficient permanent capacity for a 10- or 20-fold increase over normal loads. Callers/users will trade overall cost for busy time limitations, preferably with predictable delays.

This concept is also called "Load Shedding".
The prerequisites are:-

- Extreme peaks in demand that can't be serviced economically,
- user tolerance of limited busy time access,
- a usability requirement of reasonable service levels once connected,
- precise and repeatable identification of individual sessions, allowing user anonymity,
- real-time measurement of the primary service level characteristic, and
- the ability for a real-time load shedding response to system saturation.

On-line ABN Registration System

The on-line ABN registration system (https://trans.business.gov.au) was deployed in November 1999. It was expected to serve 5% of a total 2M registrations. Businesses wishing to claim GST (Goods & Services Tax) rebates after 1st July, 2000, had to apply for an ABN 2-4 weeks in advance to allow for Tax Office checks and processing. The advertised deadline, our "Christmas Day", was 31 May 2000. Figures from Canada suggested 60% of registrations would occur in the last 3 weeks.

The application consisted of ~20 HTML forms backed by a CGI script interfacing to an ORACLE database. Completed applications were transferred by verified e-mail to the Tax Office system. Security regulations required the database system be isolated from the web servers by a firewall. A 'layer 4' (L4) load balancer was used to share the load across multiple

web servers, and to provide increased manageability and availability. All connections were SSL (https) hence sessions could be identified by the L4 switch and web server without using cookies - a key part of the privacy policy.

The CGI application could not see this information and relied on HTTP's username/password facility to identify sessions and allow later 'resumption' of sessions.

There was a single 'entry' page where registrants were allocated a unique session identifier and requested for a password. This page was used as the 'gate', either allowing users access to the registration forms or delivering a 'busy, please try later' page.

The site accepted 350,000 ABN applications of 3.3M total by 1 June 200. Near 50% of sessions lead to submitted applications. A factor in this was an application time limit of 36 hours to 'resume' a held application.

Registrations grew from a few hundred a day at launch, to 21,000 on 31st May, 2000. 'Steady state' traffic became ~650 a day, 4000 a week - probably half current new registrations. Traffic doubled about every 3 weeks, excepting the six week school holiday period over Christmas.

There were consistent daily and weekly traffic patterns - two roughly equal peaks during the day, AM and PM, and an evening peak up to 75% of the days'. Traffic tailed off quickly after midnight East Coast time, resuming about 6 AM, and getting busy about 9 AM. The evening peak was consistently missing on Friday nights, whilst it took until midday Saturday for the traffic to rise. Friday is still 'party night' in Australia. The Sunday night peak lasted later than any other day - till past midnight. Weekend traffic was about 60% normal. Mondays were always 'slow days', whilst the busiest day moved from Tuesday to Thursday.

The web and database servers were all multi-CPU systems capable of being quickly upgraded. There was sufficient system memory to avoid any paging, except during database backups. The database server used a hardware disk array giving about a 10 fold performance increase on most disk operations over the other systems. There was a six week lead time in purchasing additional CPU's. For the last of the 'crunch' time, the hardware supplier generously loaned some disks and a CPU board, giving 2 and 4 CPU's on the two web servers and 6 CPU's on the database. Commissioning additional web servers was investigated, but abandoned due to the relatively low performance of available hardware and the high cost and delay in building and configuring 'hardened' systems.

ORACLE was hired to perform a comprehensive performance analysis of the database and to tune it. Another contractor was employed to analyse the network traffic, identify bottlenecks, and report daily on system throughput, response to load, and estimate system throughput limits. A number of significant application improvements were made as a result.

Some simple light-weight system monitoring, via pages of graphs available from the web server, was set-up with the public domain software, RRD (Round Robin Database). This became the most powerful and useful tool for monitoring system health and performance. It was crucial in explaining performance to mangers and developers alike. It was also crucial in tuning and monitoring the 'busy tone' software when deployed.
A key system overhead, CPU time used by the web server processing the encrypted SSL connections was not available. Replaying and timing live traffic captured after decryption to establish a comparison was not possible at the time

ABN "Busy Tone" Implementation

At the end of February, well before the final deadline of 31st May, the system hit saturation. At the time there was no L4 switch, a single 2 CPU web server, and 2 CPU's in the database server. The peak system load had been 70% the previous week, allowing about 2 weeks before saturation. A number of system activities were scheduled to meet this projected demand in time.

The Tax Office launched its advertising campaign for the site over the week end, and traffic roughly doubled on the Monday. As queuing theory models, when a resource comes near saturation, service times explode. Response times that had sat around 2 seconds for months went out beyond 20 seconds, and a flood of complaints came down the hotline. All users were unable to get response from the system with the TCP timeouts. Although the systems were flat out, no useful work was done.
This provided the impetus to commission the 'busy tone' subsystem and for management approve roughly doubling CPU capacity. The overload was addressed in the short term by bringing forward a scheduled upgrade to the script interpreter used.

The 'gate' page CGI acted in concert with a monitoring program that set a busy flag in the database. Additionally, to cater for failure of the monitor program, the 'gate' page CGI also counted, using the database, the number of applications started. When this exceeded a tuneable threshold per period, the busy page was served.

Because there was a single database and multiple web servers, the monitor program was only run on a single web server, that judged to be the busiest. The L4 switch was used to apportion sessions according to the CPU capacity of the frontends.

The performance target was a per page 'response time' of 5 seconds. Only the page start time was available from the web server logs, not the elapsed time per page. Also, there was no way to separate the server response time (internal) from the network (external) delays. Web browser delays (rendering pages) were also ignored, being completely client dependent and unmeasurable.

The response time monitor program read the system accounting logs each period (settable) and calculated a mean response time. During overnight database backups, apparent response times blew out, so a minimum traffic threshold was added to the monitor. A huge assumption was made: that the execution time of the CGI program reflected the server part of the total response time.

An initial design idea was to model behaviour on 'sendmail' and set the busy flag based on 'system load average', a number that accurately scales with multiple CPU's and systems, is very low-cost to obtain, and does not need a background monitor. This was abandoned when it was realised that it is a secondary measure, not primary, and behaves exceedingly non linearly. By the time the load average started to rise, the response times would've exceeded the target. Data for load average and nominal response times still exist and this hypothesis could be tested.

The L4 switch also tested the 'health' of each of the web servers by polling a page. If a server failed to respond before the next health check, the server was automatically taken out of service. The first time the database system saturated, all the web servers were 'failed' as response times uniformly blew out. For the rest of the duration, the L4 switch 'health checks' were set to 'TCP/IP' (ping).

The busy tone subsystem was deployed 3 weeks before the deadline. There was still more than 30% of the system unused at the time. Apart from testing, it 'fired' for the first time about 10 days before the deadline. It was active for most of 2nd last day, and from about 9 AM until after midnight on the final day. Due to other commitments on test system resources and privacy problems with recording and replaying live traffic, none of the design and implementation assumptions had been tried before deployment, nor were system response characteristics known.

Tuning the busy tone parameters, and understanding the interactions between load, demand, and system behaviour, proved somewhat 'hit and miss'. A final set of tuning parameters were settled on that kept response time 'cycling' between 3 and 5 seconds, and the rate of new sessions reasonably constant, maximising system throughput whilst maintaining user response times.

The final model was a feedback system with high inertia - due to the 'gate' committing the system to 20 or more pages of work over ~30 minutes for each accepted connection. The monitor cycle time was initially too high, it provided a 'moving average' of 5 to 10 minutes, not close to an instantaneous value. The system showed classic feedback systems 'hunting' behaviour - cycling between 2 and 15 seconds response time. The 'undershoot' significantly lowered system throughput, causing the subsystem to be switched off at times. ('Hunting' is a control system failure - usually heard in stationary motors where the speed surges up and down.)

The resources used in running the 'gate' CGI script was not monitored, resulting in only about 75% best throughput of registrations on the final night. The rest of the system resources presumably were devoted to serving 'busy pages'.

Unexpectedly, the firewalls became the ultimate limiting factor. All systems only went to about 80-90% CPU at the busiest time, whilst a number of subsystems, such as DNS and e-mail experienced network connectivity problems.

Other strategies were employed to maximise throughput over the last two days. The supporting website and searches were turned off and replaced by a single static page, e-mailing submitted applications was rescheduled to the overnight quiet time, and replicating the database logs, 500 Mb/hour, using the CPU intensive 'secure shell' was stopped. A number of small websites containing mostly static pages were moved to an unused small system well beforehand.

Further Work

Whilst the subsystem developed performed adequately, it only had to survive a single overwhelming peak. Intensive management and manual intervention and some extreme trade-offs, such as stopping rollback log replication exactly when it was needed most, were acceptable for the last two days.

For large scale commercial systems that can reasonably expect ongoing unpredictable overwhelming loads, moving a number of facilities off the web servers to dedicated switches or servers optimised for their tasks would be most useful:-

- SSL servers are now available to lighten server CPU load considerably.
- Definitive response timings, even per server, are available at the L4 switch.

The L4 switch, if it buffers server responses, can simultaneously calculate both the server and network delay, and minimise use of kernel resources on servers by quickly releasing connections.

The L4 switch is the perfect place to make the busy tone decision and return a static busy page with a short cache life.

L4 switches already recognise sessions, allowing busy tone to be extended from the 'gate' configuration described here, to sets of individual CGI pages.

Modifying individual CGI scripts for busy tone requires considerable redesign and rework. The L4 switch is already a separate, high reliability device that could provide a consistent, easily implemented and tuned implementation.

This article is re-printed with permission. The originals can be found at:

URL://www.canb.org.au/~sjenkin/busytone.swx

Gridbus: A toolkit for service-oriented grid computing

Authors: Rajkumar Buyya <raj@cs.mu.oz.au> and Srikumar Venugopal, The University of Melbourne

ABSTRACT

Computational Grids enable the sharing, selection, and aggregation of geographically distributed resources (such as computers, data bases, scientific instruments) for solving large-scale problems in science, engineering, and commerce. However, application development, resource management, scheduling, and supporting end-to-end quality-of-services (QoS) in these environments is a complex undertaking. This is due to the geographic distribution of resources that are owned by different organizations having different usage policies and cost models, and varying loads and availability patterns. This article describes Gridbus, an open-source framework that addresses those challenges in support of a distributed computational economy.

The Gridbus project (http://www.gridbus.org) is engaged in the design and development of open source cluster and grid middleware technologies for service-oriented computing. Gridbus emphasizes the end-to-end quality of services driven by computational economy at various levels - clusters, peer-to-peer (P2P) networks, and the Grid - for the management of computational, data, and application services.

At the cluster level, the Libra scheduler has been developed to support economy-driven cluster resource management. Libra is used within a single administrative domain for distributing computational tasks among resources that belong to a cluster. At the P2P network level, the CPM (compute-power- market) infrastructure is being developed through the JXTA community. At the Grid level, various tools are being developed to support a quality-of-service (QoS) - based management of resources and scheduling of applications. To enable performance evaluation, a Grid simulation toolkit called GridSim has been developed. GridSim supports the modeling and simulation of application scheduling on simulated Grid resources. Finally, to support the accounting of resource or service usage and enable sustainable resource sharing across virtual organizations, we have developed Grid Accounting Services infrastructure.

GRIDBUS TECHNOLOGIES

Gridbus technologies and their utilization in deploying real-world applications, such as brain activity analysis, on Global Grids has been demonstrated at the recent IEEE/ACM Supercomputing (SC 2002) conference held in Baltimore, MD, USA. A high-level interaction between various Gridbus components is shown in Figure 1.

We briefly discuss some of the key technologies developed as part of the Gridbus project.

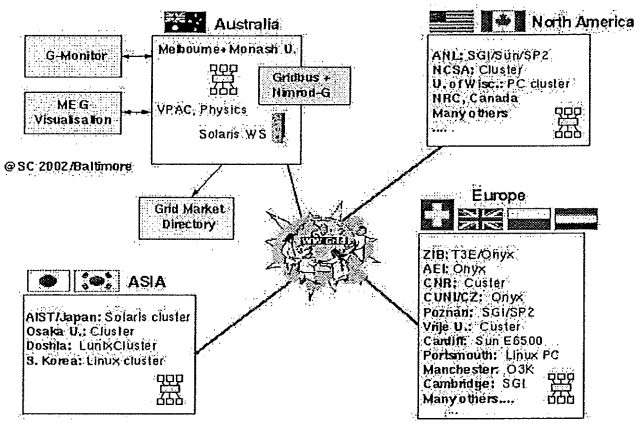


Figure 1: Gridbus at SC2002:

A high-level view of system interaction in a grid-based computation Visual Parameter Sweep Application Composer The Gridbus project developed a Java based IDE, called Visual Parametric Tool (VPT), for rapid creation of parameter sweep (data parallel/SPMD) applications. VPT allows users to parameterize the input data files to transform static values to variable parameters, and to create a script that defines parameters and tasks. VPT also allows the rapid creation and manipulation of the parameters. While being flexible, it is also simple enough for a non-expert to create a parameter script, known as a plan file, within minutes. The parameter sweep applications composed using VPT can be deployed on global Grids using the Nimrod-G resource broker (see Resources). Nimrod-G supports scheduling based on the user's quality of service (QoS) requirements, such as computational deadline, budget, and optimization preference, and the access price of resources.

GRID MARKET DIRECTORY (GMD)

The Grid Market Directory (GMD) (see Resources) serves as a registry for high-level service publication and discovery in virtual organizations. It enables service providers to publish the services which they provide along with the costs associated with those services. Consumers browse GDM to find services that meet their requirements.

GMD is built over standard Web service technologies such as SOAP (Simple Object Access Protocol) and XML. Therefore, it can be queried by other programs. To provide with an additional layer of transparency, a client API (Application Programming Interface) has been provided that could be used by programs to query the GMD without the developers having to concern themselves with SOAP details. The Gridbus scheduler interacts with the GMD to discover the testbed resources and their high-level attributes such as access price.

GRID SCHEDULER

The Gridbus scheduler, developed as a plugin scheduler for Nimrod-G, has been used instead of the default Nimrod-G scheduler, as it has been designed to utilize the GMD. To support the notion of application services and pricing based on AO (Application Operation) instead of vanilla CPU service, the GMD already allows GSPs (Grid Service Providers) to publish application services along with their AO service price. Hence, the Gridbus scheduler can utilize the GMD services and perform resource allocation based on AO cost model. In this model, the user is charged a price for executing each job on the resource. Thus, the resource owner may offer the application as a service and charge a fixed price for executing it.

The Gridbus Scheduler implements three algorithms:
1. Cost minimization
2. Time minimization, and
3. Cost-time optimization

All three algorithms are constrained by two parameters: the deadline by which the job is required to complete, and the user's budget. Time minimization tries to execute the project within the shortest time while keeping within the budget. Cost minimization tries to complete the execution with the least cost while keeping to the deadline. Cost-time optimization gives jobs to the cheapest servers but performs time optimization among those.

G-MONITOR

G-Monitor is a web portal for monitoring and steering application execution on global grids. It interacts with Grid Resource Broker (GRB), Nimrod-G in the current implementation, to provide the user with a GUI (graphical user interface) to the underlying Grid framework.

G-Monitor provides a ubiquitous interface that is easy to use, enabling the end-user to monitor and control jobs running within the Grid environment. It is flexible enough to be run from anywhere without the need for custom client software or network overhead. G-Monitor is also scalable, and can therefore handle thousands of nodes and jobs running in a Grid environment.

GridBank

GridBank (GB) is a secure Grid-wide accounting and (micro) payment handling system. It maintains the users' (consumers and providers) accounts and resource usage records in a database. GridBank supports protocols that enable its interaction with the resource brokers of Grid Service Consumers (GSCs) and the resource traders of Grid Service Providers (GSPs). It has been primarily designed to provide services for enabling a Grid computing economy; however, we envision its usage in e-commerce applications as well. The GridBank services can be used in both co-operative and competitive distributed computing environments.

GRIDSIM

The GridSim toolkit supports modeling and simulation of a wide range of heterogeneous resources: Single- or multiprocessors, shared and distributed memory machines, such as PCs, workstations, SMPs, and clusters with different capabilities and configurations.

GridSim can be used for modeling and simulation of application scheduling on various classes of parallel and distributed computing systems, such as clusters, grids, and P2P networks. The GridSim toolkit provides facilities for the modeling and simulation of resources and network connectivity with different capabilities, configurations, and domains. It supports primitives for application composition, information services for resource discovery, and interfaces for assigning application tasks to resources and managing their execution. These features can be used to simulate resource brokers or Grid schedulers for evaluating performance of scheduling algorithms or heuristics.

The GridSim Toolkit has been used to create a resource broker that simulates Nimrod-G for the design and evaluation of deadline and budget constrained scheduling algorithms with cost and time optimizations. It is also used to simulate a market-based cluster scheduling system in a cooperative economy environment.

RECENT GRIDBUS RELEASE

In November, 2002, the Gridbus project has released GridSim 2.0 that provides support for the simulation of time or space shared, single and multiprocessor systems (both shared and distributed memory systems), as well as a new package for creating Grid model visually.

ABOUT THE AUTHORS

Rajkumar Buyya is founding director of the Grid Computing and Distributed Systems (GRIDS) Laboratory at the Department of Computer Science and Software Engineering at the University of Melbourne. He is also co-chair of the IEEE Computer Society's Task Force on Cluster Computing. He edited the two- volume High Performance Cluster Computing: Architectures and Systems (Prentice Hall, 1999), as well as High Performance Mass Storage and Parallel I/O: Technologies and Applications (With Tony Cortes and Hai Jin. John Wiley and Sons, 2001.)

Srikumar Venugopal is a doctoral candidate at the Grid Computing and Distributed Systems (GRIDS) Laboratory at the Department of Computer Science and Software Engineering at the University of Melbourne.

RESOURCES

The Gridbus project home page
http://www.gridbus.org/

Rajkumar Buyya. Economic-based Distributed Resource Management and Scheduling for Grid Computing. (Doctoral Dissertation, Monash University, Australia, 2002)
http://www.cs.mu.oz.au/~raj/thesis/

R. Buyya. D. Abramson. J. Giddy. Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid. In Proceedings of the 4th International Conference on High-Performance Computing in the Asia-Pacific Region (HPC Asia 2000). 2000
http://www.cs.mu.oz.au/~raj/papers/nimrodg.pdf

The JXTA Community Web site
http://www.jxta.org/

Rajkumar Buyya's home page
http://www.buyya.com/

(X)dialog: Talking shells

Katja and Guido Socher *<katja(at)linuxfocus.org, guido(at)linuxfocus.org>*

Abstract

Xdialog and dialog are two classic utilities to enhance your shell scripts with a graphical user interface.
You will need some shell programming knowledge to understand the article. To learn about the basics of shell programming you can read our article on Shell Programming.

Introduction

The UNIX shell is a very productive environment in itself and works well without a graphical user interface. In a few cases however it makes sense to have a graphical dialog with the user. An example would be the installation dialog of a program. You have a number of options for the features to install and you can select the target directory ....

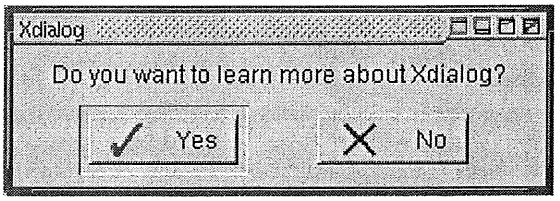
With dialog and Xdialog you can design a graphical application by writing just a short shell script. Dialog is a purely terminal based program and Xdialog is a X11 program

Here is an example:

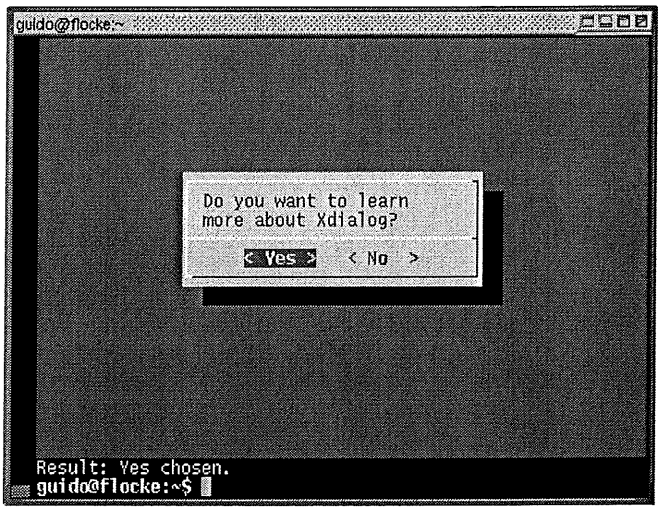
You can type (or copy/paste) the following lines into a shell window (xterm, konsole,....):

```
bash
Xdialog --yesno "Do you want to learn
more about Xdialog?" 0 0;\
case $? in
0)
echo "Result: Yes chosen.";;
1)
echo "Result: No chosen.";;
255)
echo "ESC pressed.";;
esac
```

The box that appears will look like this:



If you use dialog instead of Xdialog (remove the X on the second line in the script shown above) then you get a curses based application that runs within the xterm and does not open a separate window. In some cases this is more appropriate for a shell script as it runs just within the terminal window. Especially if you want to run it remotely with a number of different hosts between your computer and the remote host where direct IP routing is not available. In this case dialog will work but you will not be able to start an X11 application like Xdialog.



The above was a pretty useless example of dialog/Xdialog but it shows how easy it is to program a simple graphical dialog. There are more interesting dialog boxes. Boxes like calendar, menus, filemanager, progess bar, text input, message box, password dialog, ... You can run

```
dialog -help
```

or

```
Xdialog --help
```

to get a list of the available dialog boxes. Xdialog has a few more boxes than dialog.

How it works

The dialog boxes are configured on the command line.

```
dialog --yesno "text string" <height>
```

```
<width>
```

After typing dialog or Xdialog you have to give the name of the box you want followed by its specific parameters.

The yesno box takes 3 parameters. The <height> and <width> can be set to zero in which case the geometry of the box will be automatically adjusted to the size of the text. The result is returned as exit status to the script in the "$?" variable. If there is more to return like e.g. the names of selected options then this is returned on standard error. Standard error is normally just printed to the screen but can be redirected with "2>".

A very simple but efficient solution.

Real applications

Now a real world application where Xdialog/dialog really provides an advantage over conventional shell script programs: A menu where you can select different Internet Service Providers and dial out to connect to the Internet. This script requires the ppp-on/ppp-off scripts from the March 2001 article Using different ISPs for your Internet access. The script is called pppdialout and displays a different menu dependent on whether you are online or not.

```
#!/bin/sh
#
#DIALOG=Xdialog
DIALOG=dialog
#
# name of your default isp:
defaultisp=maxnet
#
error()
{
    echo "$1"
    exit 2
}
help()
{
  cat <<HELP
pppdialout -- select an ISP and dial
out.
All available ISPs must have a config
file in /etc/ppp/peers

pppdialout executes the ppp-on/ppp-off
scripts as described
in
http://linuxfocus.org/English/March2001/
article192.shtml

pppdialout, copyright gpl,
http://linuxfocus.org/English/November20
02
HELP
  exit 0
}

# parse command line:
while [ -n "$1" ]; do
case $1 in
    -h) help;shift 1;; # function help
is called
    --) shift;break;; # end of options
    -*) echo "error: no such option $1.
-h for help";exit 1;;
    *)  break;;
esac
done

tempfile=/tmp/pppdialout.$$
trap "rm -f $tempfile" 1 2 5 15

# check if we have a ppp network
interface
if /sbin/ifconfig | grep '^ppp' >
/dev/null; then
    # we are already online
    $DIALOG --title "go offline" --yesno
"Click YES to \
                    terminate the ppp
connection" 0 0
    rval="$?"
    clear
    if [ "$rval" = "0" ]; then
        echo "running
/etc/ppp/scripts/ppp-off ..."
        /etc/ppp/scripts/ppp-off
    fi
else
    # no ppp connection found, go online
    # get the names of all available ISP
by listing /etc/ppp/peers
    for f in `ls /etc/ppp/peers`; do
        if [ -f "/etc/ppp/peers/$f" ];
then
            isplist="$isplist $f ==""
        fi
    done
    [ -z "$isplist" ]&&error "No isp def
found in /etc/ppp/peers"
    #
    $DIALOG --default-item "$defaultisp"
--title "pppdialout"  \
        --menu "Please select one of\
    the following ISPs for dialout" 0 0
0 $isplist 2> $tempfile
    rval="$?" # return status, isp name
will be in $tempfile
    clear
    if [ "$rval" = "0" ]; then
        isp=`cat $tempfile`
        echo "running
/etc/ppp/scripts/ppp-on $isp..."
        /etc/ppp/scripts/ppp-on "$isp"
    else
        echo "Cancel..."
    fi
    rm -f $tempfile
fi
# end of pppdialout
```

How the script works:

At the beginning we define some functions, error and help, next we check for commandline arguments then a name for a temporary file is defined (/tmp/pppdialout.$$). $$ is the name of the current process and is a unique number for every computer.

The trap statement is executed if the program terminates abnormally (like the user presses crtl-C) and deletes the tempfile in our case. After that we check if we are already online or not (command: /sbin/ifconfig | grep '^ppp'). If we are already online then we open a yesno-box, the one you have already seen above, and ask the user if she wants to go offline. If we are not online then a menu box is opened. We get all the available ISPs by listing the files in /etc/ppp/peers (ls /etc/ppp/peers). The syntax of the menu box is:

```
dialog --menu "text" <height> <width>
<menu height> <tag1> <description> ...
```

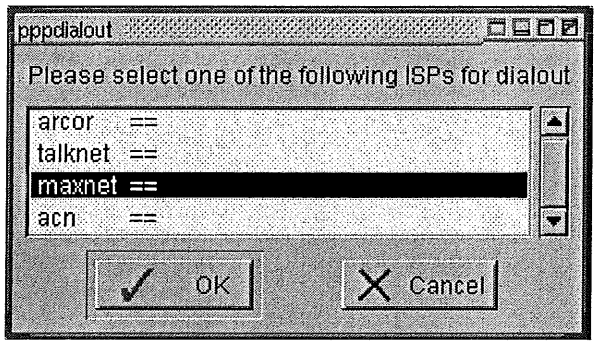
The <height>, <width> and <menu height> are again set to zero (automatic size, see above) and then the program expects pairs of strings (<tag1> <description>). We don't really have a description so we set it to something meaningless (== in this case). The data in the variable isplist will look like this:

```
isp1 == isp2 == isp3 ==
```

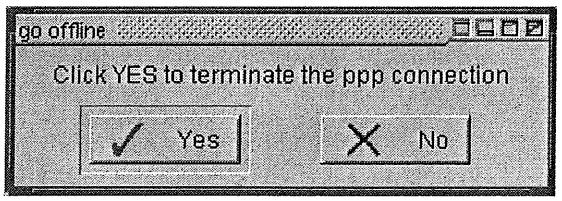
The result of the user's choice is printed by (X)dialog on standard error. The shell command "2> $tmpfile" writes it to our tmpfile. The menu box offers also the possibility to press cancel. Therefore we have to check $? (exit status) to find out which button is pressed.

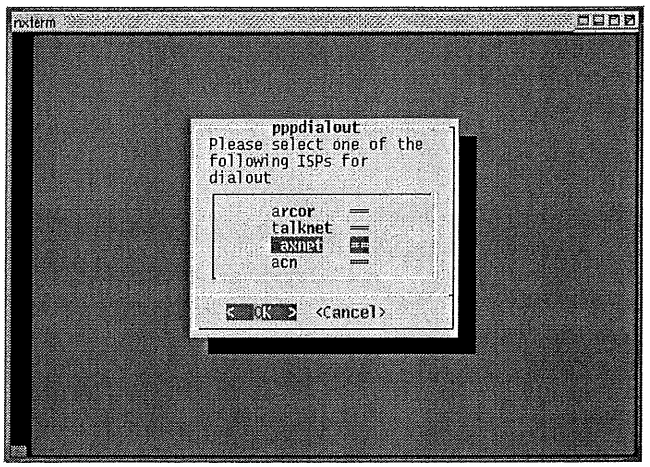
Ok, enough theory. Here is how it looks like

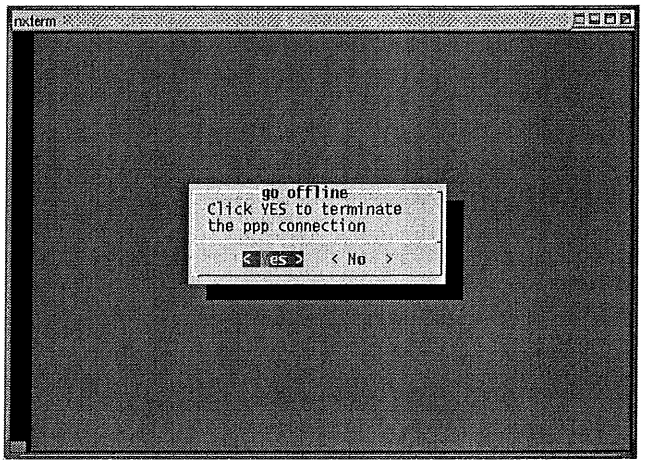
... as nice GTK GUI with Xdialog:



.. with the curses based dialog in the terminal:







More applications

We have one more application for you. It is called mktgz and uses the checklist box of Xdialog. The plain terminal based dialog does not have a checklist therefore it works only with Xdialog. You can use mktgz to build tar.gz packages.

```
mktgz yourpackage .
```

This displays all files of the current working directory (".") and you can select which ones to include into the yourpackage.tar.gz package. You can download it here (mktgz.txt) We will not go through the code line by line as you should already know enough to read the script.

Xdialog and dialog come with a directoy called "samples" where you can find more examples (Redhat 7.3 stores them under /usr/share/doc/Xdialog-2.0.5/samples). Be careful however as some of them really do something and are not pure demo applications.

Conclusion

Xdialog and dialog offer a number of different dialog boxes. Not all of them are always appropriate for all

types of shell scripts. The shell itself is also a very "powerful" environment. Completing a path with the tab-key can be a lot faster than searching the different directories in a GUI and clicking on them. Especially the possibility to pipeline and combine commands makes it a very powerful tool. Something like:

```
grep -i "somestring" file.txt | sort |
uniq | wc -l
```

(for those not so experienced with UNIX shells: This counts all distinct lines in file.txt which contain the string "somestring")
Such pipeline constructs are possible because all the commands are controlled by command line arguments. In other words: they don't stop and ask the user how she wants to continue.
There are however applications where graphical dialogs are really useful. Xdialog and dialog are very easy to use but of course not as powerful as a real graphical application. They fill the gap between a pure ASCII shell script and a full graphical application.

Where to get Xdialog and dialog?

The CDs of your linux distribution are the first place to look for dialog and Xdialog. It could be that they are already installed on your computer (ask your computer: e.g.

```
rpm -qil Xdialog, dpkg -L Xdialog).
```

References

Xdialog: http://www.chez.com/godefroy/
dialog: http://hightek.org/dialog/

Xdialog documentation:
http://www.chez.com/godefroy/doc/index.html

This article is re-printed with permission. The originals can be found at:

http://www.linuxfocus.org/English/November2002/article267.shtml

Programming Bits: Meeting C# and Mono

Author: Ariel Ortiz Ramirez <ariel.ortiz@itesm.mx>

Introduction

C# (pronounced C-sharp) is a new object-oriented programming language designed to take advantage of Microsoft's .NET development framework. It has many similarities with other popular object-oriented languages such as C++ and Java, yet it offers some new goodies.

Linux offers the opportunity to develop C# applications thanks to a project called Mono. Mono is an open source implementation of the .NET platform. In the following sections, I will describe the main elements of the current implementation of the Mono system.

The Mono Project

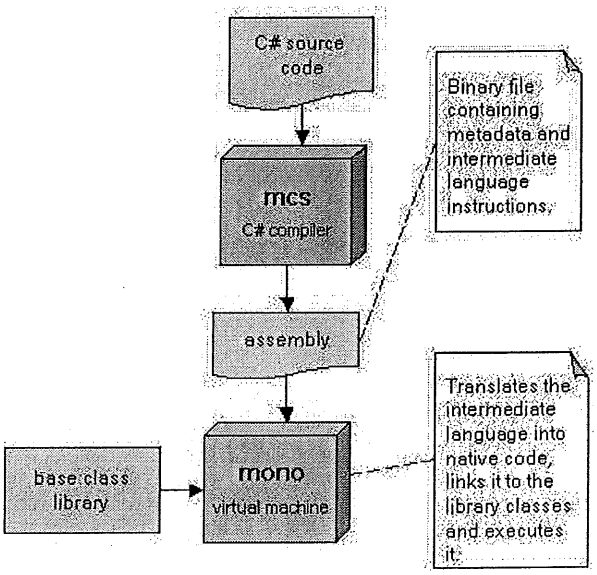
At this time, Mono implements two standards: the C# programming language (Standard ECMA-334) and the Common Language Infrastructure (Standard ECMA-335). Both of these specifications were developed by Microsoft and submitted to ECMA (a vendor consortium formerly known as the European Computer Manufacturers Association) on October 2000. They were formally approved on December 2001, and they will probably become ISO standards (thanks to a "fast-track" agreement that ISO has with ECMA) some time before the end of next year.

The Mono project is sponsored by Ximian, the same company that brought us the GNOME graphical desktop environment. Mexican hacker and Ximian CTO Miguel de Icaza currently leads the development of this project. In my opinion, the people involved with the development of Mono have done a remarkable job in quite a short amount of time. By the way, the word "Mono" means monkey in Spanish. These guys at Ximian really like monkeys.

Hello Mono World!

Lets follow a simple programming example in order to understand how C# and the different Mono components fit together. To see this in action, make sure you have a working Mono installation (see the resources section for information on downloading and installing Mono).

The following figure summarizes the process we will follow in order to compile and run our C# program:



First, we will create a simple C# source program (the classical "Hello World!" couldn't be missing). Type the following program using your favorite text editor and save the file as hello.cs:

```
class Hello {
    public static void Main() {
        System.Console.WriteLine("Hello Mono
```

```
World!");
        }
    }
```

This program is composed of a class named `Hello` which contains a method called `Main`. This method establishes the program entry point, in the same way that the main function is the start of a C/C++ program. In this example, the `Main` method prints to the standard output the message "Hello Mono World".

We can now compile the program using the Mono C# compiler, called mcs. At the shell prompt type:

```
mcs hello.cs
```

We now should have a file called **hello.exe** in the current directory. But don't get baffled about the .exe file name extension. It is not a Windows executable file, at least not in the way we're used to. Contrary to what happens when we compile a program written in languages like C or C++, the C# compiler does not generate a machine-specific object file (for example a Linux ELF x86 object file), but instead generates a special binary file called an **assembly**, which is made up of **metadata** and **intermediate language** (IL) instructions. These two together represent a platform-agnostic translation of the program source code. This means, of course, that when we actually run the program contained in the assembly, its intermediate language code has to be translated to the native code of the computer where the program is being run. This last translation phase is carried out by a **virtual machine**, whose behavior is defined by the **Common Language Infrastructure** (CLI) specification. The CLI defines an object oriented runtime environment that supports a base class library, dynamic class loading and linking, multiple thread execution, just-in-time compilation, and automatic memory management. The Microsoft implementation of the CLI specification is usually referred as the **Common Language Runtime** (CLR). We say that the CLR is a superset of the CLI because the CLR contains some extensions that are not part of the CLI.

To execute our assembly, we must invoke the program called `mono`, which is the Mono virtual machine. Type at the shell prompt the following:

```
mono hello.exe
```

The output should be:

```
Hello Mono World!
```

Behind the Scenes

Lets see how to examine the contents of our assembly. The program monodis (Mono disassembler) reads the binary information of an assembly and produces a textual representation of its contents. Type at the shell prompt:

```
monodis hello.exe
```

The disassembler output should be something like the following:

```
.assembly extern mscorlib
{
  .ver 0:0:0:0
}
.assembly 'hello'
{
  .hash algorithm 0x00008004
  .ver  0:0:0:0
}
  .class private auto ansi beforefieldinit Hello
        extends [mscorlib]System.Object
  {
    // method line 1
    .method public hidebysig  specialname
rtspecialname
          instance default void .ctor()  cil
managed
    {
        // Method begins at RVA 0x20ec
        // Code size 7 (0x7)
        .maxstack 8
        IL_0000: ldarg.0
        IL_0001: call instance void valuetype
[corlib]System.Object::.ctor()
        IL_0006: ret
    } // end of method instance default void
.ctor()
    // method line 2
    .method public static
          default void Main()  cil managed
    {
        // Method begins at RVA 0x20f4
        .entrypoint
        // Code size 11 (0xb)
        .maxstack 8
        IL_0000: ldstr "Hello Mono World!"
        IL_0005: call void class
[corlib]System.Console::WriteLine(string)
        IL_000a: ret
    } // end of method default void Main()
  } // end of type Hello
```

The first part of this output corresponds to the metadata. It contains information about the current version of the assembly, any optional security constraints, locale information, and a list of all externally referenced assemblies that are required for proper execution. The rest of the output represents the IL code. We can spot two methods in this code: the default class constructor called `.ctor`, provided automatically by the compiler, and our `Main` method. As described before, when the virtual machine is asked to run this code, it uses a just-in-time (JIT) compiler to translate the IL into the native machine code of the hosting environment. The native code is not generated until it is actually needed (thus the name just-in-time). For our example, the following is the native x86 machine code (in AT&T assembly language syntax) that gets generated for the `Main` method:

```
    push   %ebp
    mov    %esp,%ebp
    sub    $0x30,%esp
    push   $0x80c9eb0
    mov    0x805462c,%eax
    push   %eax
    cmpl   $0x0,(%eax)
    mov    (%eax),%eax
    call   *0x94(%eax)
    add    $0x8,%esp
    mov    0x805462c,%eax
    push   %eax
    cmpl   $0x0,(%eax)
    mov    (%eax),%eax
    call   *0xb4(%eax)
    add    $0x4,%esp
    leave
    ret
```

Mono also comes with an interpreter called `mint`. If you use this program, the IL instructions are interpreted instead of being compiled to native code

by the JIT. Actually, our simple program might be a little bit faster when run under `mint` because the JIT compiler will take some time to compile the code of our program and store it some where in memory. Of course, subsequent execution of the native code already in memory is definitely faster than interpretation. Currently the Mono JIT compiler is only available for x86 machines. The Mono interpreter must be used in any non-x86 machine. To see the interpreter running, type at the shell prompt:

```
mint hello.exe
```

If you're familiar with Java, you might be thinking that all this technology sounds pretty much like the way that the Java platform works. And this is indeed so. The CLI virtual machine is the key factor for platform independence. This means that I can write and compile a program in Linux using Mono, and then run it in a Windows computer with the .NET framework. There is no need to rewrite or recompile the source code. But in contrast to the Java Virtual Machine, which is tightly coupled to the Java programming language, the CLI specification not only allows platform independence, it also allows language independence. Windows has compilers that target the CLR from a number of languages. The most important ones are part of Microsoft's Visual Studio .NET development environment: Visual Basic .NET, JScript .NET, Managed C++ and C#. Other languages supported, from third party vendors, include APL, COBOL, Eiffel, Forth, Fortran, Haskell, Mercury, Mondrian, Oberon, Pascal, Perl, Python, RPG, Scheme and SmallScript. The Mono project only has a C# compiler at this time, but we will probably see in the near future other languages being supported.

Another important element of the CLI is the **Common Type System** (CTS). The CTS fully describes all the data types supported by the virtual machine, including how these data types interact with each other and how they are represented as metadata inside the assemblies. It is important to note that not all languages available for the CLI support all data types in the CTS. So there is a **Common Language Specification** (CLS), that defines a subset of common types that ensure that binary assemblies can be used across all languages that target the CLI. This means that if we build a CLI class that only exposes CLS compliant features, any CLI compatible language can use it. You could create a class in Eiffel, subclass it in C# and instantiate it in a Visual Basic.NET program. Now this is real language interoperability.

Some Advantages

Using a CLI compliant platform, such as Mono or the .NET framework, has some important advantages:

- Programs can be run without recompiling on any operating system and processor that supports the platform.
- There is complete multiple language integration.
- The system supports important security measures.
- A common runtime engine is shared by all CLI aware languages.
- A consistent object model is used by all CLI aware languages, including a standard API offered by a single base class library. Once you learn this API, you can use it in any language supported by the platform.
- There is a simplified deployment model. There is no need to register a binary unit into the system registry.
- Multiple versions of the same binary library (DLL) can coexist in harmony on the same computer.

C#, as a programming language, has also some important features:

- It includes constructs such as properties, events and attributes that ease the construction of software components.
- It does not require the use separate header of interface definition language (IDL) files.
- It has a simplified versioning mechanism.
- It's type safe and has a unified type system. All data types (including primitive types) derive from a single base class.
- It has automatic memory management, through the use of garbage collection.
- It's closely integrated to the CLI.

I will discuss these and other C# issues more thoroughly in later articles.

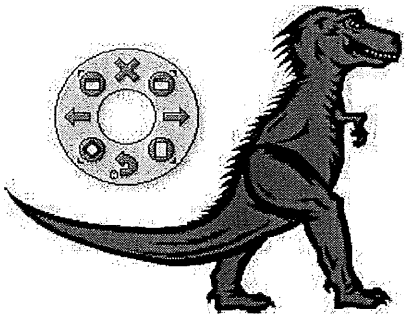
Resources

- http://www.go-mono.com/
  The official Mono home page. The download and install instructions can be found here.
- http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/deicazainterview.asp
  A very interesting interview with Miguel de Icaza about the Mono project and the use of ECMA standards.
- http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cscon/html/vcoriCStartPage.asp
  Information on the C# programming language at MSDN.
- http://www.ecma.ch/ecma1/STAND/ecma-334.htm
  The Standard ECMA-334 C# Language Specification.
- http://www.ecma.ch/ecma1/STAND/ecma-335.htm
  The Standard ECMA-335 Common Language Infrastructure.

This article is re-printed with permission. The originals can be found at:
http://www.linuxgazette.com/issue84/ortiz.html

Mozilla dissected

Author: Floris Lambrechts <floris@linuxfocus.org>



Abstract

After the long waiting time, Mozilla has already become an old animal. They bread fast in the Mozilla family: the new generation (`1.0.1` and `1.1`) has emerged and when you read this, `1.2` may just be born. After many hours of testing and fiddling with the browser capabilities of these youngsters, I describe my experiences below.

In the first part I take a look at what's available in the standard browser, and then I review two of the many available add-ons.

Installation

First off, you have to choose a version. `1.0.1` is actually the '*one and only*™' Mozilla `1.0`, but with a few less bugs and some security issues fixed. If you prefer stability and compatibility, this is the one for you. Otherwise you take version `1.1` or later - that's where all the new functionality will show up. You can track the development on the road map http://www.mozilla.org/roadmap.html.

The installation went very smoothly: just choose the Linux x86 Net Installer on the 'download' page and execute it (sometimes also available in other languages). The Net Installer (less than 100 kB) allows you to choose which Mozilla components you like to have - if you take the 'Custom' installation - and after that it does its job. They just can't make it simpler than this.

By default, it installs in `/usr/local/mozilla`, but you can change the location. It's possible to install different Mozilla versions alongside each other - my `/usr/bin/mozilla` is just a simple shell script that goes to the right directory and starts '`mozilla &`'. The different versions are hard to distinguish from each other - the differences that I noted are explained in the text below. You can check what version you are running in the Help menu (`Help=>About Mozilla`).

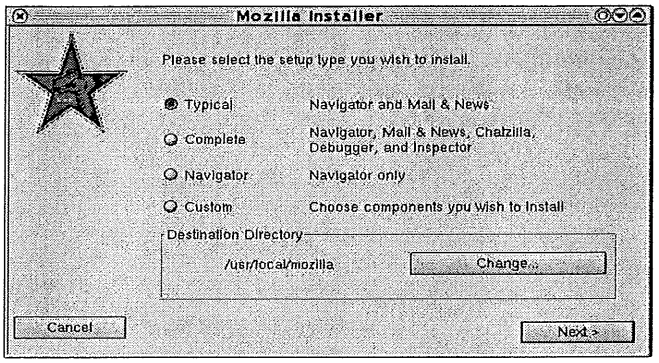


Figure 1: A simple installation process

The fat boy

Mozilla is more than just a browser. In general, it is a platform for developers and for personal communication (and the platform is in turn platform-independent :). There is the browser, the news/mail client, the chat program, the html editor, JavaScript debugger, XML tools, the 'interface description language' XUL, libraries for downloading and installing applications and components and so on.

Never forget that in a lot of cases, you just don't need stuff like ChatZilla (the IRC chat-client), the Composer, the Mailer/newsreader or the address book. If all you want is the browser, it's a smart thing to only install what you need (don't forget to choose the 'Custom' install in that case). As a simple, plain browser Mozilla has of course though competition, Galeon for example. That one uses the heart of Mozilla, (the *rendering engine* Gecko), but everything else is lighter and smaller. And also Konqueror of KDE and Opera are still around, both worthy alternatives.

If you choose to use Mozilla, you'll have to accept the fact that it uses a lot of memory. I never found that it felt 'slow', and the startup-time also has also improved since the days before `1.0`. But Mozilla still likes a huge chunk of RAM, just like KDE by the way. Sad but true. For some older systems a RAM upgrade is probably something you can consider.

> Just a remark: I often hear that 'the Linux desktop' is really heavy compared to 'Windows'. Agreed, KDE is not small, but you have to know what you're comparing with! It can do much more than -say- Windows98. In fact you should compare a recent Linux with a server edition of Windows XP (although... ;-). And then it shows: Linux does a great job. For older systems you just have to seek out a lighter window manager/desktop - and your system will be just as fast. Or just run your desktop on the heavy server in the network... I also hear that the new Gnome needs less resources than its predecessor, which I find a very good development.

A problem with 'alternative' browsers is that some sites become unusable. Let it be clear that Mozilla is not an 'alternative' browser! There are very few sites that display errors, or nothing at all. And in that case

you can safely blame the site designer: Mozilla follows the standards almost perfectly, a very important accomplishment. Congratulations to the developers! (and death to all the webmasters that make sloppy sites on purpose!)

This being said, let's investigate whether the bells and whistles of the browser are of any use to us.

THE BUILT-IN BELLS...

The amount of included features and options is quite impressive. You get a good idea of the possibilities by simply browsing the well-structured Preferences menu (`Edit=>Preferences`). Below we take a look at many of these features.

Tabbed browsing

Tabbed browsing is pretty well-known by now. It is a system that allows you to open multiple sites in a single window. Each page has its own 'tab', on which you can click to 'activate'. It is mainly useful to avoid having tens of Mozilla windows open at once. I use and like the tabs, but something bothers me: why does every application has to invent yet another *multiple document interface*? Mozilla has such a `MDI`, Kate the KDE editor, Opera and Galeon too... Each one with different concepts and different key bindings. And then there is KDE itself, which groups windows of the same program under a single button in the task bar. Added together, this makes for a very confusing situation - but in the end I think Mozilla has one of the better systems. Note that Konqueror from KDE 3.1 on will also support http://mozillaquest.com/News02/KDE_Konqueror_Tabs_Coming_Story01.html tabbed browsing.

Be sure to take a look at the options for tabbed browsing. I like to 'load tabs in the background', and I make sure that clicking the middle mouse button over a link opens a new tab. You can also make sure that Control+Enter in the location bar opens a new tab. The best thing about the tabs are the key bindings: Control+T opens a new empty tab, Control+Page Up and Control+Page Down change between tabs, and Control+W closes the current one. And finally, just for convenience: Control+L selects the text (URL) in the location bar.

Bookmarks

The nice thing about the bookmarks (that come with an extensive management system) is the ability to save multiple tabs under one bookmark (`Bookmarks=>File Bookmark`, and then choose 'File as group'). In version `1.1` it can be done even faster: simply choose `Bookmarks=>Bookmark This Group of Tabs`. Very nice to open up a standard set of news sites for example, or to start surfing in the morning on the pages you had open the day before. I only mis a little feature from Konqueror, where you can not only `Add to Bookmarks` in the main `Bookmarks` menu, but also in its sub-menu's. But don't worry, so far I can still live with it ;-).

The sidebar

With F9 you turn on the sidebar (or, rather, turn it off :-). By clicking on its border, you can minimise it to a couple of pixels width. What exactly would make the sidebar an advantage over a nice simple website is unclear to me, although I suspect that Netscape uses them extensively in their browsers. Inside it you can see tabs like 'search', 'history', 'what's related' and 'bookmarks'.

Luckily anyone can make their own sidebars, and as a result many others are available. Some sites offer a 'Netscape sidebar' to track their news messages. I tried some that contain information about the tags of the document formats HTML, XHTML and SVG (from http://www.zvon.org/Output/bar_list.html, who by the way do translations of their pages the LF way). Installation of these sidebars is straightforward: click, choose 'OK' and everything will be all right, even when you're not root. I personally would like to see a sidebar that validates the current page with http://validator.w3.org. As a lazy but well-meaning html writer I would really welcome such a thing ;-)

PRIVACY AND SECURITY

This is a domain where Mozilla really shines. There are lots of features and possibilities regarding cookie management, pop-up prevention... I recommend the options that only accept images and cookies from the same web server that provides the actual website you read. This can already stop or cripple quite some Adv. networks. And there is more: you can view and remove individual cookies, or choose which sites have permission to 'cookie you'. Mozilla can also just ask permission for each single cookie, and can remember your choices at that. Consider whether you need Java, JavaScript and cookies and turn off as much as possible (by default Mozilla accepts all cookies and JavaScript has the right to open pop-ups - turn it off asap!).

Mozilla is also able to remember passwords and form data; this is something I turn of immediately. (Yuk, I just don't trust it.)

APPEARANCE

You can configure which buttons should be visible in the toolbar - 'Home', 'Search', 'Print', 'Go' and 'Bookmarks' are not necessary in my view. This yields more space on the screen to display the URL. Configurable in `Edit=>Preferences=>Navigator`.

I do not really like the standard look, the 'Classic' theme. It resembles the old and grey Netscape 4 and makes it difficult to recognise the active tab. Compared with 'Classic', I find the also supplied 'Modern' theme a whole lot better. There are several other themes on the web, so everyone should be able to find her favourite. For smaller displays I would recommend a 'tiny' theme; I myself use the colourful and small 'Orbit 3m'.

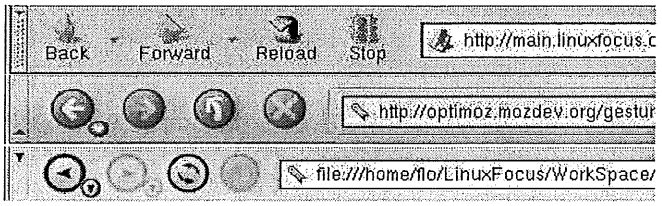


Figure 2: Three themes: Classic, Modern and the small version of Orbit, 'Orbit 3m'.

SMART BROWSING

Using smart browsing, you don't always have to type URLs. In the first place, there are the 'Internet keywords', actually a genuine Netscape.com thing. Certain keywords, such as 'seas', 'quote', 'help' and 'goto' are recognised when you type them in the location bar (like 'search linuxfocus' of 'goto linuxfocus'). Mozilla will then recognise the keywords, and opens a certain site with the right search parameters. The only thing I like about this is that you can add your own keywords (using JavaScript) for your favourite sites. Mozilla also has a setting for a standard search engine (by default netscape.com, but google.com is also an option), which is also called when when you type single words in the location bar. This last feature was not present in some of the Mozilla versions I tested. All in all, I never got used to using this 'Internet keywords' stuff, so I always turn it off now. Maybe it's more useful for people who type very slowly :-) The other part of smart browsing is the 'location bar auto complete', something that most current browsers already have. I'm not lyrical about it :-)

Type Ahead Find

Mozilla `1.2`, of which I was only able to test the alpha version, contains the new feature called 'Type Ahead Find'. This is a nice way to navigate without using the mouse: you just type a part of the link you want to activate, and Mozilla will select it. Then an `Enter` will do to follow the link. For example: you want to follow the first link below this line (Plugins and...), so you type `plu+Enter` and Mozilla will take care of the rest. Once you know enough key bindings, it becomes very easy to navigate textually without switching between the keyboard and the mouse. A nice extra feature is that you can also search regular text by beginning the search with a /slash. Other things to remember: `Escape` cancels the current search, and with (`Shift+`)`F3` you can search forwards (or backwards).

Figure 3: Type Ahead Find activates the right link after typing '`cast`' on the LinuxFocus.org main page.

AND THE WHISTLES

Aside from the standard features, you can get a lot of extras for the browser. Mozilla's openness makes it relatively simple to build extensions for it, and that has obviously paid off. There are the 'old-school' plugins (see the article: Plugins and Mozilla `1.0` http://www.linuxfocus.org/English/July2002/article248.shtml), there are sidebars that start to resemble complete applications, and then there are the other add-ons. By lack of time space we can not describe them all, but there are ad blockers, surf anonymisers, a calendar, spellcheck, Jabber messenger, a site downloader, ... Following the links at the bottom: MozDev lists a lot of add-ons. The following addons were tested by yours truly.

Mother tongue

Mozilla speaks English of course, but that is not all. Installing other languages is easy: clicking trough in `Edit=>Preferences=>Appearance=>Languages/Content` will lead you to a page where all the translations are listed. Clicking will do for the installation, at least if you have root rights and your Mozilla version is not struggling (it once refused the translation, but kept on working fine as always). In some (or all?) languages you will see that the keyboard shortcuts are not translated, which is partly very convenient and partly annoying. I was told that translating the shortcuts is a source of conflicts, and no software exists for Mozilla to detect and to help solve these conflicts. Also be aware of the fact that not all Mozilla builds have translations in your preferred language - you can check that in advance http://www.mozilla.org/projects/l10n/mlp_status.html#contrib.

If you're not a native English speaker, don't forget to set up your favourite languages (`Navigator=>Languages`), as this setting is used by some sites to automatically show you a translated version of their pages (e.g. http://www.debian.org).

Mouse gestures

One of the projects I stumbled upon on the MozDev.org site is called Optimoz. These people make two of the nicer add-ons: 'Mouse gestures' and 'RadialContext'. Mouse gestures (also known as MozGest) is a system for controlling the computer with the mouse movements you make, not with buttons or menus. By drawing a 'shape' on the screen with a certain mouse button pressed down, you perform a specific action. The Windows game Black&White also uses mouse gestures, just like the Opera browser in its Windows version. You can view the list with defined gestures at Optimoz http://optimoz.mozdev.org/gestures/index.html. In addition, it is possible to define your own gestures by editing the JavaScript code.

Figure 4: Optimoz mouse gestures banner...

The installation is smooth as always, provided you have root rights. I really enjoyed playing with mouse gestures, but it never made me 'addicted' to it. Most often clicking on buttons (or using keyboard shortcuts) is just as fast, and sometimes a 'gesture' was mis-interpreted by the software (i.e. I performed it incorrectly). But the possibility of creating your own gestures is very promising...

RadialContext

The other part of Optimoz is 'RadialContext'. This is an implementation of a so-called 'pie menu', a kind of round menu system that appears around the cursor. By moving the mouse in 1 of the 4 (or 8) directions you make a menu choice. It is kind of difficult to explain, but you can see it with your own eyes if you are using Mozilla: just surf to http://optimoz.mozdev.org/piemenus/ and see the look & feel in action. This is just a demonstration: the actions you perform will have no effect until you install RadialContext yourself (simple if you are root, as always).

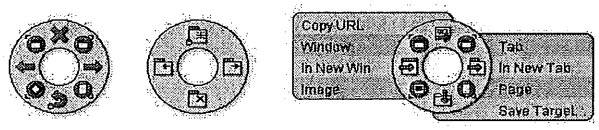


Figure 5: A view on RadialContext: the main menu, the tab menu and the image menu with help text.

In the image you see the standard pie menu containing four functions in the directions up, down, left and right. In between are four sub-menus which are accessed by moving the mouse pointer diagonally in the corresponding direction without releasing the mouse button. Top-right, for example, there is the menu of the tabs (shown in the middle part of the image). This sub-menu has functions to open or close a tab, and to go to the next or previous one. This is pretty convenient to browse without touching the keyboard. The other sub-menus deal with functions concerning 'window', 'tasks' and 'page'. In the 'normal' menu, the four main functions serve to reload the page, to stop the loading, or to go back or forth in the history. Used over an image, the four main directions get different functions, as you can see in the right part of the image. There you can also see what happens when you don't move the mouse for a while: a little text explanation appears next to the menu choices. Used over a link, the menu changes once again, this time providing functions to open the link in a new window or in a new tab, to copy the link or to save its target.

A nice overview of the menu contents is at http://www.gamemakers.de/mozilla/radialcontext/functionality.html.

Of course you can configure which mouse button RadialContext occupies; there is also a key available to access the 'normal' use of that mouse button as well. A little tip: do not use RadialContext together with MozGest - that only confuses the browser. Possible improvements (according to me) could be: a menu that does not move once it's appeared, and showing the little help text by default (reducing the useless waiting time).

I notice that on my system mouse gestures are always turned off (easy to do in the preferences), and I use the pie menu a lot. Often enough I don't even wait for the menu to appear; I have learned to memorise some of the actions and a 'wrong' movement is very rare. When my hands are on the keyboard I often use shortcut keys, but if not I find nearly everything I need in the little round menu. I think it's an innovative and efficient system and I'm starting to really miss it in other browsers. Add to that the fact that RadialContext is updated regularly and the verdict is clear: I recommend you to try it out yourself sometime.

ANFSCD*: MOZILLA & PRESENTATIONS

* And now for something completely different

Presentation are performed in thousands of ways, and the software used can vary almost just as much. Popular programs are, among others, OpenOffice, KPresenter and PDF- or PostScript readers. Ignoring these last two, most programs use their own format (or that of Microsoft's PowerPoint). As a result the presentations are not readable on all systems, which is a big disadvantage when you can not use your own computer.

But actually, HTML fits really well for a solid presentation, certainly in combination with style sheets. This ensures that you can display on all available platforms, and if you can use Mozilla you can even do it full-screen (under Linux full-screen only works on `1.1` and later). Just hit `F11`... It's also possible to describe your presentation in a home-made XML language, and display it nicely with CSS - of course this involves a risk with browsers whose XML or CSS support is not adequate. But it certainly works in Mozilla - I've tested it.

Are those tiny buttons still distracting your audience in full screen mode? `View=>Show/Hide=>Navigation Toolbar` solves the problem. With RadialContext you no longer need menus and buttons! You can still type an address if you open the location bar beforehand with `Control+L`. Right, now make sure that your page is smaller than the display and even the scrollbar on the side will disappear...

LINKS

- http://galeon.sourceforge.net: Galeon, light browser based on Gecko, the Mozilla rendering engine.
- http://www.opera.com: Opera, good little browser for Linux and others.
- http://www.mozilla.org: official site of the Mozilla developers.
- http://www.mozilla.org/releases/stable.html: here are the installation programs for the stable releases. Less stable, newer releases can be found at http://www.mozilla.org/releases/.

- http://www.mozilla.org/start/1.0/: default start page of Mozilla `1.0` has good information.
- http://www.mozillazine.org: online magazine 'bout the moz.
- http://www.mozdev.org: is home to more than 70 Mozilla-related projects
- http://www.mozdev.org/projects.html is the impressive list of projects.
- http://optimoz.mozdev.org: Optimoz, source of 'Mouse gestures' and 'RadialContext'.
- http://www.opera.com/support/operashow/: use Opera and style sheets to give presentations.

As dutch editor of LinuxFocus I work on the magazine every week. At this moment I am mostly proofreading other people's translations to dutch. In the non-virtual world I was just recently trying to improve my juggling skills - I do fairly well now with 3 clubs, but 4 balls at once is still one too much for me.

Translated to English by Floris Lambrechts <floris@linuxfocus.org>

Process Tracing Using Ptrace, part 3

Author: Sandeep S <busybox@sancharnet.in>

The basic features of `ptrace` were explained in Part I (AUUGN 23.3). In Part II (AUUGN 23.4) we saw a small program which accessed the registers of a process and modified them so as to change the output of that process, by injecting some extra code. This time we are going to access the memory of a process. The purpose of this article is to introduce a methods for infecting binaries on runtime. There are many possible areas of use for this technique.

Introduction.

We are familiar with `ptrace` and know the techniques of attaching a process, how to trace it and finally to free it. We also have an idea about the structure of the Linux binary format - ELF.

Our plan is to fetch/modify a running binary. So we have to locate the symbols inside the binary. There we need `link_map`. link_map is the dynamic linker's internal structure with which it keeps track of loaded libraries and symbols within libraries.

The format of `link_map` is (from `/usr/include/link.h`)

```
struct link_map
{
  ElfW(Addr) l_addr;  /* Base address shared
                         object is loaded at. */
  char *l_name;       /* Absolute file name object
                         was found in. */
  ElfW(Dyn) *l_ld;    /* Dynamic section of the
                         shared object. */
  struct link_map *l_next, *l_prev;
                      /* Chain of loaded objects */
};
```

A small explanation for the fields.

1. **l_addr**: Base address where shared object is loaded. This value can also be found from `/proc/pid/maps`
2. **l_name**: pointer to library name in string table
3. **l_ld**: pointer to dynamic (DT_*) sections of shared lib
4. **l_next**: pointer to next `link_map` node
5. **l_prev**: pointer to previous `link_map` node

Link-map is a linked list, each item on list having a pointer to loaded library. What we have to do is, to follow this chain, go through every library and find our symbol. Now we have a question. Where we can find this `link_map`?

For every object file, there is a global offset table (GOT) which contains many details of the binary. In GOT, the second entry is dedicated for the `link_map`. So we get the address of `link_map` from **GOT[1]** and we go on searching our symbol.

Straight to code.

Now we have collected the basic information needed to access the memory. Let's start now. First of all we attach the process 'pid' for tracing. Now we go for finding out the `link_map` we require. You will find functions `read_data`, `read_str` etc. These are helper functions to make working with `ptrace` easier. Helper functions are self explaining.

The function for locating the `link_map` is:

```
struct link_map *locate_linkmap(int pid)
{
  Elf32_Ehdr *ehdr = malloc(sizeof(Elf32_Ehdr));
  Elf32_Phdr *phdr = malloc(sizeof(Elf32_Phdr));
  Elf32_Dyn *dyn = malloc(sizeof(Elf32_Dyn));
  Elf32_Word got;
  struct link_map *l =
      malloc(sizeof(struct link_map));
  unsigned long phdr_addr, dyn_addr, map_addr;

  read_data(pid, 0x08048000, ehdr,
      sizeof(Elf32_Ehdr));
  phdr_addr = 0x08048000 + ehdr->e_phoff;
  printf("program header at %p\n", phdr_addr);
  read_data(pid, phdr_addr, phdr,
      sizeof(Elf32_Phdr));

  while (phdr->p_type != PT_DYNAMIC) {
    read_data(pid, phdr_addr += sizeof(Elf32_Phdr),
        phdr, sizeof(Elf32_Phdr));
  }

  read_data(pid, phdr->p_vaddr, dyn,
      sizeof(Elf32_Dyn));
  dyn_addr = phdr->p_vaddr;

  while (dyn->d_tag != DT_PLTGOT) {
    read_data(pid, dyn_addr += sizeof(Elf32_Dyn),
        dyn, sizeof(Elf32_Dyn));
  }

  got = (Elf32_Word) dyn->d_un.d_ptr;
  got += 4;       /* second GOT entry, remember? */
```

```
  read_data(pid, (unsigned long) got,
      &map_addr, 4);
  read_data(pid, map_addr, 1,
      sizeof(struct link_map));
  free(phdr);
  free(ehdr);
  free(dyn);
  return 1;
}
```

We start from the location 0x08048000 to get elf header of the process we are tracing. We get the elf header and from its fields we can get the program header. (The fields of headers were discussed in Part II http://www.linuxgazette.com/issue83/sandeep.html) Once we get the program header, we go on checking for the header with dynamic linking information. From the header/struct with dynamic linking information, we fetch the location of the information. Go on searching until we get the base address of global offset table.

Now we have the address of GOT with us and take the second entry of GOT (there we have `link_map`). From there get the address of the `link_map` which we require and return.

We have the `struct link_map` and we have to get `symtab` and `strtab`. For this, we move to `l_ld` field of `link_map` and traverse through dynamic sections until DT_SYMTAB and DT_STRTAB have been found, and finally we can seek our symbol from DT_SYMTAB. DT_SYMTAB and DT_STRTAB are the addresses of symbol table and string table respectively.

The function resolv_tables is:

```
void resolv_tables(int pid, struct link_map *map)
{
  Elf32_Dyn *dyn = malloc(sizeof(Elf32_Dyn));
  unsigned long addr;
  addr = (unsigned long) map->l_ld;
  read_data(pid, addr, dyn, sizeof(Elf32_Dyn));
  while (dyn->d_tag) {
    switch (dyn->d_tag) {
    case DT_HASH:
      read_data(pid, dyn->d_un.d_ptr +
          map->l_addr + 4,
          &nchains, sizeof(nchains));
      break;
    case DT_STRTAB:
      strtab = dyn->d_un.d_ptr;
      break;
    case DT_SYMTAB:
      symtab = dyn->d_un.d_ptr;
      break;
    default:
      break;
    }
    addr += sizeof(Elf32_Dyn);
    read_data(pid, addr, dyn, sizeof(Elf32_Dyn));
  }
  free(dyn);
}
```

What we actually do here is just reading dynamic sections one by one and checks whether the tag is DT_STRTAB or DT_SYMTAB. If yes, we can get their respective pointers and assign to `strtab` and `symtab`. Once the dynamic sections are over, we can stop.

Our next step is getting the value of symbol from the symbol table. For this we take every symbol table entry one by one and check it whether it's a function name. (We are interested in finding the value of a library function). If it is then it's compared with the function name given by us. If here also they match now the value of the symbol is returned.

Now we have got the value of the symbol what we actually required. What help will the value do for us? The answer depends upon the reader. As I have already stated we may use this for both good and evil purposes.

You might be thinking that everything is over. We forgot a step that we shouldn't forget - detaching the traced process. This may leave the process in a stopped state for ever and the consequences are already discussed in Part I of this series http://www.linuxgazette.com/issue81/sandeep.html. So our last and final step is to detach the traced process.

The `Ptrace.c` program may be obtained from. http://www.linuxgazette.com/issue85/misc/sandeep/Ptrace.c.txt
Almost the whole code is self-explanatory.

Compile it by typing

```
#cc Ptrace.c -o symtrace
```

Now we want to test the program. Run some process in some other console, come back and type. (Here my test program is `emacs` and the symbol I give is `strcpy`). You may trace any program that is traceable instead of emacs and any symbol you want to inspect.

```
#./symtrace `ps ax | grep 'emacs' |
          cut -f 2 -d " "` strcpy
```

and watch what is going on.

Conclusion

So, we come to the end of a series of three articles which has gone through the basic programming with `ptrace`. Once you have understood the basic concept it is not difficult to make steps by your own. More details on `ptrace` and elf are available at http://www.phrack.org. One more thing I have to write is that, we reached here without even mentioning a major topic. One major feature of `ptrace` is its play with system calls. In User Mode Linux, this feature is used in a large scale. I am busy with my classes and final year project, and I promise, if time permits we will continue this series and then we will have a look at those features of `ptrace`.

All Suggestions, Criticisms, Contributions etc. are welcome. You can contact me at busybox@sancharnet.in (changed since last issue).

Sandeep is a final year student of Government Engineering College in Thrissur, Kerala, India. His interests include FreeBSD, Networking and also Theoretical Computer Science.

http://www.linuxgazette.com/copying.html.
Published in Issue 85 of *Linux Gazette*, October 2002

This article is re-printed with permission. The original can be found at:

http://www.linuxgazette.com/issue83/sandeep.html

Concurrent programming - Principles and introduction to processes

Author: Leonardo Giordani <leo.giordani@libero.it>
Translated to English by: Leonardo Giordani

Abstract

This series of articles has the purpose of introducing the reader to the concept of multitasking and to its implementation in the Linux operating system. Starting from the theoretical concepts at the base of multitasking we will end up writing a complete application demonstrating the communication between processes, with a simple but efficient communication protocol. Prerequisites for the understanding of the article are:

- Minimal knowledge of the shell
- Basic knowledge of C language (syntax, loops, libraries)

All references to manual pages are placed between parenthesis after the command name. All the glibc functions are documented in gnu info pages (info Libc, or type info:/libc/Top in konqueror).

Introduction

One of the most important turning points in the history of operating systems was the concept of multiprogramming, a technique for interlacing the execution of several programs in order to gain a more constant use of the system's resources. Let's think of a simply workstation, where a user can execute at the same time a word processor, an audio player, a print queue, a web browser and more. It's an important concept for modern operating systems. As we will discover this little list is only a minimal part of the set of programs that are currently executing on our machine, even though the most "visual-striking".

The concept of process

In order to interlace programs a remarkable complication of the operating system is necessary; in order to avoid conflicts between running programs an unavoidable choice is to encapsulate each of them with all the information needed for their execution.

Before we explore what happens in our Linux box, let's give some technical nomenclature: given a running **PROGRAM**, at a given time the **CODE** is the set of instructions which it's made of, the **MEMORY SPACE** is the part of machine memory taken up by its data and the **PROCESSOR STATUS** is the value of the microprocessor's parameters, such as the flags or the Program Counter (the address of the next instruction to be executed).

We define the term **RUNNING PROGRAM** as a number of objects made of CODE, MEMORY SPACE and PROCESSOR STATUS. If at a certain time during the operation of the machine we will save this informations and replace them with the same set of information taken from another running program, the flow of the latter will continue from the point at which it was stopped: doing this once with the first program and once with the second provides for the interlacing we described before. The term **PROCESS** (or **TASK**) is used to describe such a running program.

Let's explain what was happening to the workstation we spoke about in the introduction: at each moment only a task is in execution (there is only a microprocessor and it cannot do two things at the same time), and the machine executes part of its code; after a certain amount of time named QUANTUM the running process is suspended, its informations are saved and replaced by those of another waiting process, whose code will be executed for a quantum of time, and so on. This is what we call multitasking.

As stated before the introduction of multitasking causes a set of problems, most of which are not trivial, such as the waiting processes queues management (SCHEDULING); nevertheless they have to do with the architecture of each operating system: perhaps this will be the main topic of a further article, maybe introducing some parts of the Linux kernel code.

Processes in Linux and Unix

Let's discover something about the processes running on our machine. The command which gives us such informations is **ps(1)** which is an acronym for "process status". Opening a normal text shell and typing the ps command we will obtain an output such as

```
PID TTY          TIME CMD
2241 ttyp4    00:00:00 bash
2346 ttyp4    00:00:00 ps
```

I state in before that this list is not complete, but let's concentrate on this for the moment: ps has given us the list of each process running on the current terminal. We recognize in the last column the name by which the process is started (such as "mozilla" for Mozilla Web Browser and "gcc" for the GNU Compiler Collection). Obviously "ps" appears in the list becouse it was running when the list of running processes was printed. The other listed process is the Bourne Again Shell, the shell running on my terminals.

Let's leave out (for the moment) the information about

TIME and TTY and let's look at PID, the Process Identifier. The pid is a unique positive number (not zero) which is assigned to each running process; once the process has been terminated the pid can be reused, but we are guaranteed that during the execution of a process its pid remains the same. All this implies is that the output each of you will obtain from the ps command will probably be different from that in the example above. To test that I am saying the truth, let's open another shell without closing the first one and type the ps command: this time the output gives the same list of processes but with different pid numbers, testifying that they are two different processes even if the program is the same.

We can also obtain a list of all processes running on our Linux box: the ps command man page says that the switch -e means "select all processes". Let's type "ps -e" in a terminal and ps will print out a long list formatted as seen above. In order to comfortably analyze this list we can redirect the output of ps in the ps.log file:

```
ps -e > ps.log
```

Now we can read this file editing it with our preferred editor (or simply with the less command); as stated at the beginning of this article the number of running processes is higher than we would expect. We actually note that list contains not only processes started by us (through the command line or our graphical environment) but also a set of processes, some of which with strange names: the number and the identity of the listed processes depends on the configuration of your system, but there are some common things. First of all, no matter what type of configuration you gave to the system, the process with pid equal to 1 is always "init", the father of all the processes; it owns the pid number 1 because it is always the first process executed by the operating system. Another thing we can easily note is the presence of many processes, whose name ends with a "d": they are the so called "daemons" and are some of the most important processes of the system. We will study in detail init and the daemons in a further article.

MUTITASKING IN THE LIBC

We understand now the concept of process and how important it is for our operating system: we will go on and begin to write mutitasking code; from the trivial simultaneous execution of processes we will shift towards a new problem: the communication between concurrent processes and their synchronization; we will discover two elegant solutions to this problem, messages and semaphores, but the latter will be deeply explained in a further article about the threads. After the messages it will be the time to begin writing our application based on all these concepts.

The standard C library (libc, implemented in Linux with the glibc) uses the Unix System V multitasking facilities; the Unix System V (from now on SysV) is a commercial Unix implementation, is the founder of one of the two most important families of Unix, the other being BSD Unix.

In the libc the pid_t type is defined as an integer capable of containing a pid. From now on we will use it to bear the value of a pid, but only for clarity's sake: using an integer is the same thing.

Let's discover the function which give us the knowledge of the pid of the process containing our program.

```
pid_t getpid (void)
```

(which is defined with pid_t in unistd.h and sys/types.h) and write a program whose aim is to print on the standard output its pid. With an editor of your choice write the following code

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>

int main()
{
   pid_t pid;

   pid = getpid();
   printf("The pid assigned to the process is
%d\n", pid);

   return 0;
}
```

Save the program as print_pid.c and compile it

```
gcc -Wall -o print_pid print_pid.c
```

this will build an executable named print_pid. I remind you that if the current directory is not in the path it is necessary to run the program as "./print_pid". Executing the program we will have no great surprises: it prints out a positive number and, if executed more than once, you see that this number will increase one by one; this is not mandatory, because another process can be created from a program between an execution of print_pid and the following. Try, for example, to execute ps between two executions of print_pid...

Now it's time to learn how to create a process, but I have to spend some more words about what really happens during this action. When a program (contained in the process A) creates another process (B) the two are identical, that is they have the same code, the memory full of the same data (not the same memory) and the same processor status. From this point on the two can execute in two different ways, for example depending on the user's input or some random data. The process A is the "father process" while the B is the "son process"; now we can better understand the name "father of all the processes" given to init. The function which creates a new process is

```
pid_t fork(void)
```

and its name comes from the property of forking the execution of the process. The number returned is a pid, but deserves a particular attention. We said that the present process duplicates itself in a father and a son, which will execute interlacing themselves with the other running processes, doing different work; but

immediately after the duplication which process will be executed, the father or the son? Well, the answer is simply: one of the two. The decision of which process has to be executed is taken by a part of the operating system called scheduler, and it pays no attention if a process is the father or the son, following an algorithm based on other parameters.

Anyway, it is important knowing what process is in execution, because the code is the same. Both processes will contain the father's code and the son's one, but each of them has to execute only one of this codes. In order to clarify this concept let's look at the following algorithm:

- FORK
- IF YOU ARE THE SON EXECUTE (...)
- IF YOU ARE THE FATHER EXECUTE (...)

which represents in a sort of meta language the code of our program. Let's unveil the mystery: the fork function returns '0' to the son process and the son's pid to the father. So it is sufficient to test if the returned pid is zero and we will know what process is executing that code. Putting it in C language we obtain

```
int main()
{
   pid_t pid;

   pid = fork();
   if (pid == 0)
   {
      CODE OF THE SON PROCESS
   }
   CODE OF THE FATHER PROCESS
}
```

It's time to write the first real example of multitasking code: you can save it in a fork_demo.c file and compile it as done before. I put line numbers only for clarity. The program will fork itself and both the father and the son will write something on the screen; the final output will be the interlacing of the two output (if all goes right).

```
(01) #include <unistd.h>
(02) #include <sys/types.h>
(03) #include <stdio.h>

(04) int main()
(05) {
(05)    pid_t pid;
(06)    int i;

(07)    pid = fork();

(08)    if (pid == 0){
(09)      for (i = 0; i  < 8; i++){
(10)        printf("-SON-\n");
(11)      }
(12)      return(0);
(13)    }

(14)    for (i = 0; i < 8; i++){
(15)      printf("+FATHER+\n");
(16)    }

(17)    return(0);
(18) }
```

Lines number (01)-(03) contain the includes for the necessary libraries (standard I/O, multitasking).

The main (as always in GNU), returns an integer, which normally is zero if the program reached the end without errors or an error code if something goes wrong; let's state this time all will run without errors (we will add error control when the basic concepts will be clear). Then we define the data type containing a pid (05) and an integer working as counter for loops (06). These two types, as stated before, are identical, but they are here for clarity's sake.

At line (07) we call the fork function which will return zero to the program executed in the son process and the pid of the son process to the father; the test is at line (08). Now the code at lines (09)-(13) will be executed in the son process, while the rest (14)-(16) will be executed in the father.

The two parts simply write 8 times on the standard output the word "-SON-" or "+FATHER+", depending on which process executes it, and then ends up returning 0. This is really important, because without this last "return" the son process, once the loop has ended, would go further executing the father's code (try it, it does not harm your machine, simply it does not do what we want). Such an error will be really difficult to find, since the execution of a multitasking program (especially a complex one) gives different results at each execution, making debugging based on results simply impossible.

Executing the program you will perhaps be unsatisfied: I cannot assure you that the result will be a real mix between the two strings, and this due to the speed of execution of such a short loop. Probably your output will be a succession of "+FATHER+" strings followed by a "-SON-" one or the contrary. Try however to execute more than once the program and the result may change.

Inserting a random delay before every printf call, we may obtain a more visual multitasking effect: we do this with the sleep and the rand function.

```
sleep(rand()%4)
```

this makes the program sleep for a random number of seconds between 0 and 3 (% returns the remainder of the integer division). Now the code looks as

```
(09)   for (i = 0; i < 8; i++){
(->)     sleep (rand()%4);
(10)     printf("-FIGLIO-\n");
(11)   }
```

and the same for the father's code. Save it as fork_demo2.c, compile and execute it. It is slower now, but we notice a difference in the output order:

```
[leo@mobile ipc2]$ ./fork_demo2
-SON-
+FATHER+
+FATHER+
-SON-
-SON-
+FATHER+
+FATHER+
-SON-
-FIGLIO-
+FATHER+
+FATHER+
-SON-
```

```
-SON-
-SON-
+FATHER+
+FATHER+
[leo@mobile ipc2]$
```

Now let us look at the problems we have to face now: we can create a certain number of son processes given a father process, so that they execute operations different from those executed by the father process himself in a concurrent processing environment; often the father needs to communicate with sons or at least to synchronize with them, in order to execute operations at the right time. A first way to obtain such a synchronization between processes is the wait function

```
pid_t waitpid (pid_t PID, int *STATUS_PTR, int
OPTIONS)
```

where PID is the PID of the process whose end we are waiting for, STATUS_PTR a pointer to an integer which will contain the status of the son process (NULL if the information is not needed) and OPTIONS a set of options we have not to care about for now. This is an example of a program in which the father creates a son process and waits until it ends

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>

int main()
{
   pid_t pid;
   int i;

   pid = fork();

   if (pid == 0){
      for (i = 0; i < 14; i++){
         sleep (rand()%4);
         printf("-SON-\n");
      }
      return 0;
   }

   sleep (rand()%4);
   printf("+FATHER+ Waiting for son's
termination...\n");
   waitpid (pid, NULL, 0);
   printf("+FATHER+ ...ended\n");

   return 0;
}
```

The sleep function in the father's code has been inserted to differentiate executions. Let's save the code as fork_demo3.c, compile it and execute it. We just wrote our first multitasking synchronized application!

In the next article we'll learn more about synchronization and communication between processes; now write your programs using described functions and send me them so that I can use some of them to show good solutions or bad errors. Send me both the .c file with the commented code and a little text file with a description of the program, your name and your e-mail address. Good work!

Recommended readings

- Silberschatz, Galvin, Gagne, **Operating System Concepts - Sixth Edition**, Wiley&Sons, 2001
- Tanenbaum, WoodHull, **Operating Systems: Design and Implementation - Second Edition**, Prentice Hall, 2000
- Stallings, **Operating Systems - Fourth Edition**, Prentice Hall, 2002
- Bovet, Cesati, **Understanding the Linux Kernel**, O'Reilly, 2000

This article is re-printed with permission. The originals can be found at: http://www.linuxfocus.org/English/November2002/article272.shtml

Using the Logical Volume Manager

Vinayak Hegde vinayak@myrealbox.com

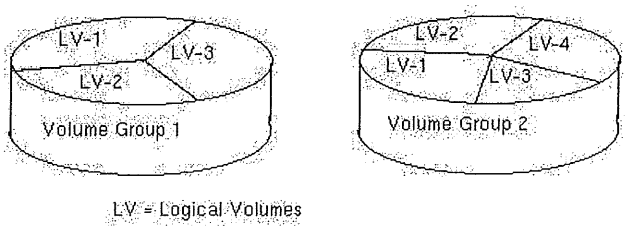
The Problem

One of the biggest problems faced by a linux user is the problem of estimating and allocating enough disk space to partitions when setting up a linux box. It does not matter much whether he is a system administrator looking after a server farm or an intermediate/power user of linux who has realized that he is going to run out of disk space. Sounds familiar doesn't it? Then starts the struggle to overcome the problem. Aha, the user has a brain-wave and problem is solved (after some sleepless nights) by using some non-elegant methods (read dirty hacks) like symlinks spanning partitions or using some partition resizing tools like parted. But these are only generally temporary solutions and we are faced with the same problem again.

How you wish that this problem could be solved!! The hacker in you wishes that you had a system on which you can experiment freely regardless of disk space and you could add or delete disk space as and when required. If you are a system administrator of a site with a number of servers which are always connected to the Internet, the stakes are all the more higher. Each minute of downtime causes losses. Even the danger of customers going away from your site. You can ill afford to reboot the server after you make changes to the partition table every time this scenario arises. LVM can be a lifesaver in such situations.

Introduction to LVM

The Linux LVM can make your life a little easier. LVM takes a higher level view of storage space as compared to that of partitions and hard disks. Read on to discover how. LVM was introduced into the main kernel source tree from 2.4.x series onwards. Before we move on to LVM, let us have a look at some of the concepts and terminology that will be used.



Physical Volume

Physical Volume generally refers to the hard disk partitions or a device that looks (logically) similar to a hard disk partition such as a RAID device.

Logical Volume

One or many physical volumes make up a **Logical Volume**. In LVM, a logical volume is similar to a hard disk partition in non-LVM systems. The logical volume can contain a file-system e.g. /home or /usr.

Volume Groups

One or many such logical volumes make up a **Volume Group**. For LVM, a volume group is similar to a a physical hard disk in a non-LVM system. The volume group brings together many logical volume to form one administrative unit.

How LVM works

Now that we have got a grip on the terminology of LVM, let us see how it actually works. Each physical volume is divided into a number of basic units called as **Physical Extents (PE)**. The size of a physical extent is variable but same for physical volumes belonging to a volume group. Within one physical volume, every PE has a unique number. The PE is the smallest unit that can be addressed by a LVM on a physical storage.

Again each logical volume is divided into a number of basic addressable units called as **Logical Extents (LE)**. In the same volume group the size of the logical extent is same as that of the physical extent. Obviously, the size of LEs is same for all the logical volumes of a volume group.

Each PE has a unique number on a physical volume but not necessarily for a logical volume. This is because a logical volume can be made up of several physical volumes in which case the uniqueness of PE IDs is not possible. Hence the LE IDs are used for identifying the LE as well as the particular PE associated with it. As has been noted earlier there is 1:1 mapping between the LEs and PEs. Every time the storage area is accessed the address or the IDs of the LE is used to actually write the data onto the physical storage.

You might be wondering by now, where all the meta-data about the logical volume and volume groups is stored. As a analogy, the data about the partitions is stored in the partition table in non-LVM systems. The **Volume Group Descriptor Area (VGDA)** functions similar to the partition table for LVM. It is stored at the beginning at the beginning of each physical volume.

The VGDA consists of the following information :-

1. one PV descriptor
2. one VG descriptor
3. the LV descriptors
4. several PE descriptors.

When the system boots the LVs and the VGs are activated and the VGDA is loaded into memory. The VGDA helps to identify where the LVs are actually stored. When the system wants to access the storage device, the mapping mechanism (constructed with the help of VGDA) is used to access the actual physical location to perform I/O operation.

Getting down to work

Let us now see how to use LVM :-

Step 1 -Configure the kernel

Before we begin to install LVM there are some prerequisites:- your kernel should have the LVM module configured.

This can be done as follows:-

```
# cd /usr/src/linux
# make menuconfig
```

under the Submenu:-

```
Multi-device Support (RAID and LVM) -->
```

enable the following two options:-

```
[*] Multiple devices driver support
(RAID and LVM)
<*> Logical volume manager (LVM)
Support.
```

Step2 - Check the Amount of Disk Space free on your drive

This can be done using the following command:-

```
# df -h
Filesystem  Size  Used  Avail  Use%  Mounted on
/dev/hda1   3.1G  2.7G  398M   87%   /
/dev/hda2   4.0G  3.2G  806M   80%   /home
/dev/hda5   2.1G  1.0G  1.1G   48%   /var
```

Step 3 - Create LVM partitions on your hard disk

Use fdisk or any other partition utility to create the LVM partitions. The partition type of linux LVM is 8e.

```
# fdisk /dev/hda
press  p (to print the partition table)
and    n (to create a new partition)
```

After the creation of the Linux LVM partition. Print the partition table. It will look something like this:-

```
Device  Boot Start End    Blocks   Id   System
```

```
/dev/hda1 *     1   506   4064413+  83  Linux
/dev/hda2     507   523    136552+   5  Extended
/dev/hda5     507   523    136521   82  Linux swap
/dev/hda6     524   778   2048256   8e  Linux LVM
/dev/hda7     779  1033   2048256   8e  Linux LVM
```

Step 4 - Create physical Volumes

```
# pvcreate /dev/hda6
pvcreate -- -physical volume "/dev/hda6"
successfully created

# pvcreate /dev/hda7
pvcreate- -- physical volume "/dev/hda7"
successfully created
```

The above command creates a volume group descriptor at the start of the partition.

Step 5 - Create Volume Groups

Create a new volume group and add the two physical volumes to it in the following way.

```
# vgcreate test_lvm /dev/hda6 /dev/hda7
vgcreate- -- INFO: using default physical extent
size 4 MB
vgcreate- -- INFO: maximum logical volume size is
255.99 Gigabyte
vgcreate- -- doing automatic backup of volume
group "test_lvm"
vgcreate- -- volume group "test_lvm" successfully
created and activated
```

This will create a volume group named test_lvm containing the physical volumes /dev/hda6 and /dev/hda7. We can also specify the extent size with this command if the extent size of 4MB is not suitable for our purpose.

Activate the volume groups using the command

```
# vgchange -ay test_lvm
```

The command "vgdisplay" is used to see the details regarding the volume groups created on your system.

```
# vgdisplay
--- Volume group ---
VG Name test_lvm
VG Access read/write
VG Status available/resizable
VG # 0
MAX LV 256
Cur LV 1
Open LV 0
MAX LV Size 255.99 GB
Max PV 256
Cur PV 2
Act PV 2
VG Size 3.91 GB
PE Size 4 MB
Total PE 1000
Alloc PE / Size 256 / 1 GB
Free PE / Size 744 / 2.91 GB
VG UUID T34zIt-HDPs-uo6r-cBDT-UjEq-EEPB-GF435E
```

Step 6 - Create Logical Volumes

The lvcreate command is used to create logical volumes in volume groups.

```
# lvcreate -L2G -nlogvol1 test_lvm
```

Step 7 - Create a file system

Now you need to build a filesystem on this logical volume. We have chosen to make the reiserfs journalling filesystem on the logical volume.

```
# mkreiserfs /dev/test_lvm/logvol1
```

Mount the newly created filesystem using the mount command.

```
# mount -t reiserfs /dev/test_lvm/logvol1 /mnt/lv1
```

Step 8 - Add entries to /etc/fstab and /etc/lilo.conf

Add the following entry to /etc/fstab so that the filesystem is mounted at boot.

```
/dev/test_lvm/logvol1 /mnt/lv1 reiserfs
defaults 1 1
```

Copy the recompiled kernel if you have not replaced your original kernel with it yet so you have the option of using LVM or not using it.

```
  image   = /boot/lvm_kernel_image
  label   = linux-lvm
  root    = /dev/hda1
  initrd  = /boot/init_image
  ramdisk = 8192
```

After adding the above lines reinstall lilo by using

```
# /sbin/lilo
```

Step 9 - Resizing logical volumes

Logical volumes can be resized easily using the lvextend command.

```
# lvextend -L+1G /dev/test_lvm/logvol1
lvextend -- extending logical volume
"/dev/test_lvm/logvol1" to 3GB
lvextend -- doing automatic backup of
volume group "test_lvm"
lvextend -- logical volume
"/dev/test_lvm/logvol1" successfully
extended
```

Similarly logical volumes can be reduced by using the following command

```
# lvreduce -L-1G /dev/test_lvm/lv1
lvreduce -- -Warning: reducing active logical
volume to 2GB
lvreduce- -- This may destroy your data (filesystem
etc.)
lvreduce -- -do you really want to reduce
"/dev/test_lvm/lv1"? [y/n]: y
lvreduce- -- doing automatic backup of volume group
"test_lvm"
lvreduce- -- logical volume "/dev/test_lvm/lv1"
successfully reduced
```

Conclusion

As we can see from the above discussion LVM is quite extensible and pretty straightforward to use. After the volume groups have been set up. It is pretty easy to resize logical volumes as per requirements.

Resources

The LVM Homepage
http://www.sistina.com/products_lvm.htm

The LVM HOWTO
http://www.tldp.org/HOWTO/LVM-HOWTO/index.html

This article is re-printed with permission. The originals can be found at:
http://www.linuxgazette.com/issue84/vinayak.html

Intrusion detection with Debian GNU/Linux

Author: José Salvador González Rivera <jsgr@tec.com.mx>
English Translation: **Georges Tarbouriech** <gt@linuxfocus.org>

Introduction

When selecting a Linux Operating Sytem, we must consider the numerous available distributions. Most of them are based on RedHat, for instance Conectiva (Brazil), Hispa source (Spain), Mandrake (France), SuSE (Germany), Caldera and many others using the RPM package manager. There is also Slackware, trying to be closer to traditional Unix only using .tgz archives. "Almost" all of them are developed by commercial companies, but this is not true for Debian. Debian provides a package manager (DPKG) helping us in updating since it automatically looks for updates from Internet; it also checks dependencies, thus making system administration easier and allows a system to be up-to-date as far as security patches are concerned.

Why Debian GNU/Linux ?

Debian also provides a few important features:
1) It does not have a commercial purpose and does not follow the dictates of market emergencies.
2) It does have a good bug tracking system, and problems are solved in less than 48 hours.
3) From the beginning its main priority is to develop a complete and reliable operating system.
4) It is developed by volunteers all around the world.

Every new version provides new hardware architecture support; at the moment, there is support for: Alpha, ARM, HP PA-RISC, Intel x86, Intel IA-64, Motorola 680x0, MIPS, MIPS (DEC), Power PC, IBM S/390, Sparc and they are working on Sun UltraSparc and Hitachi SuperH. It is the Linux system supporting the highest number of platforms. Among the existing Debian packages, there are various real time intrusion detection tools able to detect hostile behavior towards a connection. There are two types of tools: the ones monitoring a network attack attempt and the ones monitoring a specific host activity..

Host tools

We use PortSentry to detect portscans, TripWire to detect system changes and LogSentry for log analysis. The first one and the last one are part of the TriSentry suite by Psionic Technologies.

Portscan detection

PortSentry monitors the ports of our system and executes an action (usually blocking) if it detects a connection attempt to a port we do not want to be listened to.

Its home page is at
http://www.psionic.com/products/portsentry.html
and PortSentry is available for Solaris, BSD, AIX, SCO, Digital Unix, HP-UX, and Linux.

On Debian it can be installed typing the following instruction:

```
apt-get install portsentry
```

Different activity levels can be selected: the classic mode, the stealth mode and the advanced mode. The configuration relies on the /usr/local/psionic/portsentry/portsentry.conf file

I found the main options in an article from José Torres Luque in ES Linux Magazine and they are as follows:

```
TCP_PORTS, here you define the ports to be
controlled either in classic mode or in stealth
mode. The program's author provides three ports
lists according to the sensitivity level you want
to apply. The maximum number of ports is 64.

UDP_PORTS, is like the previous one but for UDP
ports.

ADVANCED_PORTS_TCP, ADVANCED_PORTS_UDP, indicate
the highest port number to use in advanced mode.
Every port under the one selected will be checked
except the ones already excluded. The highest can
be defined till the 65535. However, it is
recommended not to exceed 1024 to avoid false
alarms.

ADVANCED_EXCLUDE_TCP, ADVANCED_EXCLUDE_UDP, provide
a list of excluded ports. The ports found in this
section will not be monitored in advanced mode.
Here you write the connection ports usually
dedicated to remote clients and the ones not
providing a real service. For instance: ident

IGNORE_FILE, here we give the path of the file
where we write the IP addresses to be ignored at
monitoring time. The local interface, including lo,
should be found there too. You can also add the
local IP addresses.

KILL_ROUTE, here we can add the command to be
executed to block the attacker host. For instance:
iptables -I INPUT -s $TARGET$ -j DROP where
$TARGET$ refers to the attacker host.

KILL_RUN_CMD, we indicate a command to be executed
before blocking the access to the attacker host.

SCAN_TRIGGER, defines the number of attempts before
activating the alarm.

PORT_BANNER, displays a message on the open ports
in connect mode.

Once configured, it must be executed in one of the
three modes using the following options: for TCP
there is -tcp (basic mode), -stcp (stealth mode)
and -atcp (advanced mode); for UDP it can be -udp,
-sudp, -audp.
```

Integrity analysis

TripWire allows to check the file system integrity; the home page is at http://www.tripwire.org and it is freely available for Linux and commercial for Windows NT, Solaris, AIX and HP-UX. On Debian it can be installed typing the following instruction: apt-get install tripwire To store the information two keys are needed: the first one, the "site key" is used to cipher the policies and the configuration files, and the second one, the "local key" is used to cipher the information showing the monitored files status. The configuration is simply done in the /etc/tripwire/twpol.txt file and once it has been adapted, you can "install" it typing: twadmin -m P /etc/tripwire/twpol.txt To create the initial database containing the present status of the files, we execute the command: tripwire -m i 2 To check the integrity of the file system we type the instruction: tripwire -m c The configuration file can be deleted to prevent an intruder from knowing which files have been changed, using this command: rm /etc/tripwire/twcfg.txt /etc/tripwire/twpol.txt To create them if needed, type the following:

```
twadmin -m p > /etc/tripwire/twpol1.txt twadmin -m
f > /etc/tripwire/twcfg.txt
```

Logs analysis

LogCheck is part of LogSentry and allows logs analysis in a very efficient way since it classifies and makes reports about activity and errors that require reading. It provides 4 different logging levels: ignore, unusual activity, violation of security and attack. Its home page is at http://www.psionic.com/products/logsentry.html. It is available for Solaris, BSD, HP-UX and Linux. On Debian it can be installed typing the following instruction: apt-get install logcheck This installs the logtail program in /usr/local/bin to keep a list of the already analyzed logs. The following files are also installed:

```
Logcheck.sh
A script holding the basic configuration.

Logcheck.hacking
Holds the rules defining the activity levels.

Logcheck.ignore
Holds expressions to be ignored.

Logcheck.violations
Holds expressions that can be considered as
violation of security.

Logcheck.violations.ignore
The expressions found in this file are to be
ignored.
```

You can use cron to run logcheck every hour:

```
0 * * * * /bin/sh /usr/local/etc/logcheck.sh
```

Network tools

We use Snort to detect the network attack attempts. Its home page can be found at http://www.snort.org and it is available for BSD, Solaris, AIX, Irix, Windows, MacOS X and Linux. On Debian it can be installed typing the following instruction:

```
apt-get install snort
```

It works in three different modes: sniffer, packet logger and intrusion detector. It can use the following parameters:

```
-l directory
indicates the directory where to store the files.

-h IP defines the network IP address we want to
control.

-b captures every packet as binary.

-r file processes a binary file.]
```

Snort Sniffer and Packet Logger modes

In sniffer mode, it reads every packet circulating through the network and displays them on the console while in packet logger mode it sends the data to a file in a directory.

Snort -v
Shows IP and headers.

Snort -dv
Also shows the data circulating.

Snort -dev
A more detailed way.

Snort Intrusion Detection mode

In this mode, snort informs us about portscans, DoS (Denial of Service) attacks, exploits, etc. It relies on rules found in /usr/local/share/snort that you can download from the website and the server updates them about every hour. Its configuration is very simple since it consists in making changes to the snort.conf file, where we provide our network details and the working directories.
Just change the IP:
var HOME_NET IP

To execute snort, type:

```
snort -c snort.conf
```

The log files are stored in /var/log/snort where we can see the IPs of the attackers. This is of course a very short review of what you can do with snort and I recommend reading more about it. Most of the organizations, magazines, security groups consider this great tool as the best Intrusion Detection system for any Unix or Windows platform and recommend it. There is commercial support from companies such as Silicon Defense and Source Fire and GUIs are beginning to appear to provide a more attractive presentation of the results. Sometimes emergency situations appear requiring a deeper analysis since there are problems that have not been taken into account and that must be solved at once. These problems usually are caused by ill-intentioned people or intruders trying to access our servers for one reason or the other, either stealing or altering our data or attacking other machines from ours, either installing a sniffer or a rootkit which are sets of tools

allowing an intruder to gain more privileges on any system.

Other useful tools

Sniffer detection

A sniffer is a tool that changes our network interface to promiscuous mode with the goal of listening to the whole network traffic. The ifconfig command provides us with the full information about the network interface:

```
eth0 Link encap:Ethernet HWaddr 00:50:BF:1C:41:59
inet addr:10.45.202.145 Bcast:255.255.255.255
Mask:255.255.128.0
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:7180 errors:0 dropped:0 overruns:0
frame:0
TX packets:4774 errors:0 dropped:0 overruns:0
carrier:0 collisions:0 txqueuelen:100
RX bytes:8122437 (7.7 MiB) TX bytes:294607 (287.7
KiB)
Interrupt:10 Base address:0xc000
```

However, if the ifconfig command has been replaced or if the sniffer works from another machine in the network, you have to check the outside connections, for instance, sending mail to "strange" accounts or detecting the logs of the sniffer. There is a tool called neped, designed by a Spanish hacker group, which informs us about the interfaces working in promiscuous mode within our network. It is not part of Debian but it can be downloaded from ftp://apostols.org/AposTools/snapshots/neped/neped.c Note: this server seems to have been down for a few weeks.

Executing this program gives a result like the following:

```
neped eth0
-----------------------------------------------------
>My HW Addr: 00:80:F6:C2:0E:2A
>My IP Addr: 192.168.0.1
>My NETMASK: 255.255.255.0
> My BROADCAST: 192.168.1.255
-----------------------------------------------------
- Scanning ... * Host 192.168.0.2,
00:C2:0F:64:08:FF **** Promiscuous mode detected
!!!
End.
```

When sending an IP packet from 191.168.0.1 to 192.168.0.2 we need to know its MAC address. This is done sending a broadcast packet to the network asking for the MAC address of the specified IP: all the machines get the request but the right host is the only one answering. In this case neped asks every network IP, however it does not send a broadcast but uses a non-existent IP address instead. Only the hosts having their interface in promiscuous mode will answer since they are the only ones able to see these packets. I discovered this program in an article about spy detection found on the net. It was providing a similar example. If you know the URL for this article, feel free to send it to me by mail, since I lost it:-)

Rootkits detection

The rootkits provide a means of getting more privileges than a normal user can have. Generally, they replace our system binary files with different versions to gain a later access to the system. This is why we must check if we still have the original ones using chkrootkit. It can be installed like this:

```
apt-get install chkrootkit
```

The website is at www.chkrootkit.org and it checks the following files:

```
aliens, asp, bindshell, lkm, rexedcs, sniffer,
wted, z2, amd, basename, biff, chfn, chsh, cron,
date, du, dirname, echo, egrep, env, find, fingerd,
gpm, grep, hdparm, su, ifconfig, inetd, inetdconf,
identd, killall, ldsopreload, login, ls, lsof,
mail, mingetty, netstat, named, passwd, pidof,
pop2, pop3, ps, pstree, rpcinfo, rlogind, rshd,
slogin, sendmail, sshd, syslogd, tar, tcpd, top,
telnetd, timed, traceroute, w, write

To use it, type:
chkrootkit

It checks the files, looks for known sniffers and
rootkits. There are other tools to check log files
alteration (chkwtmp and chklastlog) and also
ifpromisc to tell us if our network interface is in
promiscuous mode.
```

References

Reading these programs man pages is recommended. I provide you with a few references I did use. Please, feel free to send me suggestions and comments to my email address.

- Alexander Reelsen, Securing Debian How To, version 1.4, 18 February 2001
- Anónimo, Linux Máxima Seguridad, Pearson Educación, Madrid 2000
- Brian Hatch, Hackers in Linux, Mc Graw Hill 2001
- Jim Mellander, A Stealthy Sniffer Detector, Network Security
- Antonio Villalón Huerta, Seguridad en Unix y redes, Open Publication License, octubre 2000
- CSI FBI Computer Crime and Security Survey, CSI Issues&Trends, Vol.7
- Who's Sniffing Your Network?, http://www.linuxsecurity.com/articles/intrusion_detection_article-798.html
- Root-kits and integrity: http://www.linuxfocus.org/November2002/article263.shtml

Shielded Processors: Guaranteeing Sub-millisecond Response in Standard Linux

Author: Steve Brosk <steve.brosky@ccur.com>
Author: Steve Rotolo <steve.rotolo@ccur.com>

Abstract

There has been significant progress making standard Linux into a more responsive system for real-time applications. The low latency patches and the preemption patches have allowed guarantees on worst case interrupt response time at slightly above a millisecond. These guarantees today are only met when there is no networking or graphics activity in the system. The shielded processor concept dedicates selected processors in a symmetric multiprocessing system for the real-time components of an application. This paper will describe the implementation of shielded processors in RedHawk Linux and the benefits of shielded processors. It will also present the results of benchmarks for both interrupt response and program execution determinism. Interrupt response time guarantees are significantly below one millisecond and can be guaranteed even in the presence of networking and graphics activity.

1. Introduction

Concurrent Computer has had more than a decade of experience in utilizing the shielded CPU model for attaining real-time performance under a real-time version of an SRV4 UNIX-based operating system. The key benefit of the shielded CPU approach is that it allows a commodity operating system to be used for applications that have hard real-time deadlines. Commodity operating systems like UNIX or Linux provide a benefit for these applications because they have large numbers of programmers that are familiar with the programming API and there is a rich set of development tools and other application software available for these operating systems.

Shielded CPUs can provide more deterministic performance because the overhead of the operating system is essentially off loaded onto a subset of CPUs in the system. A shielded CPU is therefore able to provide a more deterministic execution environment. In applying the shielded processor model to Linux several nuances were found which affected the expected behavior of processes running on shielded CPUs.

2. The Shielded CPU Model

The shielded CPU model is an approach for obtaining the best real-time performance in a symmetric multiprocessor (SMP) system. This approach does not apply to uniprocessor systems. The shielded CPU model allows for both deterministic execution of a real-time application as well as deterministic response to interrupts. A task has deterministic execution when the amount of time that it takes to execute a code segment within that task is predictable and constant. Likewise the response to an interrupt is deterministic when the amount of time it takes to respond to an interrupt is predictable and constant.

When the worst-case time measured for either executing a code segment or response to an interrupt is significantly different than the typical case, the application's performance is said to be experiencing *jitter*. Because of computer architecture features like memory caches and contention for shared resources, there will always be some amount of jitter in measurements of execution times. Real-time applications are defined by the fact that they must respond to real world events within a predetermined deadline. Computations that are completed after this deadline are considered incorrect. This means that the worst-case jitter that the operating system allows determines whether that operating system is suitable for hosting a given real-time application. Each real-time application must define the amount of jitter that is acceptable to that application.

In the shielded CPU model, tasks and interrupts are assigned to CPUs in such a way as to guarantee a high grade of service to certain important real-time functions. In particular, a high-priority task is bound to one or more shielded CPUs, while most interrupts and low priority tasks are bound to *other* CPUs. The CPUs responsible for running the high-priority tasks are shielded from the unpredictable processing associated with interrupts and the other activity of lower priority processes that enter the kernel via system calls, thus these CPUs are called *shielded* CPUs.

It will be shown that a shielded CPU can be used to guarantee deterministic execution and deterministic interrupt response times using a modified Linux kernel that presents a standard Linux API to the user. The benefit is that real-time applications can be developed using standard Linux interfaces and standard Linux debugging tools while still being able to guarantee very deterministic real-time performance.

3. Implementation of Shielded Processors

To create a shielded processor, it must be possible to set a CPU affinity for every process and every interrupt in the system. In this way a system administrator can define which processes and interrupts are allowed to execute on a shielded CPU. The Linux kernel already has support for CPU affinity in the form of a entry in the process structure for storing the CPU affinity and code in the scheduler that restricts processes to run only on CPUs that are a part of their CPU affinity. The only thing lacking in standard Linux is a user interface for setting a process' CPU affinity. Several open source patches provide this capability. Standard Linux does supports a CPU affinity for interrupts. In this case, the user interface is already present via the

/proc/irq//smp_affinity* files.

These two CPU affinity capabilities would allow a system administrator to setup a shielded processor, but it would require all processes and users in the system to honor the shielded processor by not explicitly changing their processor affinity to run on the shielded CPU. A better mechanism for setting up a shielded CPU is desirable.

In addition, there are some interrupts that cannot be assigned a CPU affinity. The local timer interrupt interrupts every CPU in the system, by default at a rate of 100 times per second or once every 10 milliseconds. This interrupt is generally the most active interrupt in the system and therefore it is the most likely interrupt to cause jitter to a real-time application. The local timer interrupt provides functionality such as the accounting of CPU execution time, system profiling and CPU resource limits. The shielded processor mechanism allows this interrupt to be disabled. Some of the functionality, such as CPU time accounting can be accomplished via other techniques. Other functionality that is more geared towards debugging and performance analysis, such as profiling, is simply lost when this interrupt is disabled.

A new set of /proc files were added to a new directory, */proc/shield*, to allow the system administrator to specify a bit mask of CPUs that should be shielded. It is possible to shield a CPU from both interrupts and processes. Separate files control shielding a CPU from processes, interrupts that can be assigned to a CPU and the local timer interrupt. It is possible to shield a CPU from all of these activities or just a subset.

Since we do want the ability to have some processes and some interrupts active on a shielded CPU, it was necessary to create a semantic for the interaction of process and interrupt affinity with the shielded CPU mask. In general, the CPUs that are shielded are removed from the CPU affinity of a process or interrupt.
The only processes or interrupts that are allowed to execute on a shielded CPU are processes or interrupts that would otherwise be precluded from running unless they were to allowed to run on the shielded CPU. In other words, to run on a shielded CPU, a process must set its CPU affinity such that it contains *only* shielded CPUs.

When one of the /proc files that controls CPU shielding is modified, the shielded CPU is dynamically enabled. This means that the affinity masks of all processes and interrupts are examined and modified accordingly. The processes currently assigned to the shielded processor, which will no longer be allowed to run on that processor, will be migrated to other CPUs the next time that they run. Because the affinity mask associated with interrupts is also modified; the shielded CPU will handle no new instances of an interrupt that should be shielded. The local timer interrupt is disabled on shielded CPUs that have been specified by the shield command. The ability to dynamically enable CPU shielding allows a developer to easily make modifications to system configurations when tuning system performance.

4. RedHawk kernel

Before describing the test scenarios that were used, it is necessary to describe the RedHawk Linux kernel, which was used for running benchmark tests that show the effect of shielded processors. The RedHawk kernel used was version 1.3. RedHawk is a Linux kernel based on kernel.org 2.4.20. Various open source patches have been applied to this kernel to augment both real-time functionality and real-time performance including the MontaVista preemption patch, Andrew Morton's low-latency patches and the Posix timers patches. Other changes have also been incorporated by Concurrent for improving real-time performance. This includes further low-latency work and the implementation of shielded processors. In addition, support was added for the Concurrent manufactured Real-time Clock and Interrupt Module (RCIM) PCI card. The RCIM provides the ability to connect external edge-triggered device interrupts to the system and also supports additional high-resolution timers. It will be shown how the RCIM driver is an important part of the RedHawk strategy for supporting deterministic interrupt response under Linux.

5. Determinism in Execution

Determinism refers to a computer system's ability to execute a particular code path in a fixed amount of time. The extent to which the execution time for the code path varies from one instance to another indicates the degree of determinism in the system. Determinism applies not only to the amount of time that is required to execute a time-critical portion of a user's application but also to the amount of time that is required to execute system service code in the kernel.

The standard Linux kernel has already addressed some of the primary causes of non-deterministic execution. For example Linux supports the ability to lock an application's pages in memory, preventing the jitter that would be caused when a program first accessed a page that is not resident in memory and turning a simple memory access into a page fault. Linux also supports strict priority-based scheduling so that the highest priority real-time processes are guaranteed to get as much CPU time as they require without having their priority eroded by scheduling fairness algorithms.

Previous experience with creating a real-time variant of UNIX showed that the primary remaining cause of indeterminism in program execution would be caused by interrupts. Because interrupts will preempt the execution of even the highest priority task, interrupts are essentially the highest priority activity in the system. An interrupt can occur at any point in time because it is asynchronous to the operation of the programs executing in the system. This means that interrupts can cause significant jitter to a real-time application because they cause delays in program

execution at unpredictable points in time.

5.1 Determinism Test

For this test, the system used was a dual processor 1.4GHz Pentium 4 Xeon with one GB of RAM and a SCSI hard drive.

Since we are measuring CPU execution determinism, it is desirable to have an application which is CPU bound for this measurement. The determinism test simply measures the length of time that it took to execute a function using double precision arithmetic to compute a sine wave. The sine function was called in a loop such that the total execution time of the outer loop should be around one second in length. Before starting this loop, the IA32 TSC register was read and at the end of the loop the TSC register is again read. The difference between these two high-resolution times represents that amount of time that it took to perform this CPU-bound loop. The test locks it pages into memory and the test process is scheduled under the SCHED_FIFO scheduling policy.

The base time for the determinism test was based on the ideal case for running the CPU-bound loop and was determined by running the test on an unloaded system. Both kernels under test were tried in an unloaded state. The best time was measured under RedHawk, on a shielded CPU.

Subsequently the test was run under various kernel configurations with a load on the system. Any run of the CPU-bound loop that took more time than the ideal case was considered to have been impacted by indeterminism in the system. The difference between the worst-case time it took to run the CPU bound loop and the ideal case represents the amount of jitter.

To measure worst-case jitter, a strenuous background workload must be run on the rest of the system. This workload should have a heavy interrupt load to show the worst-case performance. Two shell scripts were used to create Ethernet and disk interrupts. The first script was run on a foreign system and it copies a compressed kernel boot image over the Ethernet to the system that is being tested:

```
while true
do
    scp bzImage wahoo:/tmp
done
```

The second test generates disk traffic on the system by running a shell script that recursively concatenates files:

```
echo boo >9
while true
do
    for f in 0 1 2 3 4 5 6 7 8 9
        do
            cat * >$f
        done
done
```

5.2 Determinism Results

The determinism test was first run on a standard Linux kernel (kernel.org 2.4.20-rc1). The graph below graphs the amount variance from the ideal case in milliseconds. This means that a deterministic run would have a graph that has the majority of its data points on the left hand side of the graph. Also interesting is the worst-case time that it took to execute the computational loop. The results for the kernel.org kernel are summarized in figure 1. The results are also summarized in terms of minimum, maximum and the amount of jitter. The jitter reported is the difference between the maximum amount of time it took to run the computational loop and the ideal time it took to run the computational loop, expressed in both seconds and as a percentage of the ideal case.

```
<GRAPH> kernel.org 2.4.20-rc1
      ideal: 1.147132sec
        max: 1.447316sec
     jitter: 0.300184sec (26.17%)
determinism: 73.83%
```

Clearly there was significant variance in the amount of time that it took to run the computational loop on a standard Linux kernel when the system is busy with a load that causes interrupt traffic. In the worst case, the computational loop, which should have taken 1.15 seconds, took an additional 300 milliseconds to complete.

The test was next run on the RedHawk 1.3 kernel, on a shielded processor. Figure 2 graphs the amount variance from the ideal case with a summary of the results in the legend below the graph.

```
<GRAPH> RedHawk 1.3 (shielded CPU)
      ideal: 1.147132sec
        max: 1.168630sec
     jitter: 0.021498sec (1.87%)
determinism: 98.12%
```

As expected, a shielded processor provides a significant improvement in the amount of variance that we see from the ideal case. In the worst case, the computational loop, which should have taken 1.15 seconds, took an additional 21milliseconds to complete. This jitter is assumed to be due to memory contention in an SMP system.

To be sure that the improvement in determinism was due to shielding and not other differences in the system, the test was next run on the RedHawk 1.3 kernel, on a non-shielded processor. Figure 3 graphs the amount variance from the ideal case with a summary of the results in the legend below the graph.

```
<GRAPH> RedHawk 1.3 (unshielded CPU)
      ideal: 1.147132sec
        max: 1.317151sec
     jitter: 0.170019sec (14.82%)
determinism: 87.18%
```

The test confirmed that the interrupt load on an unshielded processor does indeed cause greater jitter in the execution time for executing a computational load.

However, the determinism on a non-shielded CPU was still significantly better than standard Linux. The mystery is why were the standard Linux results as

bad as they were? It was theorized that the cause was the fact that this version of Linux enables hyper-threading. A final version of the test was run on the standard Linux kernel with hyperthreading disabled via the GRUB prompt. Figure 4 graphs the amount variance from the ideal case with a summary of the results in the legend below the graph.

```
<GRAPH> kernel.org 2.4.20-rc1 (no hyperthreading)
      ideal: 1.147772sec
        max: 1.180178sec
     jitter: 0.032406sec (2.82%)
determinism: 97.18%
```

This test clearly identifies hyper-threading as the culprit for even greater non-deterministic execution. While hyper-threading does offer a performance boost for a multi-threaded application by enabling parallelism at the instruction unit level, this chip feature causes another layer of indeterminism for real-time applications. This is because with hyper-threading enabled, the execution unit itself has become a point of contention between the processes that are executing on the virtual processors of a single CPU.

6. Interrupt Response

Because real-time applications must respond to real world events and those events are communicated to the computer via an interrupt, determinism in responding to an interrupt is an especially important metric for a real-time operating system.

There are existing open source patches that address some of the issues in the standard Linux kernel for achieving good interrupt response. One such patch is the kernel preemption patch. This patch allows one process to preempt another process that is currently executing inside of the kernel. Prior to this patch when one process did a system call, no other process could execute inside of the kernel until that process either blocked or completed its system call. This has the potential to lead to very long delays when trying to wake a high priority process that was awaiting an interrupt when there is currently a non-preemptible task executing in the kernel.

Even with the preemptible kernel patch there are remaining issues with preempting a process that is executing inside of the kernel. When a process makes a system call, that process might enter into one of the Linux kernel's critical sections. A critical section is an area of code that accesses a shared kernel data structure. Because the data structure is shared, it might be simultaneously accessed by another process that is executing inside of the kernel. To prevent the shared data from being corrupted, a critical section requires synchronization primitives that allow only one process at a time to access the shared data. The preemptible kernel patch does not allow a process to be preempted while it is inside of a critical section, since the preempting process might then also try to access the shared data of the pending critical section, causing the shared data to be corrupted.

Because the kernel's critical sections cannot be preempted, the length of the critical sections inside of the kernel is significant when considering worst-case interrupt response. Other open source patches collectively known as the "low-latency patches," address the longest critical sections in the kernel by rewriting the algorithms involved so that preemption can be disabled for a shorter period of time. The combination of the preemption patch and the low-latency patch sets was used on a Red Hat based system to demonstrate a worst-case interrupt response time of 1.2 milliseconds [1].

Experience with working with a real-time variant of UNIX showed that when trying to guarantee how long it will take to respond to an interrupt, the biggest problem is the critical sections that disable preemption. Consider the case where a low priority process enters the kernel to process a system call and that process enters a critical section where preemption is disabled. If a high priority interrupt becomes active at this point in time, the system will process that interrupt, but when the interrupt routine wakes the process that is awaiting the interrupt, that process will not be able to run until the execution of the critical section is complete. This means that the worst-case time to respond to an interrupt is going to be at least as long as the worst-case time that preemption is disabled in the kernel.

In a Symmetric Multiprocessor system that supports CPU shielding it is possible to prevent low priority processes from running on a CPU where a very fast response to interrupt is required. While this means that some CPU resources will be under utilized, it does allow a very firm guarantee for processes that require a high degree of interrupt response.

6.1 Interrupt Response Test

To measure interrupt response time, the realfeel benchmark from Andrew Morton's website was initially used. This test was chosen because it would allow results to be compared between a standard Linux kernel and a RedHawk system. This test operates by measuring the response to an interrupt generated by the Real Time Clock (RTC) driver. This driver is setup to generate periodic interrupts at a rate of 2048 Hz. The RTC driver supports a read system call, which returns to the user when the next interrupt has fired. The clock used to measure interrupt response is the IA32 TSC timer. To measure interrupt response time, the test first reads the value of the TSC and then loops doing reads of /dev/rtc. After each read the test gets the current value of the TSC. The difference between two consecutive TSC values measures the duration that the process was blocked waiting for an RTC interrupt. The expected duration is 1/2048 of a second. Any time beyond the expected duration is considered latency in responding to an interrupt. The test locks it pages into memory and the test process is scheduled under the SCHED_FIFO scheduling policy.

To measure worst-case interrupt response time, a strenuous background workload must be run on the rest of the system. The workload chosen was the same as that used in Clark William's paper on Linux Scheduler Latency [1]. This workload is from the Red

Hat stesss-kernel RPM. The following programs from stress-kernel are used:

```
NFS-COMPILE
TTCP
FIFOS_MMAP
P3_FPU
FS
CRASHME
```

The NFS-COMPILE script is the repeated compilation of a Linux kernel, via an NFS file system exported over the loopback device. The TTCP program sends and receives large data sets via the loopback device. FIFOS_MMAP is a combination test that alternates between sending data between two processes via a FIFO and operations on an mmap'd file. The P3_FPU test does operations on floating point matrices. The FS test performs all sorts of unnatural acts on a set of files, such as creating large files with holes in the middle, then truncating and extending those files. Finally the CRASHME test generates buffers of random data, then jumps to that data and tries to execute it. Note that while no Ethernet activity was generated on the system, the system did remain connected to a network and was handling standard broadcast traffic during the test runs.

6.2 Interrupt Response Results

The system used for the test was a dual 933MHz Pentium 3 Xeon with 2GB of RAM with a SCSI disk drive. Two different kernels were measured under the same load conditions.

The first kernel used was a standard Linux (kernel.org 2.4.20-rc1). Note that this kernel does not contain the low-latency patches or the preemption patch. After starting the stress-kernel program, realfeel was run for 60,000,000 samples at 2048 Hz. The test was terminated before the full eight-hour run completed because we already had enough data showing poor interrupt latency on this kernel. Figure 5 graphs the interrupt response for a standard Linux kernel. Note that the y axis is a logarithmic scale. This graph is summarized in terms of histogram buckets below the graph.

```
<GRAPH for kernel.org interrupt response test>

measured 44759417 rtc interrupts
max latency: 92.3ms
        average latency: .043ms

44374681 samples < 0.1ms (99.140%)
44594353 samples < 0.2ms (99.631%)
44687849 samples < 1.0ms (99.843%)
44702467 samples < 2.0ms (99.872%)
44719462 samples < 5.0ms (99.910%)
44732301 samples < 10.0ms (99.939%)
44742797 samples < 20.0ms (99.962%)
44748489 samples < 30.0ms (99.975%)
44753080 samples < 40.0ms (99.985%)
44756536 samples < 50.0ms (99.993%)
44759250 samples < 60.0ms (99.999%)
44759363 samples < 70.0ms (99.999%)
44759402 samples < 80.0ms (99.999%)
44759416 samples < 90.0ms (99.999%)
44759417 samples < 100.0ms (100%)
```

While the majority of the responses to interrupt to occur in less than 100 microseconds, for a real-time application the most important metric is the worst-case interrupt response. This graph shows that standard Linux without the patches that implement minimal real-time performance gains has very poor guarantees on interrupt response. At 92 milliseconds, the worst-case interrupt response is completely unacceptable for a real-time application. These results are expected.

The second kernel tested was the RedHawk 1.3 kernel described above. After starting the stress-kernel program, realfeel was run for 60,000,000 samples at 2048 Hz. This run was approximately 8 hours in length. While the length of this run may seem like overkill, early results showed us that on a shielded CPU the worst-case numbers might not occur until several hours into the run.

In this test, CPU 1 was setup as a shielded processor. The RTC interrupt and realfeel have their CPU affinity set such that they run on shielded CPU 1. The stress-kernel test is run without any CPU affinity set. The results of the interrupt response test for a RedHawk shielded processor are presented in Figure 6. Again, the results are also summarized in histogram form below the graph.

```
<GRAPH for RedHawk interrupt response test>

measured 60000000 rtc interrupts
max latency: 0.565ms
average latency: .18ms

59999983 samples < 0.1ms (99.99997%)
       8 samples < 0.2ms
       5 samples < 0.3ms
       2 samples < 0.4ms
       1 samples < 0.5ms
       1 samples < 0.6ms
```

The initial tests run under RedHawk on shielded CPUs showed worse results than expected. The problems discovered resulted in several additional fixes to the Linux kernel to allow us to achieve a more optimal interrupt response on a shielded processor. The primary problem was due to critical sections that are protected by spin locks that do not disable interrupts. It is not necessary for these spin locks to disable interrupts because the critical section is never locked at interrupt level. When interrupts are not disabled, it is possible for an interrupt routine to preempt a critical section being protected by a spin lock. Because interrupt routines are relatively short, this should not be a big issue. The problem was in the bottom half interrupt routines that would run on return from interrupt. These interrupt bottom halves sometimes executed for several milliseconds of time. If the process used to measure interrupt response on the shielded processor attempts to lock the contended spin lock (which had been preempted by interrupts and bottom half activity) during the read of dev/rtc then the response to interrupt could be delayed by several milliseconds.

Because the /dev/rtc mechanism works via the read() system call, a process that wakes up after the interrupt fires must now exit the kernel through various layers of generic file system code. Embedded in this code are opportunities to get blocked waiting for spin locks. The /dev/rtc interface is therefore not ideal for guaranteeing the time to respond to an

interrupt.

6.4 A Second Interrupt Response Test

While the initial experiment did succeed in reducing interrupt latency below one millisecond, the results were not as good as expected for shielded CPUs. It was theorized that these mediocre results were due to the the fact that the realfeel test uses /dev/rtc, whose API is considered less than optimal, as described above. Therefore a new interrupt response test was designed. In this test the real-time timer on the Real-time Clock and Interrupt Module (RCIM) PCI card was utilized as the interrupt source.

To block waiting for the RCIM's timer to interrupt, the user makes an ioctl() call rather than a read() system call. In addition, the Linux kernel got a modification to correct one of the issues found with this interrupt response test. Linux locks the BKL spin lock before entering a device driver's ioctl routine. This is to protect legacy drivers which are not properly multithreaded from having issues on an SMP system. The problem is that the BKL spin lock is one of the most highly contended spin locks in Linux and attempting to lock it can cause several milliseconds of jitter.

A change was implemented to the generic ioctl support code in Linux so that it would check a device driver specific flag to see whether the device driver required the Big Kernel Lock (BKL) to be held during the driver's ioctl routine. This allows a device driver that is fully multi-threaded to avoid the overhead of the BKL. Since the RCIM driver is multi-threaded, it does not require the BKL to be locked during its ioctl routine.

Like realfeel, the RCIM interrupt response test measures the amount of time it takes to respond to an interrupt generated by a high-resolution timer. When the RCIM is programmed to generate a periodic interrupt, the length of the periodic cycle is stored in the RCIM's count register. The count register is decremented until it reaches zero, at which time an interrupt is generated. When the count reaches zero, the RCIM will also automatically reset the count register to its initial value and begin decrementing the count register for expiration of the next periodic cycle.

The RCIM interrupt response test operates by initiating a periodic interrupt on the RCIM and then, in a loop, issuing the ioctl to block until an interrupt is received. When the test is waked, it immediately reads the value of the count register on the RCIM. Because this register can be directly mapped into the program, the overhead of this read is only about 3 microseconds. The test can then calculate the time since the interrupt fired by subtracting the current value of the counter register from the initial value that is loaded into the count register at the beginning of each periodic cycle. The test locks it pages into memory and the test process is scheduled under the SCHED_FIFO scheduling policy.

In this test scenario the workload was significantly increased from that used during the realfeel benchmarking above. The same stress-kernel load was used, but in addition, the X11perf benchmark was run on the graphics console and the ttcp network performance benchmark was run, reading and writing data across a 10BaseT Ethernet connection.

The test was run on a dual processor 2.0 GHz Pentium 4 Xeon with 1GB of RAM and SCSI disks. The Ethernet controller is the 3Com 3c905C-TX/TX-M. The graphics controller is the nVidia GeForce2 MXR.

Because this interrupt response test requires the RCIM driver, which is not a part of standard Linux, no numbers were gathered for a standard Linux kernel. The results for running this test on RedHawk 1.3 are shown in figure 7. Note that the numbers in this thin bar histogram represent microseconds, NOT milliseconds. The y axis is a logarithmic scale.

```
<GRAPH for RCIM interrupt response test>

measured 28800882 RCIM interrupts
minimum latency: 11 microseconds
maximum latency: 27 microseconds
average latency: 11.3 microseconds

28800870 samples < 0.02ms (99.99999%)
12 samples < 0.03ms
```

This test demonstrates that the issues seen with the realfeel test have to do with the multithreading issues of /dev/rtc. When the RCIM driver is used to generate a high-resoultion timer interrupt, a shielded processor is able to provide an absolute guarantee on worst-case interrupt response time that is less than 30 microseconds.

7. Conclusion

It has been demonstrated that processes executing on a shielded processors on a standard Linux kernel can achieve respectable determinism in the time that it takes to execute a user-level application. Enabling hyperthreading on the Xeon chip causes another level of contention between the processes that are executing on the virtual CPUs provided by hyperthreading and causes a decrease in program execution determinism.

It has also been demonstrated that when an interrupt and the program that responds to that interrupt are run on a shielded processor it is possible to guarantee interrupt response that is less than 30 microseconds. This guarantee can be made even in the presence of heavy networking and graphics activity. This interrupt response guarantee rivals the guarantees that can be made by much smaller and much less complex real-time kernels. There are remaining multithreading issues to be solved in the Linux kernel to achieve this level of interrupt response for other interrupt mechanisms.

References

[1] Clark Williams, 2002, Linux Scheduler Latency, Red Hat web cast.

Steve Brosky and Steve Rotolo
Concurrent Computer Corporation
2881 Gateway Drive, Pompano Beach, FL 33069
{steve.brosky, steve.rotolo}@ccur.com

This article is re-printed with permission of Concurrent Computer Corp.

Fighting against Spam-Mail

Authors: Katja and Guido Socher <katja@linuxfocus.org>, <guido@linuxfocus.org>

What is spam-mail?

Spam-mail has many names. Some call it UCE (Unsolicited commercial email) others call it just Unwanted E-mail but all these names don't really say what it is. If you don't get spam (yet) then take a look at this collection of spam-mail (http://www.linuxfocus.org/common/src/article279/spam_samples.html). It's a random selection of spam-mail collected over just a few days. Read through the mails and you will soon understand that it has nothing to do with commerce or business. These spammers are criminals. No serious business man/woman would annoy and offend millions of people to find a few "idiots" who would buy their tricks.

It is a common misunderstanding of people who have not much used the Internet to believe that this type of advertisement can be compared to information they get from time to time from their local supermarket. Products sold via spam-mails are often illegal or no products at all. They are tricks to get your money.

How Much?

Spammers get your e-mail addresses from webpages, news groups or domain records (if you have your own domain). There are individuals who use robots to extract the addresses, burn them on CDs and sell them very cheap to other Spammers. If you write your e-mail address in clear text onto your homepage today such that programs can extract it, then you will have a major problem in a few months time and you can't stop it. The problem will be growing every day!

In 1998 the percentage of spam mail sent to LinuxFocus was less than 10%. As of November 2002 the statistics are as follows:
Our server gets about 4075 mails per week. 3273 are spam-mails! => **80% of all mail is Spam.**

That is 80% of the capacity of the mail server and 80% of the network bandwidth is for something that nobody wants.

Out of these 3273 spam mails about 40% originate in America (mostly Canada, US, Mexico) and about 30% in Asia (mostly Korea, China, Taiwan).

What to do with Spam

If you look at the spam-mails (http://www.linuxfocus.org/common/src/article279/spam_samples.html) you will notice that almost all offer a possibility to be removed from the list. Don't do it! You are dealing with criminals. None of the spammers get anything if they maintain a proper remove list. Why do they still add this possibility? The answer is simple. It makes a much better impression on the reader and it's an excellent statistical tool. The spammers can immediately check that their mails arrive. In other words **you confirm the reception of the mail!**

There is also a simple technical problem with the idea of a remove list. LinuxFocus is not a very big site but we would need 1 person full time to unsubscribe 3273 Spam mails per week and then this person would need to unsubscribe one mail every minute . Every spammer uses a different method, it would be an idiotic task and it can't work. Remove lists are nonsense and help only the spammers.

The only right thing to do is: delete it.

Software to handle spam

There are many different options to filter out spam and this is good because it makes it harder for spammers to circumvent them. It's however an arms race. The tools to filter spam become more sophisticated but spammers improve their methods too.

There are 2 types of filters:

1. Checks directly build into the MTA (Message Transfer Agent=Mail server). Here you can usually reject the mail. That is: you don't even store the email. You send an error code back as soon as you recognize that this is spam during the reception of the email. Typical tools of this kind are IP based blocklists and mail header checks. If you don't have your own Mailserver then your ISP would need to configure this.
2. Filtering after the reception of the mail. In this case the email is successfully delivered and will be filtered out later.

We will now discuss the different possibilities in detail, all of them have advantages and disadvantages. The best solution to get rid of all spam is to use several different tools.

Rejecting email directly at the MTA

If you reject your mail directly at the mail server during the reception of the mail then the spammer can get back an error code and knows that this address does not work. If he is one of the "CD-makers" then he might take out the address. It can save network bandwidth because you don't have to receive the full message. You can send the error code back as soon as you find that this is spam.

To do this you need a good and flexible MTA. Unfortunately the two most common servers, Sendmail and the one from Bill Gates are not good at all for this task. Two very good alternatives are Postfix http://www.postfix.org and Exim http://www.exim.org. If you can't change your server then you can put an smtp proxy such as messagewall in front of the server (smtp = Simple Mail Transfer Protocol, the Internet mail protocol).

We will now discuss some common filter techniques and how they work. We will not describe how to configure them exactly in each MTA. It would make the article too long. Instead we suggest to read the documentation that comes with the MTA that you have installed. Postfix and Exim are well documented.

- Realtime Block lists:
  These are DNS based lists. You check the IP address of the mailserver that wants to send mail to your server against a blacklist of known spammers. Common lists are www.spamhaus.org or ordb.org. There is also a tool called blq (see references) to manually query such block lists and test if a given IP address is listed. You should however not be too enthusiastic about it and carefully choose the lists since there are also some which block entire IP ranges simply because one spammer had used a dialup connection from this ISP at one point in time. We personally would at least enable ordb.org to keep out mail from poorly administrated servers.
  Experience shows that these lists block about 1%-3% of the spam mail.
- 8 bit characters in subject line:
  About 30% of the spam origins in China, Taiwan or other Asian countries these days. If you are sure that you can't read Chinese then you can reject mail which has a lot of 8 bit characters (not ASCII) in the subject. Some MTAs have a separate configuration option for this but you can also use regular expression matching on the header:

  ```
  /^Subject:.*[^ -~][^ -~][^ -~][^ -~]/
  ```

  This will reject email which has more than 4 consecutive characters in the subject line which are not in the ASCII range space to tilde. If you are not familiar with regular expressions then learn them, you will need them (See LinuxFocus article 53 http://www.linuxfocus.org/English/July1998/article53.html). Both exim and postfix can be compiled with perl regular expression support (see www.pcre.org). Perl has the most powerful regular expressions.
  This method is quite good and keeps out 20-30% of the spam-mail.
- Lists with "From" addresses of known spammers:
  Forget it. This used to work back in 1997. Spammers today use faked addresses or addresses of innocent people.
- Reject non FQDN (Fully Qualified Domain Name) sender and unknown sender domain:
  Some spammers use non existent addresses in the "From". It is not possible to check the complete address but you can check the hostname/domain part of it by querying a DNS server.
  This keeps out about 10-15% of the spam and you don't want these mails anyhow because you would not be able to reply to them even if they were not spam.
- IP address has no PTR record in the DNS:
  This checks that the IP address from where you get the mail can be reverse resolved into a domain name. This is a very powerful option and keeps out a lot of mail. We would not recommend it! This does not test if the system administrator of the mail server is good but if he has a good backbone provider. ISPs buy IP addresses from their backbone providers and they buy from bigger backbone providers. All involved backbone providers and ISPs have to configure their DNS correctly to make the whole chain work. If somebody in between makes a mistake or does not want to configure it then it does not work. It says nothing about the individual mail server at the end of the chain.
- Require HELO command:
  When 2 MTAs (mail servers) talk to each other (via smtp) then they first say who they are (e.g. mail.linuxfocus.org). Some spam software does not do that. This keeps out 1-5% of the spam.
- Require HELO command and reject unknown servers:
  You take the name that you get in the HELO command and then you go to DNS and check if this is a correctly registered server. This is very good because a spammer who uses just a temporary dialup connection will usually not configure a valid DNS record for it.
  This blocks about 70-80% of all spam but rejects also legitimate mail which comes from sites with multiple mail servers where a sloppy system administrator forgot to put the hostnames of all servers into DNS.

Some MTAs have even more options but the above are quite commonly available in a good MTA. The advantage of all those checks is that they are not CPU intensive. You will usually not need to update your mailserver hardware if you use those checks.

Filtering of already received mail

The following techniques are usually applied to the complete mail and the mail server who sends the mail does not notice that the mail could not be delivered. It means also that a legitimate sender will not get a failure report. The message will just disappear.
Having said this we must also say that this is not totally correct because it really depends on the filtering possibilities of the mail server. Exim is very flexible and would allow you to write custom filters on messages.

- SpamAssassin (http://spamassassin.org/):
  This is a spam filter written in perl. It uses carefully handwritten rules and assigns certain points to typical spam phrases such as "strong buy", "you receive this mail

because", "Viagra", "limited time offer".... If the points are above a given level then the mail is declared as spam. The problem with this filter is that it is very heavy in terms of memory and cpu power. You will probably need to upgrade your mail server hardware especially if the server is already 2-3 years old. We would not recommend to use it directly on the mail server. Spamassassin comes with a spamd program (spamd=spam daemon + spamc=client to connect to the daemon) which will reduce the startup time of spamassassin and reduce the cpu consumption but it is still a very resource demanding application.

To filter the mail you need to create a .procmailrc file (and .forward) similar to this one:

```
# The condition line ensures that only
# messages smaller than 50 kB
# (50 * 1024 = 56000 bytes) are processed
# by SpamAssassin. Most spam
# isn't bigger than a few k and working
# with big messages can bring
# SpamAssassin to its knees. If you want
# to run SpamAssassin without
# the spamc/spamd programs then replace
# spamc by spamassassin.
:0fw:
* < 56000
| /usr/bin/spamc
# All mail tagged as spam (e.g. with a
# score higher than the set threshold)
# is moved to the file "spam-mail"
# (replace with /dev/null to discard all
# spam mail).
:0:
* ^X-Spam-Status: Yes
spam-mail
```

The installation is easy and spamassassin will filter more than 90% of the spam.

- procmail (http://www.procmail.org):
  Procmail is not a spam filter on its own but you can use it to write yourself one. procmail is also very light weight as long as you limit the number of rules to something reasonable (e.g. less than 10). To use it you create a .forward file in your home-directory and add there the following line:

```
"| exec /usr/bin/procmail"
```

Some people recommend to use

```
"|IFS=' ' && exec /usr/bin/procmail"
```

but this creates new problems with an extra process being created which does not run under the control of the mailserver any longer. Secure mail servers like postfix or exim will have no problems with the .forward file as shown above.

Procmail is especially useful in an environment where you normally communicate just in a closed group. E.g. for people in a company where most of the mail should come from your colleagues and some known friends. Here is an example for "mycompany.com":

```
# .procmailrc file.
# search on header for friends:
:0 H:
* ^From.*(joe|paul|dina)
/var/spool/mail/guido

# search on header for mails which are
# not coming from
# inside mycompany.com and save them
# to maybespam
:0 H:
* !^From.*(@[^\@]*mycompany\.com)
/home/guido/maybespam

# explicit default rule
:0:
/var/spool/mail/guido
```

This makes it much easier to delete spam and you don't find the ugly spam between your normal mail.

Procmail is very flexible and can also be used for other tasks. Here is a totally different example:
Procmail comes with a "reply to sender" program called formail. This can e.g. be used to send a message back to people. A major plague are those e-mails with word documents inside. If you are a Linux developer using e-mail to exchange information about your projects or Linux in general then you are for sure not interested in people who write text into a word document and attach it to mails. Viruses can easily be spread that way. They don't usually infect Linux but it's a bad idea in general to use MS-word for sending text to other people as it requires MS word with the same version on the receiver side to read the text. There are open formats such as RTF or HTML which do not spread viruses, are cross platform, and do not have such a version problem.

-

```
# Procmail script to
# reject word documents. Reject the mail,
# but do not reply to
# error messages "From MAILER-DAEMON"
# If you use ":0 Bc" instead of ":0 B"
# then you will still get the mail
:0 H
* !^From.*DAEMON
{
  # The mime messages with word documents
  # look like this in the body
  # of the message:
  #------\
=_NextPart_000_000C_01C291BE.83569AE0
  #Content-Type: application/msword;
  #         name="some file.doc"
  #Content-Transfer-Encoding: base64
  #Content-Disposition: attachment;
  #         filename="real file.doc"
  :0 B
  * ^Content-Type:.*msword
  | (formail -r ; cat /home/guido/reject-
text-msword ) | $SENDMAIL -t
}

# explicit default rule
:0:
/var/spool/mail/guido
```

The text file /home/guido/reject-text-msword should contain a text explaining that msword documents can spread viruses and ask the sender to send the document e.g. in RTF format.

How to use procmail and what all these strange letters in the configuration file mean is very well explained in the "procmailrc" man page.

- bogofilter (http://www.tuxedo.org/~esr/bogofilter): Bogofilter is a Bayesian spam filter. It is entirely written in C and it is very fast (compared to SpamAssassin). A Bayesian filter is a statistical filter that you have to train first to learn what is spam and what is not spam. You need about 100 training messages (sorted into spam and not spam) until the filter can work efficiently on new messages.

  Bogofilter is fast but it does not work from day one as SpamAssassin. After a while it will be as efficient as SpamAssassin and filter more than 90% of the spam.
- razor (http://razor.sf.net/): This is a distributed, collaborative, spam detection system. Checksums of known spam messages are stored in a database. If you get a new mail you compute the checksum and compare it with checksums in the central database. If the checksum matches then you can discard the message as spam. razor works because special e-mail accounts have been spread over the Internet only for the purpose of getting into the address lists of all the spammers. These accounts catch only spam and no normal mail. In addition people can of course send mails to razor for marking it as spam. There is a good chance that the mails are already known as spam before they arrive in your mailbox. The system filters about 80% of the spam. razor has one characteristic that none of the other post processing and filtering techniques has: razor detects almost no false positives. That is: the number of mails which are not spam but still declared spam is very low with razor.

There are many more possible solutions to fight against spam. We believe that the above covers the most important ones.

The best solution is to use checks in the MTA as a first stage and then kill the remaining spam in a second stage with a post-processing filter.

HTML MAILS

A particularly dangerous form of e-mail are spam mails in HTML format.

Most spammers use the "unsubscribe possibility" to see how many of their mails arrive. HTML formatted mail offers a much better form of feedback: Images. You can compare this system with the visitor counters as found on some webpages. The spammer can exactly see when and how many of the mails are read. If you study Spam carefully you will see that in some cases the URL for included images contains a sequence number: The spammer can see exactly who looks at the mail and at what time time. An incredible security hole

Modern mail reader programs will not display images which are downloaded somewhere from a URL. However there is hardly any modern and secure HTML mail reader. Kmail and the very latest version of mozilla mail offer the possibility to disable images from external sources. Most other programs will generate nice statistics for the spammer.

The solution? Don't use a html mail capable program or download the mail first then disconnect from the Internet and then read the mail.

WHERE DOES THE SPAM COME FROM?

Never trust the sender address in the "From" field of spam mails! These are either non existent users or innocent people. It is very rare that this is the mail address of the spammer. If you want to know where the mail comes from then you have to look at the full header:

```
...
Received: from msn.com (dsl-200-67-219-
28.prodigy.net.mx [200.67.219.28])
        by mailserver.of.your.isp (8.12.1) with
                SMTP id gB2BYuYs006793;
        Mon, 2 Dec 2002 12:35:06 +0100 (MET)
Received: from unknown (HELO rly-xl05.dohuya.com)
                (120.210.149.87)
        by symail.kustanai.co.kr with QMQP; Mon,
                02 Dec 2002 04:34:43
```

Here an unknown host with IP address 120.210.149.87 who claims to be rly-xl05.dohuya.com sends the mail to symail.kustanai.co.kr. symail.kustanai.co.kr sends this message further on.
The spammer is hiding somewhere behind 120.210.149.87 which is probably just a dynamic dialup IP address.

In other words the police could find this person if they would go to the owner of kustanai.co.kr and ask for server logs and a printout of connections from the local telephone company. You have very little chance of finding out who that was.

It could also be that the first part is faked and the spammer is really behind dsl-200-67-219-28.prodigy.net.mx. This is very likely since there is no good reason why symail.kustanai.co.kr should send the mail to msn.com via the dsl dialup connection (dsl-200-67-219-28.prodigy.net.mx). The mailserver.of.your.isp (symbolic name) is the server of your Internet Service Provider and is the only part from this "Received:" line which is reliable.

It is possible to find the spammer but you need international intelligence and police forces to go to prodigy.net.mx.

CONCLUSION

If spam continues to increase at the current rate then the Internet will soon transport a lot more Spam than real e-mail. Spam is transported at the cost of the receiver. More bandwidth is needed and often the mail

systems need to be upgraded to handle the Spam. Laws in many countries do little to protect people against criminal spammers. In fact some countries have laws which restrict only honest people (digital rights management etc. ...) and help the criminals (e.g. to get nice statistics about the spam-mail).

Join the Coalition Against UCE!

http:// http://www.euro.cauce.org/en/

http://www.cauce.org/

Internet Service Providers should check their mail systems. No unauthenticated access to mail servers must be given and the amount of mails that one user can send per minute must be limited.

References

- http://spamassassin.org/: spamassassin homepage
- http://www.procmail.org/: procmail homepage
- http://www.spambouncer.org/: spambouncer: a procmail based spam filter
- http://www.postfix.org/: homepage of the postfix MTA
- http://www.exim.org/: homepage of the exim MTA
- http://messagewall.org/: homepage of the messagewall smtp proxy
- http://www.unicom.com/sw/blq/: the blq perl script to query DNS based block lists
- http://www.ordb.org/: DNS based open relay block list
- http://www.spamhaus.org: DNS based block list
- http://www.samspade.org/: Where does the spam come from?
- http://www.dnsstuff.com/: various blocklists and DNS based tools
- http://www.geektools.com/cgi-bin/proxy.cgi: geektools Whois proxy
- http://www.tuxedo.org/~esr/bogofilter/: bogofilter mail filter
- http://razor.sf.net/: razor
- http://pyzor.sourceforge.net: razor implemented in python
- http://lwn.net/Articles/9460/: Linux weekly news article comparing bogofilter and spamassassin.

This article is re-printed with permission. The originals can be found at:

http://www.linuxfocus.org/English/January2003/article279.shtml

Making a Multiple-Boot CD

Author: Juraj Sipos<xvudpapc@savba.sk>

Introduction

I noticed that the issue of making a multiboot CD is not very much covered on the Internet, and if so, only sparsely. Commercial Windows vendors include some possibility to create bootable CD's in their software, but I haven't yet seen an option to create a multiboot CD in their packages. For me creating a bootable CD in Linux is much easier than in Windows. There are also many free utilities that help you create a Linux bootable CD, but having a multiple boot CD is a delicacy. You can have several versions of Linux boot images on the CD - versions with support for journaling file systems, repair utilities, various breeds of Linux or BSD, or even QNX, Plan9 and more.

Why do I thing this may be good for you? Imagine you use Linux and FreeBSD simultaneously, you have more Linux distributions installed on your hard disk, but something happened to your system - there is no way to access the data anymore. Either you use a bootable diskette (but there may be many obstacles if you work with a specific system like XFS journaling file system, for example, or encrypted files system, and you find that you must have at least 5 Linux bootable diskettes to suit you), or you create a multiboot CD on which you put various breeds of Linux kernels and utilities. A little CD with 10 operating systems on it is redemption from the illusion of this world that makes you believe that something is always wrong.

I want this article to be easy, practical and intelligible for beginners, too, and I'd like to avoid too technical language that is not understood by many of us. This will help attract readers of various sort.

A bootable CD is based upon the so-called El Torrito standard - but there are other sites that explain this. Visit, for example, http://www.cdpage.com/Compact_Disc_Variations/danaboot.html
An important information for us will be that we may have up to 10 bootable operating systems on a CD that we may boot anywhere where the boot ability is supported by BIOS. The bootable ISO image file may be created with 1.44MB diskette emulation, 2.88MB diskette emulation, or hard disk emulation.

Now follows the practical guide on how to prepare a multiboot CD

First, you must have a bootable DOS or Linux diskette image file. An image is a file that contains the contents of a disk or diskette. There may be many types of image files - if you dd (disk dump) your Linux partition with a command (let's suppose that your Linux partition is on the /dev/hda1 partition):

```
dd if=/dev/hda1 of=/my_image.file
```

a file **my_image.file** will appear in your file system. Not every image file is bootable - it depends on its contents, so a good idea would be to prepare some Linux or BSD diskette image files. The simplest way would be to download such image files from the Internet. Here is the link:

http://www.ibiblio.org/pub/Linux/system/recovery/

The Ibiblio archive is very good. The image files you may download from the above URL are prepared in such a way that they are bootable, so you don't need to care much about building your own image. However, if you want to make your own image, at the above URL you may also find some utilities like Bootkit, CatRescue, SAR, disc-recovery-utils, etc., which will help you create your own bootable diskettes (or bootable image files).

The files we will need for our work, in order to make a multiboot CD, are fbsd-flp-1.0.3.bin (a bootable FreeBSD 2.8 MB diskette image), tomsrtbt, or you may create your own images from the diskettes you already have. Put your DOS or Linux diskette in the diskette drive and type the following command:

```
dd if=/dev/fd0 of=boot.img bs=512 count=2880
```

A good idea would also be to visit http://freshmeat.net and search for a keyword "mini", so you will find even some esoteric mini Linux distributions you normally don't hear about.
The site http://www.ibiblio.org/pub/Linux/system/recovery/ contains (I deleted some stuff):

- Bootkit-1.01.tar.gz
- CatRescue101E.tgz
- SAR-2.25.tar.gz
- banshee-linux.0.61.tar.bz2
- brd-2.0.tar.gz
- disc-recovery-utils-1.0.tgz
- fbsd-iso-1.0.3.bin.gz
- fspace.tgz
- genromfs-0.5.1.tar.gz
- mulinux-5r0.lsm
- mulinux-5r0.tgz
- picoboot-0.95.tar.gz
- rescue02.zip
- resque_disk-2.3.99-pre9-A.tgz
- rip-10.exe
- rip-51.iso.bin
- sash.tar.z
- tomsrtbt-2.0.103.ElTorito.288.img.bz2
- tomsrtbt-2.0.103.dos.zip
- trccs-0.8.1r2.iso.bz2
- trccs-0.8.1r2.tar.bz2
- trccs-0.8.1r2_boot_disk.img.bz2
- yard-2.1.tar.gz
- yard-prefabs-2.tgz
- zdisk-2.14.tar.gz

Some other good sites where you can download bootable diskette images:

LIAP (http://www.liap.eu.org/): LIAP is a Linux in a Pill - the site contains many 1.44MB diskette images with various utilities and kernel breeds suitable for recovery of various types of disasters.

LEKA RESCUE FLOPPY (http://leka.muumilaakso.org/): Leka Rescue Floppy is a small 1.44Mb distribution.

TOMSRTBT (http://www.toms.net/rb/): Tomsrtbt (Tom's Root Boot) is a rescue utility, a very good one. You may also download the 2.88MB image file from the above site.
You can also download bootable DOS images. Visit, for example, http://www.bootdisk.com and download DOS images if you do not have them available. The site contains DOS 5.00 to 6.22, Win 95/98/Me Bootdisks, DOS/Windows 9X/2000/XP bootdisks, Win 95/98/ME - NT4/NT5 bootdisks, DrDOS 7.X disk for Bios Flashing Basic, etc. You may also create a FreeDOS boot diskette.

First, some terms. Let's see a difference between a bootable image file of a diskette or disk and an ISO image file to be burned on a CD. What we must have are bootable diskette image files from which we will create one ISO image file.

1) You may prepare your bootable diskette images from diskettes you already have with the command:

   dd if=/dev/fd0 of=/my_image.img

   or you may download some bootable diskette image files from the Internet (see the links). Make a directory in your Linux box, for example - /CD, and copy the images to this directory (remember, you may have not more than ten bootable images). Make sure you keep the 8.3 format for file names - 8 characters for the file name and 3 characters for its suffix - this maximum is only for the compatibility issue with the DOS makebt.exe program we will later use).
2) If you want to make use of the space on the CD (ten images of bootable diskettes would only require about 14MB), place some other utilities in a subdirectory, for example, /CD/Soft. An information how to access the CD is included at the bottom of this article.
3) Run the following command from the /CD directory:

   **mkisofs -b image.img -c boot.cat -J -l -R **
   -r -o /cd.iso /CD

   The "boot.cat" or "boot.catalog" file will be automatically created, so you don't have to have it in your /CD directory - just type the command as you see it - you can type the name of any image file, as long as its name corresponds with the names of image files placed in the /CD directory. The image file included in the above command will be the one you will boot your CD from. The image files must have the size of 1.44MB or 2.8MB.
4) A cd.iso file will be created in your / directory (/cd.iso). When you check this file and mount it (mount /cd.iso /mnt -o loop), the contents of the ISO file should be seen in the directory where you

mounted it. This ISO image, if we burn the CD with it, will be bootable but only one image to boot from will be available.

5) So we must edit the ISO image to make a multiple boot CD, thus we will get other images to be included in the menu (0, 1, 2, 3, etc.) we will see when we boot the CD (we will be welcomed by a multiple boot menu with options for 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. By pressing the chosen number we will boot the desirable operating system.
6) After editing it, we may now burn the CD.

Since I don't have the time and effort to create a Perl script that would edit the ISO image for me and because the editing of the ISO image file may appear complicated for some (I want this article to be as simple as possible), it would be a good idea to use some free programs available on the Internet. One of such free programs is makebt.exe. Some time ago, I found this free program on some sites, but now I was unlucky to find it on the net, so I put it on my website http://www.tankred.sk/~juro/freebsd/makebt.zip where you can download it from.
You may run makebt.exe in DOSEMU, BOCHS emulator (http://bochs.sourceforge.net), or you can download DOS system diskette images available at http://www.bootdisk.com, or make a FreeDOS bootable diskette and boot your PC with it in order to run the makebt.exe utility. If you don't have a DOS partition, the best idea would be to use DOSEMU emulator - DOSEMU can also access Linux partitions, where you may have your CD.ISO file waiting to be "grasped in your clever hands".
When you run MAKEBT.EXE at the DOS prompt, it will ask for the full path and filename of the ISO file to be modified: you will type the name of the ISO file with multiple boot diskette images in it, for example, CD.ISO, and you will see the following screen:

```
-------------------------------------------------
Make Multiple Boot CD-ISO Image Modifier ver 1.02
ISO File path and name: cd.iso
Bootable Disk Image Boot media type Default LBA
------------------- ---------------- ------- ---
BC ) BOOT.CAT
1) FBSD.IMG         1.44M Floppy             Y
2) LINUX.IMG        2.88M Floppy             -
3) PLAN9.IMG        1.44M Floppy             -
4) QNX.IMG          1.44M Floppy             -
5) OPENBSD.IMG      2.88M Floppy             -
6)
7)
8)
9)
10)
<TAB> = move between fields, up/down arrows = move
between rows, F1 = Confirm
Press 'y' key to make this image as default boot
```

BC stands for Boot Catalog. You just write boot.cat and don't worry about it anymore, as you already used this string in the above mkisofs command (it is, however, important that the ISO image file contains the string "boot.cat" in it). Now you carefully type the names of the images. You have to type the name of images in the DOS 8.3 format (this is a DOS restriction for file names - the file may have only 8 characters and suffix 3 characters maximum).

In the middle of the screen you will choose from 1.44MB floppy emulation, 2.88MB floppy emulation, hard disk emulation, or no emulation. We will only use 1.44MB and 2.88MB emulation (if you want to make a hard disk emulation, make a 650MB Linux partition and copy there the filesystem of your Linux system you booted your hard disk from - experiment...) Use the right keyboard arrow to select between the types of emulation. On the right of the screen you have to choose one bootable image as the default one by pressing "Y".

When you are finished, press F1 (you may try this several times, as the program may not respond everytime). The program is intelligent - if you typed the image file name incorrectly, you will receive a warning message (after pressing F1). Do not include any descriptions for boot images in the menu that follows after pressing F1, as this feature is mostly exploitable in SCSI CD-ROMs and I haven't studied it very much.

That's it. Now you may burn your CD.

cdrecord -v speed=8 dev=0,0,0 /cd.iso

When you boot the CD, you will not see descriptions for operating systems, only numbers. The first and the second number will (0,1) usually stand for the same operating system. I had not much time to experiment with this issue, but a good idea would be to write down the number, so that you know which operating system you are going to boot from.

We deal here with diskette images and emulation, so if you boot your images with the multiple boot CD you just created, you may access your CD-ROM by typing "mount /dev/hdc /mnt", for example, and have also access to your /Soft directory, where you may have other utilities you plan to work with later. In case of a DOS system disk, you should include drivers to access the CD-ROM.

If you want to study or make a Linux program to patch the ISO file, you can compare an ordinary ISO image file with one boot possibility only with the ISO file patched by the makebt.exe utility. A good binary patcher is a diff utility by Giuliano Pochini. Bdiff is a simple and small program for making what the very common utilities "diff" and "patch" do with text files, but also works with binary files. It may be downloaded from: http://space.virgilio.it/g_pochini@virgilio.it/ - however, both ISO files must be identical. The diff utility (for comparing files) will show you the place (offsets) where the information with a multiboot flag was written. It is sector 17 (Boot Volume Descriptor) and the Boot Catalog Sector.

I created many multiboot CD's with the above information and I have never experienced a problem. But first, in order to avoid writing unusable CD-Rs - I had some problems making my own OS/2 images - burn the ISO image on rewritable CD-RW disks. Enjoy!

This article is re-printed with permission. The originals can be found at:
http://www.linuxgazette.com/issue85/sipos.html

Why Free Software' s Long Run TCO must be lower

Author: Brendan Scott <brendanscott@optusnet.com.au>

[Editor's note: Brendan Scott is an ICT lawer for a well known law firm. Brendan and I were formerly columnists for *Australian Internet World*. I've reprinted Brendan's piece in full, as it provides a somehwat different perspective on the economics of modern software than the ones generally acknowledged.]

Abstract

This paper argues that the long run total cost of operations (TCO) for a suite of proprietary software must necessarily be greater than that for an equivalent suite of free software, with the TCO benefits maximised in the case of the GPL and GPL-like free software. The total cost of operation of a suite of free software is the price determined by a competitive market for a bundle of goods and services associated with that suite. Because the source code is open and not subject to limitations on development or distribution, the market for services relating to that code will be perfectly competitive. A rational vendor will use a proprietary route for a program only where releasing that program in that way will allow them to increase their profit above that which would be returned to them by the operation of a competitive market. This result should be hardly surprising, given that the express objective of copyright law is to mandate a market failure and permit software creators to extract above market rents as an incentive for the creation of that software.

Customers attempting to evaluate a free software v proprietary solution can confine their investigation to an evaluation of the ability of the packages to meet the customer's needs, and may presume that the long run TCO will favor the free software package. Further, because the licensing costs are additional dead weight costs, a customer ought to also prefer a free software solution with functionality shortfalls where those shortfalls can be overcome for less than the licensing cost for the proprietary solution.

1. Definitions

Within this paper we make use of two key terms - free software and proprietary software. The "free" in the term free software is the "free as in 'speech' not the free as in 'beer'"[1] (although the "free beer" meaning is often also relevant). As we will see, another non-beer meaning of "free" in this context is "'free' as in 'market' not free as in "beer'". There are two key definitions in this area - a relatively loose one of free software (available from http://www.gnu.org/philosophy/free-sw.html) developed by the Free Software Foundation definition and another of the similar (but more comprehensively set out) concept of open source developed by the Open Source Initiative www.opensource.org/ (available from www.opensource.org/docs/definition.html). The key characteristics of these definitions are that that software is freely distributable, that the source code must accompany the distributions and that everyone is permitted to modify and distribute modifications of that code on the terms of that license. Free software licenses do not require distribution of the modified code. However if it is distributed, the source code must accompany it. The final characteristic is the requirement as to the license which must be used if modifications are distributed. In the bare Open Source Initiative definition, the license must allow modifications to be distributed on the basis of the original license, but does not require it. Contrast this with the GNU GPL (the main free software license endorsed by the Free Software Foundation) which requires that if redistribution occurs, that redistribution must be on the terms of the GPL. As we will see later, the GPL gives stronger total cost of operations results than other free software licenses because of its requirement to impose the GPL for any modifications which are distributed. We therefore draw a distinction between "GPL free software" (that is, free software, such as, but not only, the GPL which impose free software licensing requirements on distributions) and non-GPL free software.

A definition of proprietary software is harder to arrive at. Indeed, it is better to identify by reference to the level of control over the source code for the software which is asserted by its author. In effect, proprietary software is software which is not free software. The conclusions in this paper rely on the assumption that there is sufficient market power held by the person controlling the software so as to be able to extract above market rents. As a general rule this power is founded on the legislative monopoly of copyright. There may be instances where that power is exercisable through other means (such as patents, lack of interoperability and network externalities). However, for convenience in this paper we will assume that copyright is its source. The fact that no license fee is charged for a given application - as is often the case with internet desktop browsers for example, does not, by itself, make it free software. This is because there is no access provided to the source code. As the customer has no access to the source code, it would still be proprietary software. Indeed, even where the software is provided for free in conjunction with access to the source code for that software, the software is proprietary software if access to the source code is provided subject to restrictions on the ability to modify or distribute that code. For example, software under Microsoft's "Shared Source" program nevertheless remains proprietary software.

2. TCO vs TCnO?

TCO is often referred to as the total cost of "ownership" rather than of operations, as used in this paper. It has traditionally been used as a means of measuring the impact of certain strategies on a business. A canonical example is that of deciding whether to buy a cheap printer which requires expensive proprietary consumables. While the cost of acquiring the printer can be significantly lower than that of acquiring a competing printer, its cost to use

may in fact be much higher because of the need to pay higher prices for consumables going forward. In order to address this difficulty a manager can cost a given printer by taking the buy price of the printer and adding the total cost to make all of the prints it is expected to make during its lifetime. Other factors may also be added, such as the cost of its maintenance or an apportionment of the salaries for people employed to administer the printer over its lifetime. Thus managers can arrive at a single figure which better reflects the real cost of a printer's acquisition than its mere buy price. This gives managers an objective basis on which to make comparisons of different printers against a common metric and thus aids decisions about printer purchasing. Now, while ownership is a concept making perfect sense in the context of printers it is rather more problematic in relation to software. In relation to neither free software nor proprietary software does a user acquire "ownership" per se of the software in question. That said, the term "ownership" does describe the rights of a user of free software relatively well - the main right of "ownership" that a free software user does not enjoy is the right to exclude others from the use of the software and to prevent modification to or distribution of that software.

On the other hand, user "ownership" of software as a concept is anathema to proprietary software, the fundamental assumptions of which revolve around ownership of the software by the vendor. It is therefore odd to discuss the total cost of "ownership" in relation to, for example, a copy of the Windows 95 operating system because users have no such ownership. The user will, at best, have some form of (often extremely restrictive) license. Indeed, some might argue that a significant (and often uncosted) component of the cost of "ownership" of proprietary software is that users don't own it at all.

Ironically, the additional expenses incurred in relation to proprietary software are actually costs which arise from a purchaser's failure to own the software in question. As we will see, they are largely costs of non-ownership. Main among those costs of non-ownership are the monopoly rents that non ownership allows vendors to extract over and above the price that would be determined by the market. Free software vendors may object that the adoption of an existing acronym (TCO) which obscures the existence of these costs of non-ownership implicitly aids proprietary software vendors. However, for the purposes of convenience, we retain the acronym TCO in this paper. We use it to mean the total cost of acquiring and operating a given suite of software over its lifetime, rather than of "owning" that suite.

3. The Business Model for Proprietary Software

The cornerstone of proprietary software is the grant of monopolies by the legislature to private individuals. These monopolies, most commonly effected through copyright legislation, take rights (notably the right to copy) away from citizens generally and vest them in the monopoly holder for a given work.

Originally, the reason for this was to ensure that the State in Stuart and Tudor England could censor the publications of all private individuals (it is no coincidence that free software principles are strongly related to free speech principles). The State granted a monopoly right over printing to the Stationers' Company in return for the Stationer's Company acting as the State's censors. This had the side effect of creating vastly profitable publishing enterprises for the printers who administered this censorship. With the Glorious Revolution in England in the late 1600s, the State sought to reconcile the various political bodies within England at the time and, as part of that reconciliation (indeed, one of parliament's conditions for permitting William and Mary to ascend the throne) was a relaxation of the laws in relation to censorship. Around this time saw the emergence of a different justification for the granting of these private monopolies (the beneficiaries in practice of these monopolies remained unchanged). This new theory was expressed in the economic concepts of incentives and protection from competition.

The theory is, effectively, this: that the act of printing costs nothing, or next to nothing, however the act of creating a literary work requires substantial effort. If other printers (at the time it was printers from Scotland threatening to compete with English printers as a result of political union in the early 1700s) are able to print and distribute copies of a work without restriction, then they will be able to undercut a printer who invests their time and money in creating a market for that work. This would mean that there is no incentive to make such an investment. Ultimately, it would mean a decline in the production of literary works. In other words, the unregulated operation of the market will render the printing business non-viable. This is undesirable, therefore the printing business ought to be protected and the most appropriate form of protection is the grant of private monopolies to prevent this form of free riding.
Thus the State has a role in protecting printers from competition. This allows them to extract rents above what the operation of a free market would return to them. This has the socially desirable goal of promoting the printing business and, through it, gives an incentive to "learned men" to create "useful works" (to borrow the wording from the Statute of Anne 1709, the first copyright statute).

This theory has remained largely unchanged since the early 1700s. When implemented in the copyright clause in the Constitution it was modified to say that the right vests in authors, rather than printers, although the fact that these rights can be sold has led, in practice, to their being held and exercised by printers and publishers (the software industry is no exception in this regard, with most copyright in commercially marketed software vested in companies rather than individuals). However, the copyright clause still clearly states that copyright is only a means to an end. The object expressed in the Constitution is that the monopolies are granted for the "progress" of "science" and the "useful arts".

In practice, the copyright legislation creates a framework to permit the conversion of services (computer programming) into products (the software created from that programming). The theory is that this conversion encourages investment in such programming by allowing the investment to be amortized over the sale of each copy of the ultimate product. Arguably, in the absence of such legislation, the creator of a work would need to recoup their investment and make a profit during their lead time to market before others begin distributing copies of their work at much lower prices (others can sell at lower prices as they are not carrying the cost of development, only of distribution).

To summarize, therefore:

(a) A creator of software is unlikely to bother to develop their programs if they are not protected from free competition in the market.

(b) The State protects potential creators from competition as a means of encouraging the production of software.

(c) The State grants this protection through taking rights away from the balance of the population and vesting them in the author of the program by way of monopoly.

(d) The State, in doing so, is attempting to balance the interests of authors against the interests of consumers and achieve socially desirable goals (in this case, increased production of software).

In short, the State regulates the market and legislatively mandates the failure of the market in order to promote the creation of software.

4. The Free Software Business Model

If proprietary software is an example of mandated market failure, free software is the market's response to that failure. In some respects the free software movement can be regarded as the market self correcting for the market failure mandated by Congress. The free software business model leverages off two key characteristics of software - that it is both non-rival (use by A does not inhibit use by B) and durable (in theory software does not wear out) - to ensure its creation through the aggregation of a number of individual small contributions. To quote another analogy in this area - each person contributes a brick, but ultimately each person receives a house in return. As software is nonrival it is possible for each person to take the full benefit of the whole. Further, as it is durable, substantial value can be aggregated from minor contributions over a long period. It is as if everyone receives a full house when they have only contributed a brick.

Early free software modes did not include obligations in respect of the redistribution of modified code. This led to some of the software produced under this model being incorporated into commercial products. Contributors contributed their brick, but didn't get a house back. As the development model for free software is an incremental and community oriented one, this incorporation into commercial products acted as a disincentive to participation. Contributors contribute their brick in the expectation that, in the long term, a mansion will be returned to them (as the result of aggregation of individual contributions). If contributors expect to be required to reacquire modifications of their own code they are helping to create a mansion for someone else. They effectively make an investment (denominated in code) but receive no return on that investment. The requirement that modified code be returned as free software is therefore an important incentive in the creation of free software. That said, what is the most appropriate approach to free software is a philosophically contentious one and subject to much debate within the free software community. In the proprietary model, the incentive to create software arises from creators seeking to act as vendors of code. It is very much a vendor centric model. Free software on the other hand is a customer centric model. In the free software model, the assumption is made that necessity forces customers to create code for themselves (that is, development of free software is driven by the use value of the software). Free software provides a resource to assist that creation on the condition that any software which results from the use of that resource which is also distributed must be contributed back to the resource for others to use. The incentive to create code is different for different players in the market - a hardware vendor might promote the creation of free software code in order to bundle software with their hardware, thereby increasing the value of their hardware (known as "widget frosting"). End users create code in order to fulfill their own needs. An independent programmer might code to create a market for consulting or maintenance services. To extend the analogy introduced earlier, that there is a house returned does not mean that everyone who contributed to it did so to live in that house.

One of the argued benefits of the proprietary software mode is that it allows "big bang" developments to occur. It may be that there is a market for mature software with specific functionality, but no market for immature software. Without a plan no one will contribute bricks. In this environment incremental development championed by the free software paradigm may not be appropriate as there is no critical mass to kick start the development. So how does free software arise?

Experience shows that free software is primarily seeded through two pathways - which we will refer to in this paper as "strict free software seeding" and "repurposing". Under strict free software seeding a program emerges in an organic manner from nothing other the support of the customer community. Many individual programmers each have an "itch to scratch" and their communal scratching produces a valuable result. The Linux kernel, the gcc compiler, the GIMP (image manipulation - a free version of Photoshop), and Apache (the most popular web server in the world) and text formatting programs like TeX are examples. However, there are significant free software applications, including Open Office (an office

suite), and Netscape/Mozilla (browser/email client) which have been seeded from a different pathway. These programs are the remnants of commercial ventures which have been beaten in the marketplace, were repurposed by their makers as free software then adopted and perfected by the application of the free software method.

Their creators, sensing their inability to successfully take the software to market, released the kernel of each of these projects as free software. While these owners are entitled to claim some level of altruism, their actions are a standard marketplace response to competitive threats. In particular, identify those product lines where your competitor is extracting the most premium and spoil that market[2]. For example, by releasing cheap products into that market to make it impossible for anyone to profit in those areas. This permits the competitor to convert a previous market for value added products from which they could not make profit into a market for commodity products from which no one can extract profit ("if I can't have it, then nobody can"), thereby shifting the basis of competition to a different market (and one less favorable to the current incumbent). This increases their competitiveness while decreasing that of the market incumbent. Thus, Sun and IBM, who otherwise have no prospect of gaining market share on in desktop systems or applications have become significant contributors to initiatives such as Java and GNU/Linux, thereby opening up markets for themselves in service, support, training and consultancy which were previously closed to them. This behavior is exactly what is expected of a rational economic actor. While there are elements of it in non-software markets, its progress in the area of software has been more marked because software is non-rival and durable as mentioned above. It has minimal transport and holding costs, and minimal deconstruction costs to extract valuable elements. The long term play for a free software contributor is effective code ownership (except, as we noted above, for the right to exclude others). Free software also invokes the market to solve difficult problems relating to the efficiency of resource allocation. Inefficient development resources - that is programs for which there is insufficient demand - are bred out of the code base by lack of support from the user community. Indeed such projects are likely to die in vitro. This is in contrast with proprietary software which will continue to be funded until the venture capital backing it has been consumed. Conversely, successful software is self selected - code will not mature if there are no users willing to adopt and support it. If there exists a mature free software program performing certain functions it is almost a necessary consequence that there is a market for that software.

5. Components of TCO to be considered

A July 2001 MITRE Corporation report[3] lists the cost elements to be considered when considering a software business case.

Table 5 of that report lists those elements as:

```
A Direct Costs
1. Software and Hardware
        1.1 Software

            (a) Purchase price

            (b) Upgrades and additions

            (c) Intellectual property/licensing fees

        1.2 Hardware

            (a) Purchase price

            (b) Upgrades and additions

2. Support Costs
        2.1 Internal

            (a) Installation and set-up

            (b) Maintenance

            (c) Troubleshooting

            (d) Support tools (e.g., books,
            publications)

        2.2 External

      (a) Installation and set-up

      (b) Maintenance

      (c) Troubleshooting

3. Staffing Costs
        3.1 Project management
        3.2 Systems engineering/development
        3.3 Systems administration
            (a) Vendor management
        3.4 Other administration
            (a) Purchasing
            (b) Other

        3.5 Training
4. De-installation and Disposal

B. Indirect Costs
1. Support Costs
        1.1 Peer support
        1.2 Casual learning
        1.3 Formal training
        1.4 Application development
        1.5 Futz factor

2. Downtime
```

To this list there are also other indirect costs arising from the use of proprietary software which are related to administration of licensing requirements. These include the tracking of software usage (costs of creating and administering procedures and of acquiring software) and, in some cases, conducting software audits. This list does not appear to anticipate switching costs where a business is moving from one system to another (although these can be accounted for indirectly through headings such as "installation and set up" or "training").

6. Analysis of Markets

To the extent that there is demand for any of these elements, a market will emerge to satisfy that demand. This is true of both free software and proprietary software. The long term price in the market will be determined by a number of factors, revolving around the degree to which competition is available in that market. In any given market each

means of excluding competition available to the seller of free software is also available to the seller of proprietary software. In addition the seller of proprietary software has additional exclusions that they can apply - in particular the exercise of their State sanctioned monopoly over distribution and modification. The lowest cost to customers will arise where competition is maximized. Therefore as additional exclusionary mechanisms are used by the seller of proprietary software in relation to one of these markets so the cost to customers to make acquisitions in that market will rise. If the seller was not able to extract these above market rents by relying on these exclusionary mechanisms, the seller would not release the software as proprietary software.

The long run price in any of these given markets, (in the absence of cross subsidy), can therefore be broken into two components - the competitive market price for those services plus an additional monopoly premium that can be extracted. In the case of free software the competition in each market is at its highest, so the long run monopoly premium is zero. Because the long run monopoly premium in the case of proprietary software must be greater than zero, in any of these markets the long run TCO of proprietary software must be greater than that of equivalent free software. This result should hardly be surprising. It is, after all, the express purpose of the copyright legislation.

6.1 Transient Effects

The discussion above has deliberately confined itself to long run costs. It is easy to construct short run examples where this TCO result is reversed. For example, many contemporary analyses of business case costs take a snapshot approach. That is, they take empirical observations as to the state of any one of these markets and aggregate calculated costs which result from those observations. While this information is of value to managers needing to make short term decisions it is inadequate for strategic planning purposes, which may need to anticipate a lifetime for a technology platform in excess of five or seven years. For example, as at the date of this paper, operating systems such as Windows and the Macintosh OS have been mature systems for several years and, in that time, a number of ancillary services have emerged to complement each of them. Their free software equivalent, Linux, has not enjoyed the same period of maturity and has therefore not had the same period of time to refine efficiencies in business processes supporting each of these markets. It should not be surprising therefore to expect free software markets to produce higher costs for a given category based on current circumstances using this form of analysis if a sufficiently short enough evaluation window is manufactured for the purposes of the study.

6.2 Monopolies in particular submarkets

The argument can be applied equally where the anti-competitive effect of release of code as proprietary software is limited to only a few of the mentioned submarkets. In those submarkets where it is unable to affect competition in the market, the price set will be the same as the free software market price for that component. Of course, in relation to the balance, the aggregate price for those components must exceed that of the free software market for those components.

However, in the long term, when the costs across each of these separate submarkets are aggregated, the cost of free software in an open market must be less than that of equivalent proprietary software. While proprietary software may loss leader in some submarkets in order to leverage greater returns in others, overall this leverage must be profitable for the company selling proprietary software - if it wasn't the vendor would not engage in the practice. A vendor will not rely on the copyright path to ensure returns below that which could be returned to them by the market. Release of code as proprietary software will only occur where such a release can increase the vendor's returns in aggregate.

6.3 Examples of monopoly premiums

These increased returns express themselves in rents through a number of pathways. For example, the monopoly over modifications to software protects a vendor from competition in relation to both their time to implement modifications and their cost to implement those modifications. It may also use this monopoly to leverage into ancillary submarkets such as training, documentation or maintenance. Thus an operating system monopoly could be used to leverage into specific submarkets for applications running on that operating system, especially where the operating system (whether by design or oversight) is or becomes incompatible with competitors' applications in that submarket.

Further, each of these costs can be compounded by the exclusionary effect of proprietary software. Thus, because proprietary software does not allow reuse of source by other programs, the skills that individuals gain through developing or supporting proprietary software are less portable. This lack of portability leads to a greater training premium when developing or supporting other software, again pushing up development and support costs. It also leads to replication of effort.

Thus where a programmer developing free software has seen a module to perform a specified function before, they can locate the existing version of that module and incorporate it into their current development. However, in the proprietary software world they must recreate that module from scratch - technically not relying on their knowledge of the existing module (to avoid it being a derivative work) - greatly increasing the development costs through repeated reinvention. File formats are a good example of this phenomenon. As vendors of proprietary software are unable to use code for other proprietary software's file formats, they are practically required to create their own custom versions (some argue that proprietary formats are also used as an anti-competitive tool) or reverse engineer the proprietary

format. This translates directly into increased prices to the customer (they must not only pay for the added function provided by the third party provider in relation to files in that format, but also for their reverse engineering costs involved in accessing the format). This in turn increases the minimum to play for any would be entrant into the market, increasing barriers to entry and consequently reducing competition.

6.4 Cross subsidies from external markets

This analysis is not strictly valid where one or more of the mentioned submarkets is subject to subsidies from an external market. That is, a market other than one of the mentioned submarkets. This means that customers in that market are paying more in order that customers in the subsidized market can pay less. However, given that we are concerned with long run costs, it is open to question how long such a subsidy can continue in the face of competition in that external market. Further, large customers, such as government, are more likely to be acquiring in that external market so would not be quarantined from the increased costs.

6.5 Reliance on Functional equivalence

The key assumption in this analysis is that the free software suite is functionally equivalent to the proprietary suite and that there is sufficient demand for that suite (although we suspect in practice that this is implied by the first). Functional equivalence in this context does not mean that the functional characteristics of both suites are the same. Rather, it means that each of the suites can meet the functional requirements of the customer. Thus, two suites might meet the customer's requirements, and one might have additional functionality, but it is additional functionality which is irrelevant to the customer. This surplus functionality is not taken into account by this analysis. As we noted above free software is self selecting, so the existence of a market for that software might be implied from the existence of a free software equivalent.

6.6 Indirect costs

The analysis above assumes that for each of the cost components, money is the driver for the development of the market. However, in the case of some of the indirect costs listed, this is not the case. For example, peer support is something which is decoupled from pricing mechanisms in the other markets, even though it may have an effect on the cost base4. There is no reason to suspect that these indirect costs will favor one method or another in the long term. Intuitively one would expect that the greater openness permitted by free software will reduce costs in the peer support category. There is also the counterintuitive result that, in the short term, lack of applications running on a particular free software platform can actually yield cost benefits through futz factor savings. As diversionary applications are simply not available, time lost using them is equally reduced.

Switching costs seem to be a class of cost which is immune to a market analysis. Effectively an organization which has an existing proprietary solution must write off the time investment in that software when moving to a free software equivalent. It is not clear the extent to which this is a real cost in the general case. For example, there is no reason a priori why free software can not provide a seamless transition path from equivalent proprietary software. However, experience to date indicates that these costs are non trivial. Nor is it clear that that investment would not have to be written off in any event. With each new release of software, users are required to retrain.

7. The Unreasonable Effectiveness of the GPL in TCO.

7.1 Strong and weak results

A deeper analysis reveals that the argument in relation to markets is at its strongest where development forks or versions in the code must themselves remain free software. If a fork is permitted to go proprietary, then that fork will lead to a higher TCO along the lines of the argument above. Except to the extent a fork actually increases the number of consumers for the suite it must cannibalize some of the existing free software consumers, reducing demand. As demand decreases free software suppliers may leave the market, reducing competition and consequently increasing prices. In the extreme case a proprietary software vendor may engage in predatory pricing (for example by providing free support or maintenance services) to destroy the markets supporting the free software forks with a view to cornering those markets and subsequently extracting monopoly rents. In any case, where proprietary forks are permitted, the TCO result will be suboptimal5. Conversely proprietary suites which have free software forks will have a lower TCO than purely proprietary ones because customers can leverage off the more efficient markets of the free software forks.

As this paper is concerned with long run TCO, the result in relation to licenses which do not require the use of a free software license in the event of distribution of modifications must be taken with some degree of caution. The GNU GPL is a well known example of a free software license which requires the use of the GPL when (and if) redistributing modifications to GPLed free software. On the analysis above there is actually a spectrum of results:

(a) Proprietary software provides the worst option (from the point of view of long run TCO) because it minimizes competition, consequently maximizing long run TCO. Again, we note that this result is primarily due to the fact that the legislative monopoly granted over software is specifically designed to increase the return of the vendors of proprietary software in order to create an incentive for the creation of that software. In short, the legislature distorts the market for the purpose of protecting the proprietary software industry.

(b) Non-GPL free software which permits proprietary forking may or may not result in lower TCO in the long run. Whether it does or does not depends on the net effect of proprietary forking on demand in the relevant submarkets. At one end of the scale no proprietary forking in fact occurs and the result is the same as for GPLed free software. At the other end of the scale a proprietary fork can cannibalize the entirety of each market for the suite, resulting in a long run TCO equal to proprietary software (subject perhaps to a discount for the period in which it remains free software). It is conceivable that the long run TCO may actually be as high for non-GPL free software as for the most expensive proprietary software (in the event that a proprietary software vendor follows predatory pricing with price gouging subsequent to the diminution of competition). The likely outcome is something in the middle, with proprietary forks reducing demand somewhat but not entirely. We refer to this as the "weak TCO result".

(c) GPL free software maximizes demand in markets by excluding market capture by proprietary forks. These licenses therefore maximize competition in the long run, and consequently result in the lowest TCO in the long run. We refer to this as the "strong TCO result".

7.2 Dangers of source code "sharing"

Source code sharing arrangements where the source code for proprietary software is made available but not on a free software basis, are dangerous for users because of the viral nature of proprietary software. What makes proprietary software so dangerous in this regard is the privileged place accorded to it by copyright law. In the absence of an agreement to the contrary, the monopoly rights in modifications carried out by users after access to the source code will vest by default in the vendor of the proprietary software, even where theterms of access to the code are silent. Thus, the viral nature of proprietary software poses a threat to the intellectual property of any organisation that has access to shared source unless explicit (and often expensive) steps are taken. This exclusion by stealth can be contrasted with, for example, the GPL, in which the terms of use and redistribution are not only explicitly stated, but that explicit statement must accompany each copy of the source code.

As this default monopoly vesting is hidden in the copyright law and not necessarily explicitly made clear to users, the unwary user could find access to shared source to be a poison pill. They may end up in the position of creating modifications for free for a vendor without taking the benefit of the work of other users performing similar fixes. Indeed, they may not technically be permitted to use even their own fixes. As such, the potential for damage that such access allows cannot be understated.

This viral nature of proprietary software means that, in order to safely access the source code to such software, a user will require management oversight and nontrivial monitoring of that access. Independent of the software provider's views on the confidentiality of their source code, a user must either have (a) an agreement, preferably in writing, which clearly and unambiguously permits that user (and, as appropriate, their employees, agents and contractors) to make modifications and to distribute those modifications; or (b) strict accessibility controls in place to prevent access to or use of that source.

8. Strategic implications for purchasers.

The lessons that potential purchasers should take from this paper are:

(a) The short term costs of proprietary systems may be lower than those of free software systems. However, free software systems hold the advantage over the long term;

(b) while free software effectively provides code ownership, the main costs of adopting a proprietary software solution are actually costs of non-ownership;

(c) These TCO advantages are both most pronounced and more certain for GPL free software licenses, which mandate free software license terms for the distribution of code modifications. Non-GPL free software poses a significant risk of proprietary forking in the future and therefore yields only a weak TCO result, although one likely to be better than that for proprietary software.

(d) Source code sharing arrangements are no substitute for the code ownership provided by free software. The viral nature of proprietary software means that such sharing arrangements pose potentially serious risks to a user if not strictly managed.

(e) Making purchasing decisions on a 1, 2 or even 3 year window will have the effect of locking a purchaser into higher total cost of operations over the longer term. Purchasers must "bite the bullet" if they want to maximize real long term savings and access them sooner. This will be particularly pronounced where major purchasers, such as government, delay entry into the free software market thereby inhibiting the development of ancillary markets for the support of free software;

(f) Strategic decisions in relation to the choice between free software and proprietary software should come down to one of evaluation of fit to functional requirements, with long run total cost of operations presumed to favor the free software. Fit to functional requirements can include such things as network externalities. As between free software variants GPL free software should be preferred to minimize long run TCO.

(g) Where there appears to be an ongoing market for the free software in question, the organization should consider whether any shortfall in functionality can be made good at a price equal to or less than the licensing costs of the equivalent proprietary suite. That said, without demand a free software solution is unlikely to have reached sufficient maturity to present as a viable alternative. Its mere presence can of itself be indicative of the existence of an ongoing

market.

We note that these conclusions are independent of the fact or amount of any license fees charged in relation to the proprietary software. Of course, where these costs are substantial they will make a further contribution to the TCO of the suite. Also, free software advocates argue that the lifecycle of free software programs is much longer than of their proprietary equivalents. If this is true, the need to replace a proprietary system more often will also increase the TCO for the proprietary suite.

I welcome comments on and criticisms of this paper. I would like to thank Karl O. Pinc for his comments on earlier drafts of this paper.

References

[1] Attributed to Richard Stallman.

[2]Some of the examples given in this paragraph are of the "commodization of complements". That is a business attempts to reduce the status of goods and services which are complementary to its own to that of commodity. In so doing they are able to increase the demand for their own goods or services.

[3] Kenwood, C., A Business Case Study of Open Source Software, June 2001, www.mitre.org/support/papers/tech_papers_01/kenwood_software/index.shtml

[4] Individuals who are able to resolve issues through peer support do not place a strain on support resources. It is open to debate whether peer support is a real cost saving if, for example, resources are actually being drawn from other functions within the organisation - in this case a peer support saving is simultaneously a futz factor cost.

[5] At the risk of laboring the point, the theory behind such an increase is that the higher TCO results from increased returns to software vendors as an incentive for them to create software which would not otherwise be created.

This article is re-printed with permission. The originals can be found at:
http://members.optushome.com.au/brendanscott/papers/freesoftwaretco150702.html

Brendan's home-page, where other articles he's written are available, is located here:
www.members.optushome.com.au/brendanscott.

AUUG Election Procedures

These rules were approved by the AUUG Incorporated Management Committee on 14th December 1994.

1. NOTICE OF ELECTION

The Returning Officer shall cause notice of election to be sent by post to all financial members no later than 15th March each year.

2. FORM OF NOTICE

The notice of election shall include:

a) a list of all positions to be elected, namely:

- President
- Vice President
- Secretary
- Treasurer
- Ordinary Committee Members (5)
- Returning Officer
- Assistant Returning Officer

b) a nomination form;

c) the date by which nominations must be received (in accordance with clause 21(2) of the Constitution, this date is 14th April);

d) the means by which the nomination form may be lodged;

e) a description of the format for a policy statement.

3. POLICY STATEMENT

A person nominated for election may include with the nomination a policy statement of up to 200 words. This word limit shall not include sections of the statement stating in point form the nominee's name, personal details and positions held on, or by appointment of, the AUUG Management Committee and chapters. Policy statements exceeding the word limit shall be truncated at the word limit when included in the ballot information. The Returning Officer may edit policy statements to improve readability, such edits being limited to spelling, punctuation and capitalisation corrections and spacing modifications. Use of the UNIX wc program shall be accepted as an accurate way to count words.

4. RECEIPT OF NOMINATIONS

In accordance with clause 21(2) of the Constitution, nominations shall be received by the Secretary up until 14th April. A nomination shall be deemed to have been received by the due date if one of the following is satisfied:

- it is delivered by post to AUUG Inc's Post Box, the AUUG Secretariat's Post Box or the AUUG Secretariat's street address no later than 2 business days after 14th April and is postmarked no later than 12 midday on 14th April;
- it is delivered by hand to the Secretary or the AUUG Inc Secretariat no later than 5pm on 14th April;
- it is transmitted by facsimile to the Secretary or the AUUG Inc Secretariat no later than 5pm on 4th April.

5. REQUIREMENT FOR A BALLOT AND DUE DATE

In accordance with clause 21(5), no later than 1st May, the Secretary

- shall advise the Returning Officer of all valid nominations received;

- and if a ballot is required, shall advise the Returning Officer of a date no later than 15th May for the ballot for all contested election.

In accordance with clause 42(3), the due date for return of ballots shall be 4 weeks after the date advised above.

6. FORM OF BALLOT PAPER

The ballot paper shall contain:
- details of all positions for which the number of nominations exactly equals the number of positions to be filled;
- for each position for which a ballot is required, the names of all persons seeking election to that position,
- except those already elected to a higher position, with a square immediately to the left, for the elector to place a voting preference;
- instructions on how to complete the ballot paper;
- instructions on how to return the ballot paper;
- a brief description of how the ballot is to be counted.

The ballot shall not contain any identification of existing office-bearers.

The ballot shall be accompanied by a copy of all policy statements submitted by all persons nominated, including any persons elected unopposed.

These policy statements may be truncated or modified as outlined in 3.

7. METHOD OF VOTING

Voting for each position shall be by optional preferential vote. The number "1" must be placed against the candidate of the elector's first preference, and a number other than ``1" against any or all of the other candidates.

Preferences shall be determined by the numbers placed against other candidates, which must be strictly monotone ascending to count as preferences.
A vote shall be informal if:
- it does not have the number ``1" against exactly one candidate.

8. SECRECY OF BALLOT

The ballot paper shall be accompanied by two envelopes, which may be used by the elector to ensure secrecy.

On completion of the ballot paper, the paper may be placed inside the smaller envelope. This envelope is then placed inside a second envelope. The elector must then sign and date the outer envelope, making the following declaration:

``I, _____,
member-number _____,
declare that I am entitled to vote in this election on behalf of the voting member whose membership number is shown above, and no previous ballot has been cast on behalf of this voting member in this election."

9. RETURNING BALLOT

To be considered to have been returned by the due date, the ballot paper together with declaration as above must be returned by one of the following means:

- it is delivered by post to AUUG Inc's Post Box, the AUUG Secretariat's Post Box or the AUUG Secretariat's street address no later than 2 business days after the due date and is postmarked no later than 12 midday on the due date;
- it is delivered by hand to the Returning Officer or the AUUG Inc Secretariat no later than 5pm on the due date.

10. METHOD OF COUNTING

Where there is an election for a single position, the votes shall be counted by the preferential method. Where there is more than one position to be filled, the votes shall be counted by the modified preferential Hare Clark system described in schedule 1.

11. METHOD OF ELECTION

A person may be elected to only one position.
Elections shall be counted in the order of positions described in 2(a). When counting ballots, any person previously elected shall be deemed withdrawn from that election, and all ballot papers shall be implicitly renumbered as though that person was not included.

12. NOTIFICATION OF RESULT

In accordance with clause 42(7) of the Constitution, the Returning Officer shall advise the Secretary in writing of the result no later than fourteen days after the due date. The Returning Officer shall advise all candidates for election of the result no later than fourteen days after the due date. The Returning Officer shall advise the AUUGN Editor in writing of the result no later than fourteen days after the due date. The AUUG Editor shall include the results in the first issue of AUUGN published after receiving the results from the Returning Officer.

13. PUBLICATION OF THESE RULES

The Returning Officer shall advise the AUUGN Editor of the current rules, and the AUUGN Editor shall cause the current rules to be published on or after 1st January each year. Where no issue of AUUGN has been posted by 28th February in any calendar year, the Returning Officer shall cause the current rules to be distributed with the notice of election.

14. OCCASIONAL VARIATION FROM THESE RULES

Subject to the Constitution, the Management Committee may authorise occasional variations from these rules. Such variations shall be advised in writing to all members at the next stage in the

election process in which information is distributed to members.

15. EXECUTION

Where these rules require the Returning Officer to carry out an action, it shall be valid for the Returning Officer to delegate execution to the Secretariat from time to time employed by the Management Committee.

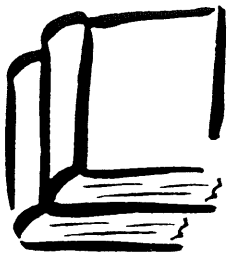
16. RETENTION OF BALLOT PAPERS

The Secretary shall retain ballot papers and member declarations (as specified in 8) until the AUUG AGM of the calendar year following the year of the election, unless a general meeting of AUUG directs the Secretary to hold them for a longer period.

SCHEDULE 1

1. Each ballot paper shall initially have a value of one.
2. The value of each ballot paper shall be allotted to the candidate against whose name appears the lowest number on the paper among those candidates not elected or eliminated. If there is no such candidate (i.e. the ballot paper is exhausted) the ballot paper shall be set aside.
3. A quota shall be calculated by dividing the number of formal votes by one more than the number of positions remaining to be elected, and rounding up to the next whole number.
4. If any candidate is allotted a total value greater than the quota, that candidate shall be declared to elected, the ballot papers allotted to that candidate shall be assigned a new value by multiplying their previous value by the excess of the candidate's vote above the quota divided by the candidate's total vote. This new value shall be truncated (rounded down) to 5 decimal places. Ballot papers that subsequently have a value of zero shall be set aside. Steps 2 and 3 shall then be repeated.
5. If no candidate is allotted a total a total value greater than the quota, the candidate who is allotted the lowest total value among those candidates not elected or eliminated shall be eliminated. Steps 2 and 3 shall then be repeated.
6. Where two or more candidates are declared elected at the same stage of counting according to Step 4 have an equality of votes, and it is necessary to determine which is deemed elected first, or a candidate is required to be eliminated under Step 5, and two or more candidates have an equally low vote, the Returning Officer shall return to the immediately preceding stage of counting and in the case of candidates elected, deem first elected the candidate with the highest vote at the immediately preceding stage, and in the case where a candidate is to be eliminated, eliminate the candidate with the lowest vote at the immediately preceding stage.

Where an equality of votes still exists at the immediately preceding stage, the Returning Officer shall continue proceeding to preceding stages until a result can be determined. In the event that candidates have maintained an equality of votes throughout the entire counting process, the Returning Officer shall determine which candidate is to be determined first elected or to be eliminated by lot in the presence of the Assistant Returning Officer.



AMERICAN
BOOK STORE

10% DISCOUNT
TO AUUG MEMBERS
ON OUR COMPLETE RANGE
OF COMPUTER, BUSINESS
AND GENERAL BOOKS

173 Elizabeth St, Brisbane Queensland 4000
Ph: (07) 3229 4677 Fax: (07) 3221 2171 Qld Country Freecall: 1800 177 395
american_bookstore@compuserve.com

Name: _____ Date: _____

Address: _____

_____ Post Code: _____

Phone Number: _____

Payment Method: ❑ Cheque ❑ Money Order ❑ Amex ❑ Bankcard
❑ Diners ❑ Mastercard ❑ Visa

Card Number: _____

Expiry Date: _____ Signature: _____

This is a: ❑ Special Order ❑ Mail Order ❑ Book on Hold

QUANTITY	TITLE	PRICE

SUBTOTAL $ .....
LESS 10% DISCOUNT $ .....
POST & PACK $ .....
TOTAL $ .....

POSTAGE AND HANDLING FEES: 1 BOOK $6.00 2-4 BOOKS $7.00
BOOKS OVER $70.00 WE WILL SEND CERTIFIED - PLEASE ADD ANOTHER $1.50 OR WAIVE CERTIFIED DELIVERY.

FOR SPECIAL ORDERS, PLEASE ENCLOSE $10.00 PER BOOK AS A DEPOSIT.

AUUG Chapter Meetings and Contact Details

CITY	LOCATION	OTHER
ADELAIDE	We meet at Internode, Level 3/132 Grenfell St aka 'the old AAMI building', at 7 pm on the second Wednesday of each month.	Contact sa-exec@auug.org.au for further details.
BRISBANE	Inn on the Park 507 Coronation Drive Toowong	For further information, contact the QAUUG Executive Committee via email (qauug-exec@auug.org.au). The technologically deprived can contact Rick Stevenson on (07) 5578-8933. To subscribe to the QAUUG announcements mailing list, please send an e-mail message to: <majordomo@auug.org.au> containing the message "subscribe qauug <e-mail address>" in the e-mail body.
CANBERRA	Australian National University	
HOBART	University of Tasmania	
MELBOURNE	Various. For updated information See: http://www.vic.auug.org.au/	The meetings alternate between Technical presentations in the even numbered months and purely social occasions in the odd numbered months. Some attempt is made to fit other AUUG activities into the schedule with minimum disruption.
PERTH	The Victoria League 276 Onslow Road Shenton Park	
SYDNEY	Meetings start at 6:15 pm Sun Microsystems Ground Floor 33 Berry Street (cnr Pacific Hwy) North Sydney	The NSW Chapter of AUUG is now holding meetings once a quarter in North Sydney in rooms generously provided by Sun Microsystems. More information here: http://www.auug.org.au/nswauug/

FOR UP-TO-DATE DETAILS ON CHAPTERS AND MEETINGS, INCLUDING THOSE IN ALL OTHER AUSTRALIAN CITIES, PLEASE CHECK THE AUUG WEBSITE AT HTTP://WWW.AUUG.ORG.AU OR CALL THE AUUG OFFICE ON **1-800-625655.**

Application / Renewal Individual or Student Membership of AUUG Inc.

Use this tax invoice to apply for, or renew, Individual or Student Membership of AUUG Inc. To apply online or for Institutional Membership please use **http://www.auug.org.au/info/**

This form serves as Tax Invoice.

Please complete and return to:

AUUG Inc, PO Box 7071, BAULKHAM HILLS BC NSW 2153, AUSTRALIA

If paying for your membership with a credit card, this form may be faxed to AUUG Inc. on +61 2 8824 9522.

Please do not send purchase orders.
Payment must accompany this form.

Overseas Applicants:

- Please note that all amounts quoted are in Australian Dollars.
- Please send a bank draft drawn on an Australian bank, or credit card authorisation.
- There is a $60.00 surcharge for International Air Mail
- If you have any queries, please call AUUG Inc on +61 2 8824 9511 or freephone 1800 625 655.

Section A:

Personal Details

Surname: .....

First Name: .....

Title: ..... Position: .....

Organisation: .....

Address: .....

.....

Suburb: .....

State: ..... Postcode: .....

Country: ..... Phone Work: .....

Phone Private: ..... Facsimile: .....

E-mail: .....

Membership Number (if renewing): .....

Student Member Certification

For those applying for Student Membership, this section is required to be completed by a member of the academic staff.

I hereby certify that the applicant on this form is a full time student and that the following details are correct:

Name of Student: .....

Institution: .....

Student Number: .....

Signed: .....

Name: .....

Title .....

Date Signed: .....

Section B: Prices

Please tick the box to apply for Membership. Please indicate if International Air Mail is required.

Renew/New* Individual Membership	$110.00 (including $10 GST)	❒
Renew/New* Student Membership	$27.50 (including $2.50 GST)	❒
Surcharge for International Air Mail	$60.00	❒

* Delete as appropriate.

GST only applies to payments made from within Australia. Rates valid from 1st October 2002.

Section C: Mailing Lists

AUUG mailing lists are sometimes made available to vendors. Please indicate whether you wish your name to be included on these lists:

Yes ❒ No ❒

Section D: Payment

Pay by cheque

Cheques to be made payable to **AUUG Inc.** Payment in Australian Dollars only.

OR Pay by credit card

Please debit my credit card for A$ .....

Bankcard ❒ Mastercard ❒ Visa ❒

Card Number: ..... Expires: .....

Name on card: ..... Signature: .....

Date Signed: .....

Section E: Agreement

I agree that this membership will be subject to rules and bylaws of AUUG Inc as in force from time to time, and this membership will run from the time of joining/renewal until the end of the calendar or financial year as appropriate.

Signed: .....

Date Signed: .....

This form serves as Tax Invoice. AUUG ABN 15 645 981 718