news@UK

The Newsletter of UKUUG, the UK's Unix and Open Systems Users Group
Published electronically at `http://www.ukuug.org/newsletter/`

Volume 16, Number 4 **ISSN 0965-9412** **December 2007**

Contents

News from the Secretariat

Jane Morrison

The minutes from the Annual General Meeting held on 26th September have not been circulated to members via hard copy. All members were emailed on 10th October and advised that the minutes and associated documents could be found on the web site at
`http://www.ukuug.org/events/agm2007/`

Printed copies are available on request.

You will note from the minutes we have one new Council member, Paul Waring.

Paul is based in Manchester and is enthusiastic about UKUUG. We are hoping he can bring new ideas and assist with work on ongoing items.

Council have also appointed a new Events Co-ordinator, Zeth Green who is joining us in a part-time paid position.

Zeth is actually organising the next UKUUG event, FAB 2008 (a two day seminar) on 19th and 20th February. You should find the information booklet and booking form for this enclosed with this Newsletter.

The first meeting of the new Council was a phone-conference held on 30th October. Alain Williams has agreed to continue in the role of Chairman and Sam Smith is continuing as Treasurer. The next face-to-face meeting will be held in London on 30th November.

Issues for Council at the meeting will be the organisation and programme for the Spring conference being held in Birmingham from 31st March to 2nd April 2008. Full event details (information booklet and booking form) will be sent to all members in early January.

I am sure this event is going to prove very popular and we are currently looking for sponsors to help with general event expenses and the conference dinner.

I still have a stock of UKUUG polo shirts available at a price of £15.00 including VAT. The polo shirts have the UKUUG logo embroidered in silver onto a black shirt and are available in Small, Medium, Large and Extra Large.
`http://www.ukuug.org/shirt/`

The annual membership subscription invoices will be sent out in January, please look out for your invoice and as always prompt payment will be gratefully received!

I would like to take this opportunity to wish you all a very Happy Christmas and a Peaceful New Year.

The Secretariat will close from 17th December to January 2nd 2008.

Please note the copy date for the next issue of the Newsletter (March 2008) is 22nd February 2008. As always, contributions from members are welcome and can be sent to
`newsletter@ukuug.org`

UKnet

Alain Williams

UKUUG has a sister company UKnet Ltd. This was set up some twenty years ago. Under the telecoms regulations at the time people needed to be a member of a group to be able to do inter-site networking, so it was needed for people to gain ‘net’ access. It has always been a financially dormant company and over the years Council had felt that there might be some re-sale value in the name; we have however failed to find a buyer. It costs us some £30 per year to keep the dormant company.

We also own the trade mark ‘UKnet’. Trade marks are renewable every 10 years, UKnet needs renewing in February at a cost of £423.25. UKUUG Council is strongly of the opinion that the trade mark is not worth renewing.

We also believe that keeping the dormant company is no longer worthwhile and propose to dissolve it next year.

If any UKUUG member is of a different opinion, would they please contact me as soon as possible. If we do not hear persuasive objections we will proceed as above. We would be more than happy to sell the trademark and/or company.

Note that renewing the trade mark is a separate issue from keeping the company.

RT Tutorial report

Alain Williams

On 16th October UKUUG ran a tutorial on RT – Request Tracker: the ticketing system. The tutor was Jesse Vincent who is the author of RT and the founder of Best Practical Solutions.

Attendance exceeded our expectations, 17 of us listened to Jesse's well delivered talk. Most delegates were considerably more experienced with RT than I was. I was there since I am somehow sysadmin for a UKUUG machine on which RT is a key application, as I was writing this review your newsletter editor 'phoned to tell me of a problem – I am going to have to put this day to good use.

After a quick overview and history of RT Jesse spent most of the time on various aspects of RT configuration and customisation.

This included: the email gateway, how to set up a development sandbox, how RT uses Mason, introduction to the RT data model, what is stored in the database, how to access via DBIx::SearchBuilder, I18N & L10N, how to produce custom queries and reports, including graphs and pie charts, debugging, profiling and tuning.

I learned a lot and the post tutorial questionnaires gave a high level of approval to the day.

The UKUUG seminar on Databases was running in an adjacent room, we joined them for a pint at the end of the day.

If there are any topics on which you think UKUUG should run a tutorial, please let me know.

Liaison News

Sunil Das

An excellent talk by Michael Meeks entitled "The Linux Desktop, Present and Future" was presented after the September AGM. Michael is a Distinguished Engineer at Novell. Novell is a Gold Sponsor member and continue to support UKUUG's activities.

In this edition Peter Salus continues his observation of the U.S. legal system while Kirk McKusick and Pawel Dawidek have submitted papers on file systems.

The UKUUG Seminar "Databases and the Web" was held in October. The slides of talks are available at UKUUG's website. Below is a brief summary of the talks compiled with the help of notes by Council members Paul Waring and Howard Thompson.

MySQL for High Availability Web Applications. Tushar Joshi, Turtle Networks
Uptime is different from availability. These terms are often used interchangeably even though they do not mean the same thing. Several configurations were covered that provide for hot-standby, clustering, on-line backup and load-balancing together with the limitations of such systems. Problems of MySQL as a high availability database server include the master-slave requiring manual intervention if the master goes down, having to keep entire databases in memory and there is no way of altering the set-up (e.g. to add new nodes) whilst the system is running.

PostgreSQL Web Projects: From Start to Finish. Simon Riggs, 2ndQuadrant
Leading UK developer gave insight into the internal structure of PostgreSQL that enables it to provide capabilities that are unavailable elsewhere, together with indications of its stability and performance under load. Certain commercial competitors do not permit their products to be benchmarked. This has restricted the extent to which PostgreSQL is a very real competitor to some better known alternatives.

Tuning Tips for Linux and AIX to run a Database. Nigel Griffiths, IBM
Comprehensive account on the pitfalls that arise when maintaining performance on large scale systems. Up to 15% increase in performance was reported when changing page size to 64Kb. Half of the database-related problems encountered were caused by poor disk layout rather than "hardware issues".

Introducing Oracle Application Express. Julian Lane, Oracle
The software is freely available and easy to use. Examples showed how well Application Express can import data from other sources such as spreadsheets or Access databases, and quickly turn them into a database application with both a friendly front-end and an admin backend.

DB2 on Linux for Web Development. John Pickford, IBM
Coverage of problems that need to be addressed in accessing data using both XML and SQL type interfaces. Processing of incoming XML data to provide appropriate indexing is essential to enable searching and fast reconstruction of the data as required by external applications. An example of transforming GIS data from a database to GML → SVG + HTML → Browser was impressive.

iPlayer Server-Side: The Evolution of an XML Tool-Chain. Matthew Browning, BBC
RDBMS proved an unsuitable option for the server-side requirements of the iPlayer software mainly due to performance reasons and the need to rewind, and playback history for testing purposes. Storing a lot of data is unnecessary as the content is only available for seven days. XML (or more specifically TV Anytime: www.tv-anytime.org) is used to control the output and ensure that the two hour service level agreement is adhered to.

GUUG Spring event

We have received notification of GUUG's Fruehjahrsfachgespraech (Spring Talk) which will be held between the 11th and 14th of March in Munich, Germany.

The target audience for this event is system administrators and system programmers.

Papers are invited for this event: please see the "Call for Papers":
`http://guug.de/ffg/cfp.html`

The first two days of the event will consist of tutorials and the second two days will take the form of a conference. Some talks and tutorials will be delivered in German and some in English.

Full details can be seen at
`http://guug.de/ffg`

UKUUG on Facebook

Roger Whittaker

Members may be interested to discover that a UKUUG group has been formed on the Facebook social networking site.

At the time of writing there seem to be about 20 members in the group.

The group can be found at
`http://www.facebook.com/group.php?gid=2401192863`

Talking to Torvalds

Brian Runciman

He hates cell phones, but thinks that acceptance of the open source concept is now taken for granted – in a good way. BCS managing editor Brian Runciman interviewed Linus Torvalds after he received the BCS Lovelace Medal.

Let's start by clearing this up: Lie-nux or Lynne-ux?
Well it is Lynne-ux, because I was working on Minix, it sounds similar.

Isn't your name pronounced Lie-nus – like Linus from Peanuts?
I was partly named after him, but in Finland it's pronounced Lee-nus anyway – so it doesn't work that way either!

BCS is pursuing professionalism in IT – what are your thoughts on this?
Well, I'm self-taught. The Linux community itself has been self-organising and most of the early people involved were students, not even programmers. Many were from a physics background, using Linux for calculations and physics problems. So I don't really have an opinion in this area.

Linux hasn't made huge inroads on the desktop – is that a worry?
Well the desktop is not really a huge market for Linux. It's hard to get hardware manufacturers to support it. They won't even tell us, for example, how their new graphics chips will work, so it's difficult to write code for them.

The desktop is special, there's a huge inertia for people because it's the only place we are really aware that we are using software. In things like iPods the software is almost invisible – it's updated behind the scenes, it only has a limited interface.

For Linux it's been easier to be in servers or embedded systems because all that matters is that it works – people don't care about what runs the system, only about the output. The desktop is a more direct interaction.

Looking back on 16 years or so of Linux is there anything you would do differently given the chance?
I think it's the 16th birthday in a couple of weeks. Realistically I wouldn't have done anything differently. A few technical choices weren't always right, but they are easily solved. But things like the community set-up and the licensing approach worked out really well, spot on.

Did Linux develop as you expected?
No, I expected it to be much smaller, just as something I would use – the commercial side wasn't planned at all. I did it for fun, because I was interested in software and interaction.

Now I do it mostly for the social side – I never thought of myself as sociable in that way, but it's fun to interact via email.

How people use Linux now was also unexpected. For example, it's used a lot in third world countries, although I'm not involved personally in that, and that's good.

BCS is celebrating its 50th anniversary this year. What developments in computing do you think are the most exciting in the last 50 years?
What I think about is IT's ubiquity. I don't like the word, but when I started out programming was viewed as odd and computing itself was expensive. But now interacting with computers is so much a part of people's everyday lives.

That has produced changes in the way we programme too. Because of the power of hardware now we do things that wouldn't have been possible even five to 10 years ago – things that then would have required real high-end hardware to solve problems that would have seemed unrealistic. So the growth of computing power and its availability have been the big things for me.

There are concerns in the industry with getting people into IT in a serious way, not just superficial usage. How can we do that?
I have this problem with my daughters now. They use computers, but to make the leap to real interest in

technology and programming, I don't know. My eldest daughter is doing a Lego robotics course this year, which involves a little programming. She uses Linux, but only to play games, surf the web and email.

Personally my interest started with a small computer club – there were only two of us, those who actually had computers. I had a Vic 20 and my friend a ZX81.

Who in the IT industry inspired you, or was a role model for you?
No-one from the industry. My role models were always scientists.

In fact I use this as an analogy for open source: it's science versus alchemy or witchcraft. I've always felt that the way to progress is to "stand on the shoulders of giants" and that needs openness, not a form of witchcraft where how you do things is hidden, where you protect your knowledge.

My inspirations were people like Einstein and Newton.

What recent developments by others have impressed you most?
One of the people I'm a fan of is Richard Dawkins. He's very famous in the US and UK for his anti-religious stance, but I've found his books on biology and genetics much more interesting. I find biochemistry fascinating.

Will computing and genetics ever truly overlap then? I'm thinking of Ray Kurzweil's views, for example?
I think the singularity idea is over-hyped. It's a cool vision, but whether it will happen in reality I'm not so sure.

But both sides can give to each other. To make AI really successful it requires an understanding of neural networks and how the brain works. And genetic research today is possible because of the processing power we now have. The protein folding stuff is very processing power intensive.

How do you feel about the media's approach to open source?
It's been very easy. Journalists tend to be very interested in communications and openness so I found it very easy to explain the philosophy behind open source. Most of my family are journalists so I've noticed how my interviews have changed over the years in this regard. I don't do many interviews; enough to see the changes, but not so many that I can't see how they've changed.

In the early days there was scepticism about why doing things openly would help or even work for programming and code. But now it's absolutely taken for granted that this is a good way of doing things. People know it's a great model. So the questions I get have changed because the assumptions have changed.

What are the biggest challenges open source faces in the near future?
I'm not that worried because it is just a better way to do things. People think of open source as a philosophy of freedom, but I think it's just a great way to get things done.

There are some worries, software patents are number one, but that doesn't just affect open source, that's a problem for all development. It's just a bit more obvious for open source.

What one piece of careers advice would you give to someone going into the industry?
What I saw a lot when I was at university was people getting into programming because they saw it as an up and coming area and a good opportunity to make money. It's the worst reason to get into anything, but especially programming, I think.

If you don't enjoy doing it you'll never be as good as someone who does. So you'll be second rate and unhappy.

That applies to anything. You need to like what you're doing.

Quick Questions

Mac or PC?
Who cares? I have both, they both run Linux.

Are you a geek or a nerd?
I use geek, but I'll answer to nerd.

Smartphone, PDA or iPhone?
None, I hate cell phones, I find them really annoying – why are people disturbing me? I used to have one so that my wife could contact me, but I work from home now anyway.

How would you like to be remembered?
I think as someone who made a difference. A positive one, of course.

If not in IT, what would you be doing?
If I was in high school now I may well want to be a geneticist.

Linus Torvalds was awarded the BCS Lovelace Medal in 2000, and was presented it on his first visit to the UK in September 2007 by past-president David Hartley.

The citation:

Linus Torvalds created Linux, a remarkably efficient, stable and scalable UNIX-like implementation that is supplied free, over the internet in both binary and source formats. The significance of Torvalds' project is threefold. Firstly, the quality of the software engineering contributed by himself resulted in a robust and efficient operating system kernel now used by an estimated 7.5m people (as of 2000). Although the majority of Linux users host the operating system using PC hardware it has been implemented on machines as diverse as palm pilots and massively parallel networks of microprocessors.

Secondly, Torvalds demonstrated that the very best software, including operating systems, is not (and perhaps even cannot be) developed by large corporations intent on profit.

Thirdly he inspired a large and growing number of developers to contribute and cooperate on what is one of the most ambitious distributed software development projects to date.

This interview with Linus Torvalds is reproduced here by the kind permission of Brian Runciman and the BCS.

The Law's Delay

Peter Salus

In Hamlet's soliloquy in Act III, Scene 1, the gloomy Dane complains about: *"The pangs of despised love, the law's delay, . . . "*

In *Bleak House* (serialized in 1853-53), Dickens tells us of the case of Jarndyce and Jarndyce, which has been going on for so long that no one recalls what it is about. And, reverting to Shakespeare, we have a great suggestion from Dick the Butcher in *Henry VI, Part 2*: "Let's kill all the lawyers."

As I wrote a year ago, I've been following the legal antics of the SCO Group since late 2003. Since August 2007, things have become both simpler and (in some sense) sillier.

These antics are important because the outcome(s) will have permanent effect on Linux, Unix and all the FOSS applications. Probably more important than the not-quite-a-victory of Neelie Kroes over Steve Ballmer. (More a Scots "not proven" than a straight "guilty" verdict.)

With that literary introduction, let me start with the ancient history.

- On March 6, 2003 The SCO Group filed suit against IBM;
- On August 4, 2003, Red Hat filed suit against SCOG;
- On January 20, 2004, SCOG filed suit against Novell.

More recently,

On August 10, 2007, Judge Kimball delivered a number of partial summary judgements, largely in Novell's favour.

1. Novell, not SCOG, owns the UNIX copyrights;

2. SCOG is obligated to heed Novell's instruction to waive claims against IBM [that is, SCOG shouldn't even have complained about IBM];

3. "The court further concludes that because a portion of SCO's 2003 Sun and Microsoft Agreements indisputably licenses SVRX products listed under Item VI of Schedule 1.1(a) to the APA, even if only incidental to a license for UnixWare, SCO is obligated under the APA to account for and pass through to Novell the appropriate portion relating to the license of SVRX products. Because SCO failed to do so, it breached its fiduciary duty to Novell under the APA and is liable for conversion." ['conversion' is the civil equivalent of criminal theft.]

But:

4. "The court, however, is precluded from granting a constructive trust with respect to the payments SCO received under the 2003 Sun and Microsoft Agreements because there is a question of fact as to the appropriate amount of SVRX Royalties SCO owes to Novell based on the portion of SVRX products contained in each agreement." [that is, because he doesn't know how much he should sequester, Judge Kimball cannot sequester money.]

On September 10, 2007, Judge Kimball granted Novell's motion to strike a jury trial. That is, as only a few questions remain, there is no need to empanel a jury.

On September 14, 2007, the Judge rejected SCOG's request for reconsideration. Trial was set for 8:30am on September 17.

However:

On September 13, 2007, SCOG's Board of Directors had passed a resolution to file for Chapter 11 bankruptcy. And on the 14th, did so in Delaware. Of course, Darl McBride had told Judge Kimball – prior to August 10 – that there was no danger of SCOG filing for bankruptcy.

On September 14, 2007, the Utah cases were "stayed" because of the bankruptcy.

There was a hearing in Bankruptcy court on November 6, 2007. There were many topics to be ruled upon, and there were two strange moves.

First, on 17 July 2007, SCOG had set up a subsidiary, "Cattleback Holdings", and transferred a patent to it "without consideration". They'd now like the court to say it's OK that they did that and let them pay some unnamed employees some bonuses, and the filed document tells all about why it isn't a fraudulent transfer. Of course, this was all prepetition (don't you know), so there's no question of this being a fraudulent transfer. The patent is: "Method and Apparatus for Monitoring Computer Systems and Alerting Users of Actual or Potential System Errors". It has never been tested, and there appears to be scads of prior art. It might be noted that US bankruptcy law states: "a transfer made for no consideration within a year before bankruptcy is voidable if the transfer was. . . " and goes on for half a page.

Second, on 23 October 2007, SCOG petitioned the court to be permitted to sell its "UNIX business" and other assets to "a newly formed entity affiliated with one or more funds managed by JGD Management Corp. d/b/a York Capital Management" – interestingly, the JGD is James G. Dinan, the president and CEO of York, and who is listed on SCOG's submission to the US Securities and Exchange Commission in 2004 as owning over 90,000 shares. For some strange reason, Novell and IBM have filed objections to this. Judge Gross – the bankruptcy judge – has put a decision off.

Back to 6 November.

First on the Agenda was the arbitration (in Switzerland!) between SCOG and SUSE. SCOG wants it stayed. SUSE says 'no,' and anyway the Swiss arbitrators aren't under the jurisdiction of a US court. Not to worry, rules Judge Gross, if both SCOG and SUSE tell the arbitrators to pause, they will. (For those of you with long memories, this goes back to UnitedLinux. Remember *Bleak House*!)

Second was a question about SCOG's lawyers continuing – something that is customarily a no-no in bankruptcies. Judge Gross puts off a decision. (Remember "the law's delay"!)

Then we came to the big one: Novell's motion regarding lifting the stay on the trial in Utah. Judge Gross wonders if this should be heard by Judge Kimball in Utah or by him, and how much further along Kimball is than Judge Gross is. No one remarked that they had come to the point where SCOG had filed for bankruptcy in this court only to escape judgement in Utah. Lewis for Novell explains what Kimball has already done. Then Michael Jacobs spoke for Novell. Judge Gross wants to know how complicated the trial will be. Jacobs states that things are fairly simple. He spoke about the Sun and Microsoft agreements and how much should be apportioned to Novell. The trial should only take about 4 days to complete. He mentions about the "constructive trust" being heard by Gross's court and having Kimball determining apportionment. Mr. Spector for SCO claims Novell will not be harmed by waiting a few months. He says Kimball ruled against both Novell and IBM concerning the "constructive trust".

Well, that's not true. Kimball said Novell deserved a constructive trust, but he just didn't know how much to put into it. Spector admits that some of the Sun and Microsoft funds might be Novell's, but all that money is gone, he asserts, although some might still exist. [In the fantasy universe of SCOG and its lawyers, money isn't fungible.] Judge Gross promised a decision – but will "not take too long". [More "delay".]

On 13 November, the "US Trustee in Bankruptcy" filed objection to the proposed sale, noting:

> *"Per the Debtors' public filings, the success of the Linux litigation hinges upon the Debtors' ability to establish ownership of certain intellectual property rights, the same rights which the United States District Court for the District of Utah recently found were owned by Novell".*

On 16 November, we had the next installment. The two important issues (to me) were the "proposed sale" and the permission to retain an "interim Chief Financial Officer". Surprise! Hamlet won. [More "delay".] Decisions were put off till 5 December. Judge Gross gets to digest his Thanksgiving turkey.

To me, the two big open questions are

- Why do both SCOG and York Capital want to close this deal by 31 December? Tax reasons?
- What entity is backing York Capital? York is an investment company. Someone/something is offering up millions to perpetuate this FUD. The "usual suspects" here would be Microsoft, Oracle or Sun. But who knows?

Incidentally, as of their 31 July 07 10-Q filing with the SEC, SCOG had (only) $10.4 million cash, $3.1 million receivables and $1.3 million Other Current Assets, for a total of $14.9 million in Current Assets. As of the end of November, the 30 September form has not yet been filed.

Other events

The **big** news where the FSF is concerned is that GPLv3 is alive, functioning and getting accepted. Brett Smith has written a splendid "Quick Guide". Read it at:
`http://www.fsf.org/licensing/licenses/quick-guide-gplv3.html`

It begins with:

Nobody should be restricted by the software they use.

There are four freedoms that every user should have:

- the freedom to use the software for any purpose,
- the freedom to share the software with your friends and neighbors
- the freedom to change the software to suit your needs, and
- the freedom to share the changes you make.

When a program offers users all of these freedoms, we call it free software.

Yep.

A brief history of the BSD Fast File System

Marshall Kirk McKusick

I first started working on the UNIX file system with Bill Joy in the late 1970s. I wrote the Fast File System, now called UFS, in the early 1980s. In this article, I have written a survey of the work that I and others have done to improve the BSD file systems. Much of this research has been incorporated into other file systems.

1979: Early Filesystem Work

The first work on the UNIX file system at Berkeley attempted to improve both the reliability and the throughput of the file system. The developers improved reliability by staging modifications to critical filesystem information so that the modifications could be either completed or repaired cleanly by a program after a crash [14]. Doubling the block size of the file system improved the performance of the 4.0BSD file system by a factor of more than 2 when compared with the 3BSD file system. This doubling caused each disk transfer to access twice as many data blocks and eliminated the need for indirect blocks for many files. The performance improvement in the 3BSD file system gave a strong indication that increasing the block size was a good method for improving throughput. Although the throughput had doubled, the 3BSD file system was still using only about 4% of the maximum disk throughput. The main problem was that the order of blocks on the free list quickly became scrambled as files were created and removed. Eventually, the free-list order became entirely random, causing files to have their blocks allocated randomly over the disk. This randomness forced a seek before every block access. Although the 3BSD file system provided transfer rates of up to 175 kbytes per second when it was first created, the scrambling of the free list caused this rate to deteriorate to an average of 30 kbytes per second after a few weeks of moderate use. There was no way of restoring the performance of a 3BSD file system except to recreate the system.

1982: Birth of the Fast File System

The first version of the current BSD file system was written in 1982 and became widely distributed in 4.2BSD [13]. This version is still in use today on systems such as Solaris and Darwin. For large blocks to be used without significant waste, small files must be stored more efficiently. To increase space efficiency, the file system allows the division of a single filesystem block into fragments. The fragment size is specified at the time that the file system is created; each filesystem block optionally can be broken into two, four, or eight fragments, each of which is addressable. The lower bound on the fragment size is constrained by the disk-sector size, which is typically 512 bytes. As disk space in the early 1980s was expensive and limited in size, the file system was initially deployed with a default blocksize of 4 kbytes so that small files could be stored in a single 512-byte sector.

1986: Dropping Disk-Geometry Calculations

The BSD filesystem organization divides a disk partition into one or more areas, each of which is called a cylinder group. Historically, a cylinder group comprised one or more consecutive cylinders on a disk. Although the file system still uses the same data structure to describe cylinder groups, the practical definition of them has changed. When the file system was first designed, it could get an accurate view of the disk geometry, including the cylinder and track boundaries, and could accurately compute the rotational location of every sector. By 1986, disks were hiding this information, providing fictitious numbers of blocks per track, tracks per cylinder, and cylinders per disk. Indeed, in modern RAID arrays, the “disk” that is presented to the file system may really be composed from a collection of disks in the RAID array.

Although some research has been done to figure out the true geometry of a disk [5, 10, 23], the complexity of using such information effectively is high. Modern disks have greater numbers of sectors per track on the outer part of the disk than on the inner part, which makes calculation of the rotational position of any given sector complex to calculate.

So in 1986, all the rotational layout code was deprecated in favor of laying out files using numerically close block numbers (sequential being viewed as optimal), with the expectation that this would give the best performance. Although the cylinder group structure is retained, it is used only as a convenient way to manage logically close groups of blocks.

1987: Filesystem Stacking

The early vnode interface was simply an object-oriented interface to an underlying file system. By 1987, demand had grown for new filesystem features. It became desirable to find ways of providing them without having to modify the existing and stable filesystem code. One approach is to provide a mechanism for stacking several file systems on top of one another [22B]. The stacking ideas were refined and implemented in the 4.4BSD system [7]. The bottom of a vnode stack tends to be a disk-based file system, whereas the layers used above it typically transform their arguments and pass on those arguments to a lower layer.

Stacking uses the mount command to create new layers. The mount command pushes a new layer onto a vnode stack; a umount command removes a layer. Like the mounting of a file system, a vnode stack is visible to all processes running on the system. The mount command identifies the underlying layer in the stack, creates the new layer, and attaches that layer into the filesystem name space. The new layer can be attached to the same place as the old layer (covering the old layer) or to a different place in the tree (allowing both layers to be visible).

When a file access (e.g., an open, read, stat, or close) occurs to a vnode in the stack, that vnode has several options:

- Do the requested operations and return a result.
- Pass the operation without change to the next-lower vnode on the stack. When the operation returns from the lower vnode, it may modify the results or simply return them.
- Modify the operands provided with the request and then pass it to the next-lower vnode. When the operation returns from the lower vnode, it may modify the results or simply return them.

If an operation is passed to the bottom of the stack without any layer taking action on it, then the interface will return the error “operation not supported.”

The simplest filesystem layer is nullfs. It makes no transformations on its arguments, simply passing through all requests that it receives and returning all results that it gets back. Although it provides no useful functionality if it is simply stacked on top of an existing vnode, nullfs can provide a loopback file system by mounting the file system rooted at its source vnode at some other location in the filesystem tree. The code for nullfs is also an excellent starting point for designers who want to build their own filesystem layers. Examples that could be built include a compression layer or an encryption layer.

The union file system is another example of a middle filesystem layer. Like nullfs, it does not store data but just provides a name-space transformation. It is loosely modeled on the work on the 3-D file system [9], on the Translucent file system [8], and on the Automounter [19]. The union file system takes an existing file system and transparently overlays the latter on another file system. Unlike most other file systems, a union mount does not cover up the directory on which the file system is mounted. Instead, it shows the logical merger of the two directories and allows both directory trees to be accessible simultaneously [18].

1988: Raising the Blocksize

By 1988, disk capacity had risen enough that the default blocksize was raised to 8-kbyte blocks with 1-kbyte fragments. Although this meant that small files used a minimum of two disk sectors, the nearly doubled throughput provided by doubling the blocksize seemed a reasonable trade-off for the measured 1.4% of additional wasted space.

1990: Dynamic Block Reallocation

Through most of the 1980s, the optimal placement for files was to lay them out using every other block on the disk. By leaving a gap around each allocated block, the disk had time to schedule the next read or write following the completion of the previous operation. With the advent of disk-track caches and the ability to handle multiple outstanding requests (tag queueing) in the late 1980s, it became desirable to begin laying files out contiguously on the disk.

The operating system has no way of knowing how big a file will be when it is first opened for writing. If it assumes that all files will be big and tries to place them in its largest area of available space, it will soon have only small areas of contiguous space available. Conversely, if it assumes that all files will be small and tries to place them in its areas of fragmented space, then the beginning of files that do grow large will be poorly laid out.

To avoid these problems the file system was changed in 1990 to do dynamic block reallocation. The file system initially places the file's blocks in small areas of free space, but then moves them to larger areas of free space as the file grows. With this technique, small files use the small chunks of free space whereas the large ones get laid out contiguously in the large areas of free space. The algorithm does not tend to increase I/O load, because the buffer cache generally holds the file contents long enough that the final block allocation has been determined by the first time that the file data is flushed to disk.

The effect of this algorithm is that the free space remains largely unfragmented even after years of use. A Harvard study found only a 15% degradation in throughput on a three-year-old file system versus a 40% degradation on an identical file system that had had the dynamic reallocation disabled [25].

1996: Soft Updates

In file systems, metadata (e.g., directories, inodes, and free block maps) gives structure to raw storage capacity. Metadata provides pointers and descriptions for linking multiple disk sectors into files and identifying those files. To be useful for persistent storage, a file system must maintain the integrity of its metadata in the face of unpredictable system crashes, such as power interruptions and operating system failures. Because such crashes usually result in the loss of all information in volatile main memory, the information in nonvolatile storage (i.e., disk) must always be consistent enough to deterministically reconstruct a coherent filesystem state. Specifically, the on-disk image of the file system must have no dangling pointers to uninitialized space, no ambiguous resource ownership caused by multiple pointers, and no unreferenced live resources. Maintaining these invariants generally requires sequencing (or atomic grouping) of updates to small on-disk metadata objects.

Traditionally, the file system used synchronous writes to properly sequence stable storage changes. For example, creating a file involves first allocating and initializing a new inode and then filling in a new directory entry to point to it. With the synchronous write approach, the file system forces an application that creates a file to wait for the disk write that initializes the on-disk inode. As a result, filesystem operations such as file creation and deletion proceed at disk speeds rather than processor or memory speeds [15, 17, 24]. Since disk access times are long compared to the speeds of other computer components, synchronous writes reduce system performance.

The metadata update problem can also be addressed with other mechanisms. For example, one can eliminate the need to keep the on-disk state consistent by using NVRAM technologies, such as an uninterruptible power supply or Flash RAM [16, 31]. Filesystem operations can proceed as soon as the block to be written is copied into the stable store, and updates can propagate to disk in any order and whenever it is convenient. If the system fails, unfinished disk operations can be completed from the stable store when the system is rebooted.

Another approach is to group each set of dependent updates as an atomic operation with some form of write-ahead logging [3, 6] or shadow-paging [2, 22A, 26]. These approaches augment the on-disk state with a log of filesystem updates on a separate disk or in stable store. Filesystem operations can then proceed as soon as the operation to be done is written into the log. If the system fails, unfinished filesystem operations can be completed from the log when the system is rebooted. Many modern file systems successfully use write-ahead logging to improve performance compared to the synchronous write approach.

In Ganger and Patt [4], an alternative approach called soft updates was proposed and evaluated in the context of a research prototype. Following a successful evaluation, a production version of soft updates was written for BSD in 1996. With soft updates, the file system uses delayed writes (i.e. write-back caching) for metadata changes, tracks dependencies between updates, and enforces these dependencies at write-back time. Because most metadata blocks contain many pointers, cyclic dependencies occur frequently when dependencies are recorded only at the block level. Therefore, soft updates track dependencies on a per-pointer basis, which allows blocks to be written in any order. Any still-dependent updates in a metadata block are rolled back before the block is written and rolled forward afterward. Thus, dependency cycles are eliminated as an issue. With soft updates, applications always see the most current copies of metadata blocks, and the disk always sees copies that are consistent with its other contents.

1999: Snapshots

In 1999, the file system added the ability to take snapshots. A filesystem snapshot is a frozen image of a file system at a given instant in time. Snapshots support several important features, including the ability to provide backups of the file system at several times during the day and the ability to do reliable dumps of live file systems.

Snapshots may be taken at any time. When taken every few hours during the day, they allow users to retrieve a file that they wrote several hours earlier and later deleted or overwrote by mistake. Snapshots are much more convenient to use than dump tapes and can be created much more frequently.

To make a snapshot accessible to users through a traditional filesystem interface, the system administrator uses the mount command to place the replica of the frozen file system at whatever location in the namespace is convenient.

Once filesystem snapshots are available, it becomes possible to safely dump live file systems. When dump notices that it is being asked to dump a mounted file system, it can simply take a snapshot of the file system and run over the snapshot instead of on the live file system. When dump completes, it releases the snapshot.

2001: Raising the Blocksize, Again

By 2001 disk capacity had risen enough that the default blocksize was raised to 16-kbyte blocks with 2-kbyte fragments. Although this meant that small files used a minimum of four disk sectors, the nearly doubled throughput provided by doubling the blocksize seemed a reasonable trade-off for the measured 2.9% of additional wasted space.

2002: Background Fsck

Traditionally, after an unclean system shutdown, the filesystem check program, fsck, has had to be run over all the inodes in a file system to ascertain which inodes and blocks are in use and to correct the bitmaps. This check is a painfully slow process that can delay the restart of a big server for an hour or more. Soft updates guarantee the consistency of all filesystem resources, including the inode and block bitmaps. With soft updates, the only inconsistency that can arise in the file system (barring software bugs and media failures) is that some unreferenced blocks may not appear in the bitmaps and some inodes may have to have overly high link counts reduced. Thus, it is completely safe to begin using the file system after a crash without first running fsck. However, some filesystem space may be lost after each crash. Thus, there is value in having a version of fsck that can run in the background on an active file system to find and recover any lost blocks and adjust inodes with overly high link counts.

With the addition of snapshots, the task becomes simple, requiring only minor modifications to the standard fsck. When run in background cleanup mode, fsck starts by taking a snapshot of the file system to be checked. Fsck then runs over the snapshot filesystem image doing its usual calculations, just as in its normal operation. The only other change comes at the end of its run, when it wants to write out the updated versions of the bitmaps. Here, the modified fsck takes the set of blocks that it finds were in use at the time of the snapshot and removes this set from the set marked as in use at the time of the snapshot – the difference is the set of lost blocks. It also constructs the list of inodes whose counts need to be adjusted, then uses a new system call to notify the file system of the identified lost blocks so that it can replace them in its bitmaps. It also gives the set of inodes whose link counts need to be adjusted; those inodes whose link count is reduced to zero are truncated to zero length and freed. When fsck completes, it releases its snapshot. The complete details of how background fsck is implemented can be found in McKusick [11, 12].

2003: Multi-Terabyte Support

The original BSD fast file system and its derivatives have used 32-bit pointers to reference the blocks used by a file on the disk. At the time of its design in the early 1980s, the largest disks were 330 Mbytes. There was debate at the time whether it was worth squandering 32 bits per block pointer rather than using the 24-bit block pointers of the file system it replaced. Luckily, the futurist view prevailed, and the design used 32-bit block pointers.

Over the 20 years since it has been deployed, storage systems have grown to hold over a terabyte of data. Depending on the blocksize configuration, the 32-bit block pointers of the original file system run out of space in the 1-to-4-terabyte range. Although some stopgap measures can be used to extend the maximum-size storage systems supported by the original file system, by 2002 it became clear that the only long-term solution was to use 64-bit block pointers. Thus, we decided to build a new file system, one that would use 64-bit block pointers.

We considered the alternatives of trying to make incremental changes to the existing file system versus importing another existing file system such as XFS [27] or ReiserFS [20]. We also considered writing a new file system from scratch so that we could take advantage of recent filesystem research and experience. We chose to extend the original file system, because this approach allowed us to reuse most of its existing code base. The benefits of this decision were that the 64-bit-block-based file system was developed and deployed quickly, it became stable and reliable rapidly, and the same code base could be used to support both 32-bit-block and 64-bit-block filesystem formats. Over 90% of the code base is shared, so bug fixes and feature or performance enhancements usually apply to both filesystem formats.

At the same time that the file system was updated to use 64-bit block pointers, an addition was made to support extended attributes. Extended attributes are a piece of auxiliary data storage associated with an inode that can be used to store auxiliary data that is separate from the contents of the file. The idea is similar to the concept of data forks used in the Apple file system [1]. By integrating the extended attributes into the inode itself, it is possible to provide the same integrity guarantees as are made for the contents of the file itself. Specifically, the successful completion of an fsync system call ensures that the file data, the extended attributes, and all names and paths leading to the names of the file are in stable store.

2004: Access-Control Lists

Extended attributes were first used to support an access control list, generally referred to as an ACL. An ACL replaces the group permissions for a file with a more specific list of the users who are permitted to access the files. The ACL also includes a list of the permissions each user is granted. These permissions include the traditional read, write, and execute permissions, along with other properties such as the right to rename or delete the file [21].

Earlier implementations of ACLs were done with a single auxiliary file per file system that was indexed by the inode number and had a small fixed-sized area to store the ACL permissions. The small size kept the size of the auxiliary file reasonable, since it had to have space for every possible inode in the file system. There were two problems with this implementation. The fixed size of the space per inode to store the ACL information meant that it was not possible to give access to long lists of users. The second problem was that it was difficult to atomically commit changes to the ACL list for a file, since an update required that both the file inode and the ACL file be written in order to have the update take effect [28].

Both problems with the auxiliary file implementation of ACLs are fixed by storing the ACL information directly in the extended-attribute data area of the inode. Because of the large size of the extended attribute data area (a minimum of 8 kbytes and typically 32 kbytes), long lists of ACL information can be stored easily. Space used to store extended attribute information is proportional to the number of inodes with extended attributes and the size of the ACL lists they use. Atomic updating of the information is much easier, since writing the inode will update the inode attributes and the set of data that it references, including the extended attributes in one disk operation. Although it would be possible to update the old auxiliary file on every fsync system call done on the file system, the cost of doing so would be prohibitive. Here, the kernel knows whether the extended attribute data block for an inode is dirty and can write just that data block during an fsync call on the inode.

2005: Mandatory Access Controls

The second use for extended attributes was for data labeling. Data labels provide permissions for a mandatory access control (MAC) framework enforced by the kernel. The kernel's MAC framework permits dynamically introduced system-security modules to modify system security functionality. This framework can be used to support a variety of new security services, including traditional labeled mandatory access control models. The framework provides a series of entry points that are called by code supporting various kernel services, especially with respect to access control points and object creation. The framework then calls out to security modules to offer them the opportunity to modify security behavior at those MAC entry points. Thus, the file system does not codify how the labels are used or enforced. It simply stores the labels associated with the inode and produces them when a security module needs to query them to do a permission check [29, 30].

2006: Symmetric Multi-Processing

In the late 1990s, the FreeBSD Project began the long hard task of converting their kernel to support symmetric multi-processing. The initial step was to add a giant lock around the entire kernel to ensure that only one processor at a time could be running in the kernel. Each kernel subsystem was brought out from under the giant lock by rewriting it to be able to be executed by more than one processor at a time. The vnode interface was brought out from under the giant lock in 2004. The disk subsystem became multi-processor-safe in 2005. Finally, in 2006, the fast file system was overhauled to support symmetric multi-processing, completing the giant-free path from system call to hardware.

Further Information

For those interested in learning more about the history of BSD, additional information is available from `http://www.mckusick.com/history/`

References

[1] Apple, “Mac OS X Essentials, Chapter 9 Filesystem, Section 12 Resource Forks” (2003).

[2] D. Chamberlin and M. Astrahan, “A History and Evaluation of System R,” Communications of the ACM (24, 10) (1981), pp. 632-646.

[3] S. Chutani, O. Anderson, M. Kazar, W. Mason, and R. Sidebotham, “The Episode File System,” USENIX Winter 1992 Technical Conference Proceedings (January 1992), pp. 43-59.

[4] G. Ganger and Y. Patt, “Metadata Update Performance in File Systems,” First USENIX Symposium on Operating Systems Design and Implementation (November 1994), pp. 49-60.

[5] J. L. Griffin, J. Schindler, S.W. Schlosser, J.S. Bucy, and G.R. Ganger, “Timing-accurate Storage Emulation,” Proceedings of the USENIX Conference on File and Storage Technologies (January 2002), pp. 75-88.

[6] R. Hagmann, “Reimplementing the Cedar File System Using Logging and Group Commit,” ACM Symposium on Operating Systems Principles (November 1987), pp. 155-162.

[7] J. S. Heidemann and G.J. Popek, “File-System Development with Stackable Layers,” ACM Transactions on Computer Systems (12, 1) (February 1994), pp. 58-89.

[8] D. Hendricks, “A Filesystem for Software Development,” USENIX Summer 1990 Technical Conference Proceedings (June 1990), pp. 333-340.

[9] D. Korn and E. Krell, “The 3-D File System,” USENIX Summer 1989 Technical Conference Proceedings (June 1989), pp. 147-156.

[10] C.R. Lumb, J. Schindler, and G.R. Ganger, “Freeblock Scheduling Outside of Disk Firmware,” Proceedings of the USENIX Conference on File and Storage Technologies (January 2002), pp. 275-288.

[11] M.K. McKusick, “Running Fsck in the Background,” Proceedings of the BSDCon 2002 Conference (February 2002), pp. 55-64.

[12] M.K. McKusick, “Enhancements to the Fast Filesystem to Support Multi-Terabyte Storage Systems,” Proceedings of the BSDCon 2003 Conference (September 2003), pp. 79-90.

[13] M.K. McKusick, W.N. Joy, S.J. Leffler, and R.S. Fabry, “A Fast File System for UNIX,” ACM Transactions on Computer Systems (2, 3) (August 1984), pp. 181-197.

[14] M.K. McKusick and T.J. Kowalski, "Fsck: The UNIX File System Check Program," in 4.4BSD System Manager's Manual (Sebastopol, CA: O'Reilly & Associates, 1994), vol. 3, pp. 1-21.

[15] L. McVoy and S. Kleiman, "Extent-like Performance from a UNIX File System," USENIX Winter 1991 Technical Conference Proceedings (January 1991), pp. 33-44.

[16] J. Moran, R. Sandberg, D. Coleman, J. Kepecs, and B. Lyon, "Breaking Through the NFS Performance Barrier," Proceedings of the Spring 1990 European UNIX Users Group Conference (April 1990), pp. 199-206.

[17] J. Ousterhout, "Why Aren't Operating Systems Getting Faster as Fast as Hardware?" USENIX Summer 1990 Technical Conference (June 1990), pp. 247-256.

[18] J. Pendry and M.K. McKusick, "Union Mounts in 4.4BSD-Lite," USENIX 1995 Technical Conference Proceedings (January 1995), pp. 25-33.

[19] J. Pendry and N. Williams, "AMD: The 4.4BSD Automounter Reference Manual," in 4.4BSD System Manager's Manual (Sebastopol, CA: O'Reilly & Associates, 1994), vol. 13, pp. 1-57.

[20] H. Reiser, "The Reiser File System" (January 2001):
`http://www.namesys.com/res_whol.shtml`

[21] T. Rhodes, "FreeBSD Handbook, Chapter 3, Section 3.3 File System Access Control Lists" (2003):
`http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/handbook/fs-acl.html`

[22A] M. Rosenblum and J. Ousterhout, "The Design and Implementation of a Log-Structured File System," ACM Transactions on Computer System (10, 1) (February 1992): 26-52.

[22B] D. Rosenthal, "Evolving the Vnode Interface," USENIX Winter 1990 Technical Conference Proceedings (June 1990), pp. 107-118.

[23] J. Schindler, J.L. Griffin, C.R. Lumb, and G.R. Ganger, "Track-aligned Extents: Matching Access Patterns to Disk Drive Characteristics," Proceedings of the USENIX Conference on File and Storage Technologies (January 2002), pp. 259-274.

[24] M. Seltzer, K. Bostic, M.K. McKusick, and C. Staelin, "An Implementation of a Log-Structured File System for UNIX," Proceedings of the USENIX Winter 1993 Conference (January 1993), pp. 307-326.

[25] K. Smith and M. Seltzer, "A Comparison of FFS Disk Allocation Algorithms," Proceedings of the USENIX 1996 Annual Technical Conference (January 1996), pp. 15-25.

[26] M. Stonebraker, "The Design of the POSTGRES Storage System," Very Large DataBase Conference (1987), pp. 289-300.

[27] A. Sweeney, D. Doucette, C. Anderson, W. Hu, M. Nishimoto, and G. Peck, "Scalability in the XFS File System," Proceedings of the 1996 USENIX Annual Technical Conference (January 1996), pp. 1-14.

[28] R. Watson, "Introducing Supporting Infrastructure for Trusted Operating System Support in FreeBSD," Proceedings of the BSDCon 2000 Conference (September 2000).

[29] R. Watson, "TrustedBSD: Adding Trusted Operating System Features to FreeBSD," Proceedings of the FREENIX Track at the 2001 USENIX Annual Technical Conference (June 2001), pp. 15-28.

[30] R. Watson, W. Morrison, C. Vance, and B. Feldman, "The TrustedBSD MAC Framework: Extensible Kernel Access Control for FreeBSD 5.0," Proceedings of the FREENIX Track at the 2003 USENIX Annual Technical Conference (June 2003), pp. 285-296.

[31] M. Wu and W. Zwaenepoel, "eNVy: A Non-Volatile, Main Memory Storage System," International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) (October 1994), pp. 86-97.

*This article was originally published in **;login: The USENIX Magazine,** 32, #3 (Berkeley, CA: USENIX Association, 2007). Reprinted by permission.*

Porting the Solaris ZFS file system to the FreeBSD operating system

Pawel Jakub Dawidek and Marshall Kirk McKusick

The ZFS file system made revolutionary (as opposed to evolutionary) steps forward in filesystem design, with its authors claiming that they threw away 20 years of obsolete assumptions to design an integrated system from scratch. In this article, we describe the porting of ZFS to FreeBSD, along with describing some of the key features of the ZFS file system.

Features of ZFS

ZFS is more than just a file system. In addition to the traditional role of data storage, ZFS also includes advanced volume management that provides pooled storage through a collection of one or more devices. These pooled storage areas may be used for ZFS file systems or exported through a ZFS Emulated Volume (ZVOL) device to support traditional file systems such as UFS.

Pooled storage model

File systems created by ZFS are not tied to a specified device, volume, partition, or disk but share the storage assigned to a pool. The pool may be constructed from storage ranging from a single partition up to farms composed of hundreds of disks. If more storage is needed, new disks can be added at run time and the space is automatically made available to all the file systems sharing the pool. Thus, there is no need to manually grow or shrink the file systems when space allocation requirements change. There is also no need to create slices or partitions. When working with ZFS, tools such as fdisk(8), bsdlabel(8), newfs(8), tunefs(8), and fsck(8) are no longer needed.

Copy-on-write design

File systems must be in a consistent state to function in a stable and reliable way. Unfortunately, it is not easy to guarantee consistency if a power failure or a system crash occurs, because most file system operations are not atomic. For example, when a new hard link to a file is created, the file system must create a new directory entry and increase the link count in the inode. These changes usually require writing two different disk sectors. Atomicity of disk writes can only be guaranteed on a per-sector basis. Two techniques have been used to maintain filesystem consistency when multiple sectors must be updated:

- Checking and repairing the file system with the fsck utility on boot [11], a technique that has lost favor as disk systems have grown. Starting with FreeBSD 5.0, it is possible to run the fsck program in the background, significantly reducing system downtime [9].
- To allow an immediate reboot after a crash, the file system uses soft updates to guarantee that the only inconsistency the file system might experience is resource leaks stemming from unreferenced blocks or inodes [5, 10].

McKusick added the ability to create snapshots to UFS, making background fsck possible [12]. Unfortunately, filesystem snapshots have a few disadvantages, because during one step of a snapshot all write operations to the file system are blocked. Luckily, this step does not depend on filesystem size and takes only a few seconds. However, the time of the step that sets up the snapshot grows linearly with the size of the file system and generates heavy I/O load. So even though the file system continues to operate, its performance is degraded while the snapshot is being prepared.

Once a snapshot is taken, all writes to the file system must be checked to see whether an older copy needs to be saved for the snapshot. Because of the design of snapshots, copies are rarely needed and thus do not appreciably slow down the system. A slowdown does occur when removing many small files (i.e. any file less than 96 kilobytes whose last block is a fragment) that are claimed by a snapshot. In addition, checking a file system in the background slows operating system performance for many hours because of its added demands on the I/O system. If the background fsck fails (usually because of hardware-based disk errors) the operating system needs to be rebooted and the file system must be repaired in the foreground. When a background fsck has failed, it means that the system has been running with an inconsistent file system, which implies undefined behavior.

The second technique used requires storing all filesystem operations (or only metadata changes) first in a special journal. Once the entire operation has been journaled, filesystem updates may be made. If a power failure or a system crash occurs, incomplete entries in the journal are removed and partially completed filesystem updates are finished by using the completed entries stored in the journal. Filesystem journaling is currently the most popular way of managing filesystem consistency [1, 17, 18].

The ZFS file system needs neither fsck nor journals to guarantee consistency. Instead it takes an alternate copy-on-write (COW) approach. COW means that ZFS never overwrites valid data. Instead, ZFS always writes data into a free area. When the data is safely stored, ZFS switches a single pointer in the block's parent. With this technique, block pointers never point at inconsistent blocks. This design is similar to the WAFL file system design [6].

End-to-end data integrity and self-healing

Another important ZFS feature is end-to-end data integrity. All data and metadata undergoes checksum operations using one of several available algorithms (fletcher2, fletcher4 [4], or SHA256 [14]). ZFS can detect silent data corruption caused by any defect in disk, controller, cable, driver, or firmware. There have been many reports from Solaris users of silent data corruption that has been successfully detected by ZFS. If the storage pool has been configured with some level of redundancy (RAID-Z or mirroring) and data corruption is detected, ZFS not only reconstructs the data but also writes valid data back to the component where corruption was originally detected.

Snapshots and clones

Snapshots are easy to implement for file systems such as ZFS that store data using a COW model. When new data are created, the file system simply does not free the block with the old data. Thus, snapshots in ZFS are cheap to create (unlike UFS2 snapshots). ZFS also allows the creation of a clone, which is a snapshot that may be written. Finally, ZFS has a feature that allows it to roll back a snapshot, forgetting all modifications introduced after the snapshot was created.

ZFS supports compression at the block level. Currently, Jeff Bonwick's variant of the Lempel-Ziv compression algorithm and the gzip compression algorithm are supported. Data encryption is also a work in progress [13].

Porting ZFS to FreeBSD

We describe work done by Pawel Jakub Dawidek in porting ZFS to FreeBSD in the remainder of this article. This task seemed daunting at first, as a student had spent an entire Summer of Code project looking at porting ZFS to Linux and had made little progress. However, a study of the ZFS code showed that it had been written with portability in mind. The ZFS code is clean, well commented, and self-contained. The source files rarely include system headers directly. Most of the time, they include only ZFS-specific header files and a special zfs_context.h header where system-specific includes are placed. Large parts of the kernel code can be compiled in a user process and run by the ztest utility for regression and stress testing.

So, Dawidek felt a fresh start on doing a port seemed appropriate, this time taking the approach of making minimal changes to the ZFS code base itself. Instead, Dawidek built a set of software compatibility modules to convert from the FreeBSD internal interfaces to those used by Solaris and expected by ZFS. Using this approach, he had an initial port up and running with just ten days of effort.

Solaris compatibility layer

When a large project such as ZFS is ported from another operating system, it is important to keep modifications of the original code to a minimum. Having fewer modifications makes porting easier and makes the importation of new functionality and bug fixes much less difficult.

To minimize the number of changes, Dawidek created a Solaris-compatible application programming interface (API) layer. The main goal was to implement the Solaris kernel functions that ZFS expected to call. These functions were implemented by using the FreeBSD kernel programming interface (KPI). Many of the API differences were simple, involving different function names, slightly different arguments, or different return values. For other APIs, the functionality needed to be fully implemented from scratch. This technique proved to be quite effective. For example, after these stubs were built, only 13 files out of 112 of the core ZFS implementation directory needed to be modified.

The following milestones were defined to port the ZFS file system to FreeBSD:

1. Create a Solaris-compatible API using the FreeBSD API.

2. Port the user-level utilities and libraries.

3. Define connection points in ZFS where FreeBSD makes its service requests. These service requests include:

- ZFS POSIX Layer, which has to be able to communicate with the virtual filesystem (VFS) layer
- ZFS Emulated Volume (ZVOL), which has to be able to communicate with the FreeBSD volume-management subsystem (GEOM)
- /dev/zfs, a control device that communicates with the ZFS user-level utilities and libraries

4. Define connection points in ZFS where the storage pool virtual device (VDEV) needs to make I/O requests to FreeBSD.

ZFS POSIX layer

The ZFS POSIX layer receives requests from the FreeBSD VFS interface. This interface was the hardest part of the entire port to implement. The VFS interface has many complex functions and is quite system-specific. Although the Solaris and FreeBSD VFS interfaces had a common heritage twenty years ago, much has changed between them over the years. VFS on Solaris seems to be cleaner and a bit less complex than FreeBSD's.

ZFS emulated volume

A ZFS VDEV managed storage pool can serve storage in two ways, as a file system or as a raw storage device. ZVOL is a ZFS layer responsible for exporting part of a VDEV-managed storage pool as a disk device.

FreeBSD has its own GEOM layer, which can also be used to manage raw storage devices either to aggregate them with RAID or by striping, or to subdivide them using partitioning. GEOM can also be used to provide compression or encryption (see [12], pp. 270-276, for details on GEOM).

To maximize the flexibility of ZVOL, a new ZVOL provider-only GEOM class was created. As a GEOM provider, the ZVOL storage pool is exported as a device in /dev/ (just like other GEOM providers). So, it is possible to use a ZFS storage pool for a UFS file system or to hold a swap-space partition.

ZFS virtual devices

A ZFS VDEV-managed storage pool has to use storage provided by the operating system [16]. The VDEV has to be connected to storage at its bottom layer. In Solaris there are two types of storage used by VDEVs: storage from disks and storage from files. In FreeBSD, VDEVs can use storage from any GEOM provider (disk, slice, partition, etc.). ZFS can access files by making them look like disks using an md(4) device.

Rather than interfacing directly to the disks, a new VDEV consumer-only GEOM class was created to interface ZFS to the GEOM layer in FreeBSD. In its simplest form, GEOM just passes an uninterpreted raw disk to ZFS. But all the functionality of the GEOM layer can be used to build more complex storage arrangements to pass up to a VDEV-managed storage pool.

Event notification

ZFS has the ability to send notifications on various events. Those events include information such as storage pool imports as well as failure notifications (I/O errors, checksum mismatches, etc.). Dawidek ported this functionality to send notifications to the devd(8) daemon, which seemed to be the most suitable communication channel for those types of messages. In the future, a dedicated user-level daemon to manage mes- sages from ZFS may be written.

Kernel statistics

Solaris exports various statistics (mostly about ZFS-cache and name-cache usage) via its kstat interface. This functionality was directed to the FreeBSD sysctl(9) interface. All statistics can be printed using the following command: `# sysctl kstat`

ZFS and FreeBSD jails

ZFS works with Solaris zones [15]. In our port, we make it work with FreeBSD jails [7], which have many of the same features as zones. A useful attribute of ZFS is that once it has constructed a pool from a collection of disks, new file systems can be created and managed from the pool without requiring direct access to the underlying disk devices. Thus, a jailed process can be permitted to manage its own file system since it cannot affect the file systems of other jails or of the base FreeBSD system. If the jailed process were permitted to directly access the raw disk, it could mount a denial-of-service attack by creating a file system with corrupted metadata and then panicking the kernel by trying to access that file system. ZFS fits into the jail framework well. Once a pool has been assigned to a jail, the jail can operate on its own file system tree. For example:

```
main# zpool create tank mirror da0 da1
main# zfs create tank/jail
main# zfs set jailed=on tank/jail
main# zfs jail 1 tank/jail

jail# zfs create tank/jail/home
jail# zfs create tank/jail/home/pjd
jail# zfs create tank/jail/home/mckusick
jail# zfs snapshot tank/jail@backup
```

FreeBSD Modifications

There were only a few FreeBSD modifications needed to port the ZFS file system.

The mountd(8) program was modified to work with multiple export files. This change allows the zfs(1) command to manage private export files stored in /etc/zfs/exports.

The vnode-pointer-to-file-handle (VPTOFH) operation was switched from one based on the filesystem type (VFS_VPTOFH) to one based on the vnode type (VOP_VPTOFH). Architecturally, the VPTOFH translation should always have been a vnode operation, but Sun first defined it as a filesystem operation, so BSD did the same to be compatible. Solaris changed it to a vnode operation years ago, so it made sense for FreeBSD to do so as well. This change allows VPTOFH to support multiple node types within one file system. For example, in ZFS the v_data field from the vnode structure can point at two different structures (either znode_t or zfsctl_node_t). To be able to recognize which structure it references, two different vop_vptofh functions are defined for those two different types of vnodes.

The lseek(2) API was extended to support the SEEK_DATA and SEEK_HOLE operation types [2]. These operations are not ZFS-specific. They are useful on any file system that supports holes in files, as they allow backup software to identify and skip holes in files.

The jail framework was extended to support "jail services". With this extension, ZFS can register itself as a jail service and attach a list of assigned ZFS datasets to the jail's in-kernel structures.

User-level Utilities and Libraries

User-level utilities and libraries communicate with the kernel part of ZFS via the /dev/zfs control device. We needed to port the following utilities and libraries:

- zpool: utility for storage pools configuration
- zfs: utility for ZFS file systems and volumes configuration
- ztest: program for stress testing most of the ZFS code
- zdb: ZFS debugging tool
- libzfs: the main ZFS user-level library used by both the zfs and zpool utilities

- libzpool: test library containing most of the kernel code, used by ztest

To make it work, we also ported libraries (or implemented wrappers) they depend on: libavl, libnvpair, libutil, and libumem.

Testing FileSystem Correctness

It is quite important and very hard to verify that a file system works correctly. The file system is a complex beast and there are many corner cases that have to be checked. If testing is not done right, bugs in a file system can lead to application misbehavior, system crashes, data corruption, or even security failure. Unfortunately, we did not find freely available filesystem test suites that verify POSIX conformance. Instead, Dawidek wrote the fstest test suite [3]. The test suite currently contains 3438 tests in 184 files and tests all the major filesystem operations including chflags, chmod, chown, close, link, mkdir, mkfifo, open, read, rename, rmdir, symlink, truncate, and unlink.

Filesystem Performance

Below are some performance numbers that compare the current ZFS version for FreeBSD with various UFS configurations. Note that all file systems were tested with the atime option turned off. The ZFS numbers are measured with checksumming enabled, as that is the recommended configuration.

Untarring src.tar archive four times one by one:

```
UFS                      256s
UFS+soft-updates         207s
UFS+gjournal+async       127s
ZFS                      237s
```

Removing four src directories one by one

```
UFS                      230s
UFS+soft-updates         94s
UFS+gjournal+async       48s
ZFS                      97s
```

Untarring src.tar by four processes in parallel:

```
UFS                      345s
UFS+soft-updates         333s
UFS+gjournal+async       158s
ZFS                      199s
```

Removing four src directories by four processes in parallel:

```
UFS                    364s
UFS+soft-updates       185s
UFS+gjournal+async     111s
ZFS                    220s
```

Executing `dd if=/dev/zero of=/fs/zero bs=1m count=5000`:

```
UFS                    78s
UFS+soft-updates       77s
UFS+gjournal+async     200s
ZFS                    111s
```

Status and Future Directions

After about six months of work, the ZFS port is almost finished. About 98% of the functionality is ported and tested. The primary work that remains is to tune its performance. Here are some missing functionalities:

- ACL support. Currently ACL support has not been ported. ACL support is difficult to implement because FreeBSD has only support for POSIX.1e ACLs, whereas ZFS implements NFSv4-style ACLs. Porting NFSv4-style ACLs to FreeBSD requires the addition of system calls, updating system utilities to manage ACLs, and preparing procedures on how to convert from one ACL type to another.

- Allowing ZFS to export ZVOLs over iSCSI. At this point there is no iSCSI target daemon in the FreeBSD base system, so there is nothing with which to integrate this functionality.
- Code optimization. Many parts of the code were written quickly but inefficiently.

ZFS was recently merged into the FreeBSD base system. Indeed, it may be ready for version 7.0 release. There are no plans to merge ZFS to the RELENG_6 branch.

The UFS file system supports system flags – chflags(2). There is no support for those in the ZFS file system, but it would be easy to add support for system flags to ZFS. There is no encryption support in ZFS itself, but there is an ongoing project to implement it. It may be possible to cooperate with Sun developers to help finish this project. With a properly defined interface within ZFS, it would be easy to integrate encryption support provided by the opencrypto framework [8].

Acknowledgments

We would like to thank the ZFS developers, who created a great file system, and the FreeBSD Foundation (www.FreeBSDFoundation.org) for their support. The machines from the FreeBSD Netperf Cluster (www.freebsd.org/projects/netperf/cluster.html) were used for much of the development work. Pawel Jakub Dawidek would like to thank Wheel LTD (www.wheel.pl). He was able to do this work during his day job. Finally, we would like to thank the FreeBSD community for their never-ending support and warm words.

References

[1] S. Best, "JFS Overview" (2000):
`http://www-128.ibm.com/developerworks/linux/library/l-jfs.html`

[2] J. Bonwick, "SEEK_HOLE and SEEK_DATA for Sparse Files" (2005):
`http://blogs.sun.com/bonwick/entry/seek_hole_and_seek_data`

[3] P Dawidek, "File System Test Suite" (2007):
`http://people.freebsd.org/~pjd/fstest/`

[4] J. Fletcher, "Fletcher's Checksum" (1990):
`http://en.wikipedia.org/wiki/Fletcher's_checksum`

[5] G. Ganger, M.K. McKusick, C. Soules, and Y. Patt, "Soft Updates: A Solution to the Metadata Update Problem in File Systems", ACM Transactions on Computer Systems 18(2) (2000): 127-153.

[6] D. Hitz, J. Lau, and M. Malcolm, "File System Design for an NFS File Server Appliance", USENIX Association Conference Proceedings (Berkeley, CA: USENIX Association, 1994), pp. 235-246.

[7] P Kamp and R. Watson, "Jails: Confining the Omnipotent Root", Proceedings of the Second International System Administration and Networking Conference (SANE) (2000):
`http://docs.freebsd.org/44doc/papers/jail/`

[8] S. Leffler, "Cryptographic Device Support for FreeBSD", Proceedings of BSDCon 2003 (Berkeley, CA: USENIX, 2003), pp. 69-78.

[9] M.K. McKusick, "Running Fsck in the Background", Proceedings of the BSDCon 2002 Conference, pp. 55-64.

[10] M.K. McKusick and G. Ganger, "Soft Updates: A Technique for Eliminating Most Synchronous Writes in the Fast Filesystem", Proceedings of the FREENIX Track at the 1999 USENIX Annual Technical Conference (Berkeley, CA: USENIX Association, 1999), pp. 1-17.

[11] M.K. McKusick and T.J. Kowalski, "Fsck: The UNIX File System Check Program", in 4.4BSD System Manager's Manual (Sebastopol, CA: O'Reilly & Associates, 1994), vol. 3, pp. 1-21.

[12] M.K. McKusick and G. Neville-Neil, "The Design and Implementation of the FreeBSD Operating System" (Reading, MA: Addison-Wesley, 2005).

[13] D. Moffat, "ZFS Encryption Project" (2006):
`http://www.opensolaris.org/os/project/zfs-crypto/filles/zfs-crypto.pdf`

[14] NIST, “SHA Hash Functions” (1993):
`http://en.wikipedia.org/wiki/SHA-256`

[15] D. Price and A. Tucker, “Solaris Zones: Operating System Support for Consolidating Commercial Workloads”, Proceedings of LISA 04: 18th Large Installation System Administration Conference (Berkeley, CA: USENIX Association, 2004), pp. 241-254.

[16] Sun Microsystems, “ZFS Source Tour” (2007):
`http://www.opensolaris.org/os/community/zfs/source/`

[17] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck, “Scalability in the XFS File System”, USENIX 1996 Annual Technical Conference Proceedings (Berkeley, CA: USENIX Association, 1996), pp. 1-14.

[18] S. Tweedie, “EXT3, Journaling Filesystem”, Ottawa Linux Symposium (2003):
`http://ssrc.cse.ucsc.edu/PaperArchive/ext3.html`

*This article was originally published in **;login: The USENIX Magazine,** 32, #3 (Berkeley, CA: USENIX Association, 2007). Reprinted by permission.*

A note on ISBNs

Roger Whittaker

ISBNs (International Standard Book Numbers) were traditionally 10 digits long. Since January 2007, books use 13-digit ISBNs and 10-digit ISBNs are being phased out.

In the last issue we showed both 10-digit and 13-digit ISBNs in book reviews: from this issue onwards we shall use 13-digit ISBNs only. The first books without 10-digit ISBNs will be published next year.

More information about the changes is available in the document
`http://www.isbn-international.org/en/manual.html`

The Exim SMTP mail server

Philip Hazel
UIT Cambridge
ISBN: 978-0-9544529-7-1
616pp.
£ 37.50
Published: August 2007

reviewed by Alain Williams

Phil Hazel is the author of Exim and writes in a dry but precise style.

Exim is one of the big 4 Unix MTAs (Mail Transfer Agents) in use today, this book will help you get going and then guide you through some of the more interesting (read: complicated) set ups that you might have.

It starts with an overview of how email hangs together (MUAs, MTAs, MX records) and enough of the SMTP protocol to understand what you are doing as a postmaster. Chapters 3 to 5 give the big picture of how Exim works with configuration fragments to illustrate points. Typical needs are presented as examples such as modifying message bodies, scanning for virus, aliases, mail lists, virtual domains. If you are new to Exim I would advise you to read these first 99 pages as a novel – straight through; the rest of the book expands on this and you could pick out bits as you need them.

The next 100 odd pages deal with routers (that decide where to send mail) and transports (that specify how to get it there). It starts with generic options followed by those specific to a particular router or transport. The style is to explain why an option is needed followed by examples and often a discussion on (dis)advantages of a particular approach in different circumstances. These chapters are important as they explain why Exim users love its flexibility (I admit to being an Exim fan).

Next comes mail filtering, either by something like procmail or by Exim's built in mechanism. Then how to read the log files, how to control retrying in the event of a delivery failure, authentication and SMTP encryption.

Some 60 pages of email protection follow: how to verify the remote host and sender, spam and virus scanning, stopping open relays. This is all tied together by the all important Access Control Lists. ACLs are are flexible but take a bit of getting your head around if you want to write your own. They are well explained, but this is a tough chapter, partly because ACLs touch many different aspects of Exim's operation.

After header processing and address rewriting comes lookups. This important feature allows Exim to transform an email address by a lookup in a file, database, NIS, LDAP, . . . You will need to supplement what you learn here with perusal of details in the reference manual.

The last 150 pages describes config file syntax and string manipulation; then the command line and hints on how to administer Exim.

The definitive guide to Exim remains the reference manual, some rarely used options are not in this book.

This is the 2nd edition of this book, my first edition is well thumbed. This is not a rewrite, but reflects improvements to Exim, in particular the new content scanning mechanism and updates on access control, address verification and address and header processing. The first edition was published in 2003 and was 595 pages long.

Summary: if you use Exim in a serious way then you need this book.

Release It!

Michael T Nygard
The Pragmatic Programmers
ISBN: 978-0-9787392-1-8
326pp.
£ 24.99
Published: April 2007

reviewed by Alain Williams

This book's subtitle is "Design and Deploy Production-Ready Software". The idea behind the book is that there is a big difference between having some code that works and computing system that will do the jobs that the customer wants.

Michael sees this as four cornerstones: maintaining system uptime, capacity, the environment that the software must work in and what happens after deployment. Each section starts with a case study – a war story – that sets the scene. This is analysed and what went wrong, and how it was discovered, detailed. Although specific problems are looked at the book draws out general lessons to be learned.

This is a book for those in producing big systems – enterprise class. Systems that cost someone a lot of money if they stop for a day.

Stability is the first cornerstone, the story of how a failure in a database close brought an airline to standstill for 3 hours. Moral: you cannot eliminate all bugs, so the must be survived instead. This is explored in many ways.

Next is capacity: can the systems cope with the peak loads, how well does your system scale, even in the worst of circumstances. What does it cost to scale, where is the money best spent ?

Followed by how software is affected by other systems that it must work with, these have their own characteristics and developers must realise that although they didn't produce them their problems and quirks must be considered part of their problem. Security and administration are also looked at.

Finally topics on monitoring systems, and recognising that requirements change over time.

This is not a book to be read all at once, or last thing before you go to bed; if you do then you will learn the to avoid the illustrating problems; you need to use this book to learn a philosophy – think out of the nice comfortable boxes that we get stuck into "when we really understand a problem"; because a the same problem in a different setting has different subtleties.

An interesting read.

Linux Systems Programming

Robert Love

O'Reilly Media
ISBN: 978-0-596-00958-8
388pp.
£ 30.99
Published: October 2007

reviewed by Alain Williams

If you want to start C programming on a Linux system this book is for you. Its aim is to introduce you to the system calls that you will need as well as some of glibc (the GNU C library).

The book is more than a copy of the manual pages, it has that with a discussion on why or alternative ways of doing it and then illustrates the system call with a code fragment, often complete programs that you can compile and run. Some of the chapters contain a few pages on what the kernel has to do to provide the system calls described.

The first real chapter is on file I/O: open(), close(), read(), write(), creating, seeking. Blocking and non blocking I/O with select() and poll(). There are comments on portability of some constructs. The next chapter covers the same area but using the stdio library: fopen() and friends. Interesting comments on thread safety. Chapter 4 is about Advanced File I/O: scatter/gather I/O, Epoll(), memory mapped and asynchronous I/O and file advice hints to the kernel.

Next is processes: what they are, getpid(), the exec family, fork(), and a lot on wait()ing. Uids (what different ones a process has), sessions and process groups. Process scheduling in the kernel is discussed although that is already going out of date since it seems to be a favourite re-write area of the kernel hackers; but the system calls to manipulate the various priorities, real time scheduling are well worth reading.

Half way through the book we reach file and directory management: the traditional stat(), chmod() and also POSIX extended attributes: getxattr(), etc. Then directory manipulation: mkdir(), readdir(), links. I learned a lot about the file monitoring systems calls, the inotify family, these will tell a process when something has happened to a file.

Memory management is mainly at the glibc malloc() and posix additions level. How to set malloc options and get statistics is covered. Memory manipulation memcpy() is followed by memory locking. Signals come next, this includes a list of reentrant functions.

The last chapter is on time: time of day, how to measure and sleep for intervals, what POSIX has to say about it. I always find time more complicated than I think it ought to be. System time setting with adjtime() is talked about, I still didn't understand all of it but I suspect that no one other than David L Mills (who wrote it) really does.

I found the appendix on GCC Extensions to the C language interesting.

I came across the book because I was looking for one that contained all the *funny* Linux calls that you don't see on other Unix systems, things like: clone(), splice() and sync_file_range(); also C program startup (linking of shared objects) and the magic behind 32 and 64 bit file offset sizes. This book does not talk about them, a book with these would be useful.

Summary: worth while getting if you want to write in C on a Linux box, but disappointing that it does not cover more of the uniquely Linux system calls.

Disclosure: I was one of the technical reviewers of the book.

Beautiful Code

A Oram
O'Reilly Media
ISBN: 978-0-596-51004-6
618pp.
£ 31.99
Published: July 2007

reviewed by Paul Waring

This book consists of a collection of essays tied together around the common theme of 'beautiful' code, covering a wide range of subjects from implementing quicksort to working on kernel device drivers. Some of the names you might recognise (Brian Kernighan for example), others are less well known but still have interesting insights to offer.

What I particularly liked about this book is that most of the essays are based on real world problems, rather than some complex and abstract solution which the authors have thought up but is never likely to occur in the course of software development. Too many essays of this type tend to discuss scenarios which most people would never come across, so it was refreshing to see a collection which was grounded in the real world.

The book is also full of sample code which illustrates the problems under discussion, and much of it is detailed enough to be transported into your own work. However, some of the code examples are a little on the long side, for example in the Subversion essay there are five pages of code listings in a row, which is a bit too much to take in at once. The other minor criticism which I have is the breadth and depth of programming knowledge which is required to fully understand and follow the essays. Although most of the essays retain a certain level of abstraction, there are a significant number of instances where low level code is brought into play, and several languages are used (LISP, C and Ruby to name just three which are quite different from one another), so I'm not convinced that many people will be able to follow all of the essays as a result.

Overall, this book is an interesting and engaging read, but to really get the most out of the essays you need to understand the code in depth, so this book isn't suitable for filling the time on your daily commute. At nearly 600 pages, the book is also quite lengthy, and perhaps could have benefited from either some judicious editing or perhaps splitting off some of the essays into a second volume or a companion website. As is often the problem with edited collections such as these, the essays don't tie into one another very well – which is good in one sense as you can dip into the book wherever you want, however I came away feeling that overall the book lacked a single thread to bind everything together.

The final verdict: it's certainly worth flicking through the table of contents (available on the O'Reilly website) and seeing how many of the essays are perhaps worth reading, but I think it's unlikely that there will be enough essays of interest to justify purchasing the whole text. Individually, the essays are generally of a high quality, but as a whole the collection doesn't gel well enough for me.

PHP and Smarty on Large-Scale Web Development

Bruno Pedro and Vitor Rodrigues

O'Reilly Media
ISBN: 978-0-596-51379-5
36pp.
$9.99
Published: June 2007

reviewed by Paul Waring

The O'Reilly 'short cut' ebooks are a new thing to me, and I'm not entirely convinced by the format as I prefer to have a proper book in my hands, but for a low price ($9.99 as a PDF, plus the cost of printing/binding if you want a hard copy) it's a novel way to approach small or niche topics which don't justify an entire book. Using PHP and Smarty for large scale web development is just such a topic, and one which I'm keen to learn more about, so I was pleased to have the chance to dive into this text.

To begin with, I was a little disappointed to see several pages devoted to introducing PHP and giving reasons why you should use it for web development. Presumably anyone reading this book is already interested in using PHP for large scale web development, so I'm not sure what the point of this entry-level information is. Such an introduction might be forgiveable in a full-length book, but in a self-proclaimed 'short cut' I expect to get straight down to the detail of the topic without any preliminary waffling.

At just under a third of the way through the book (after the lengthy introduction), the authors finally start to show some PHP and Smarty code, albeit with a very simple example which doesn't seem to have any relation to large scale web development. The authors then go off at a tangent to discuss using mod_rewrite to create 'nice-looking' URLs and some of the Smarty configuration variables, but there is no real explanation of how to use them. After meandering through a basic Smarty tutorial, you will finally come across some useful information about techniques such as caching, but the book ends rather abruptly with some general pointers to further topics such as version control and unit testing.

Overall, this book probably won't tell you anything that you can't find out for yourself by a quick search on Google, which is disappointing really. The first thirty pages tell you about how to use PHP and Smarty, but it's only in the last eight or nine pages where you actually find some material which is relevant to large scale web development. Even at the low price which these ebooks at offered at, I can't recommend this text as it simply doesn't cover the subject which it claims to in the title.

Fonts and Encodings

Yiannis Haralambous

O'Reilly Media
ISBN: 978-0-596-10242-5
1037pp.
£ 37.50
Published: October 2007

reviewed by Gavin Inglis

Computer books are normally as transient as the systems they document. Look in any charity shop and you'll find weighty volumes on Java 1.1, Photoshop 4.0 or HTML 3.2. Even in recent months, retailers have been reaching for books on "MacOS X v10.4 Tiger" and loading up the return trolleys.

This means any work with a hope of permanence is to be welcomed. Enter *Fonts and Encodings*, a thousand page monster written by a man so dedicated to getting type correct that he developed his own digital typography system named with a non-ASCII character: Ω.

This is a volume which combines history, aesthetics and dizzying amounts of technical detail. Indeed there are a number of suggested ways to consume it based on the reader's area of interest, be it font design, software development or just obsessive purchasing of elegant fonts.

The introduction features a history of letters and their forms, with vintage illustrations of letterpress typesetting. Chapter 1 is a history covering the period before Unicode, from punched cards to MIME encoding.

Did you know that ISO 8859-14 (Latin-8) is dedicated to Celtic languages? Or that the mathematical characters which *look like* sigma and pi are in fact larger, completely separate characters? If you are already drifting off, this may not be the book for you. Otherwise, this may be one publication you cannot live without. It maintains a laudable global and historical perspective.

Chapters 2 to 5 concentrate on Unicode: its organisation, properties and quirks. At one hundred and thirty pages, this section will probably go too deep for most readers, although the history and philosophy are illuminating. Here we can see the picturesque Cherokee script . . . or the Glagolithic, a predecessor to Cyrillic, invented by Saint Cyril in AD 862 for translation of the Scriptures into Old Church Slavonic. The author's deep understanding is valuable, given that Unicode concerns itself with all living writing systems, and most historic ones. These chapters could easily have been published as a self-contained introduction to Unicode.

Chapters 6 to 8 are less culturally interesting but likely to be of more practical use, covering font management in Windows, MacOS and the X Window System. The material here covers basics that tend to get overlooked by the casual user. Sadly, this section is already out of date with the omission of Vista.

Chapter 9 is for the hardcore TeX and Ω enthusiasts. Of more general interest is chapter 10, with its discussion of fonts and the web. Beginning with the bad old days of `<font>` tags, it rolls through Microsoft's proprietary font embedding technology (which only works with Internet Explorer on Windows), type facilities in Scalable Vector Graphics, and includes esoterica like using the elusive font-size-adjust property to match x-heights for legibility.

Chapter 11 will be an utter joy for font enthusiasts. It provides a compact but comprehensive history of typefaces, from Gutenberg to the Core Fonts for the Web. Three pages are devoted to Claude Garamond and the evolution of the typeface named after him. Caslon, Baskerville and Didot all appear, as do slab serifs, cubist lettering, the Nazi font Tannenberg, and of course, Helvetica. There are forty pages bursting with type samples. The remainder of the chapter is devoted to systems of font classification, including Vox, Alessandrini, IBM and Panose-1.

The body of the book concludes with chapters 12 to 14 on actually creating fonts. Although the material is of general use, the software explored in depth is the commercial package FontLab and the open source alternative FontForge. Chapter 13, on optimising a rasterisation, is going to be of minority interest but chapter 14 explores the advanced features offered by Open Type (and Apple's AAT) in some depth.

Add 390 pages of appendices documenting font formats, and a little bit about Bezier curves, and *Fonts and Encodings* is complete. It is an enduring volume which pulls off the rare trick of combining history, commentary and raw hands-dirty technical detail, while still remaining an enjoyable read. Obviously written by somebody with great enthusiasm for his field, this book must be the culmination of many years of work. For anybody serious about computer type, this is both a fascinating briefing and a one-stop essential reference.

High Performance Websites

S Sounders

O'Reilly Media

ISBN: 978-0-596-52930-7
168pp.
£ 18.50
Published: September 2007

reviewed by Mike Smith

With less than 140 pages of content and just 14 "rules" this is no tome, but it is an interesting little book that could almost fit into the hacks series (. . . if there were 100 tips, rather than just the 14). These 14 rules are recommendations on how to improve the performance of a website at the front-end. That is, optimising the page; the HTML code; scripts etc rather than the database or other back-end infrastructure.

I know a couple of website performance measurement tools/services – Site Confidence is one example. These tools display a bar chart indicating the download time of each component of the web page under scrutiny. Output similar to that provided by such tools is used throughout the book to show the download and processing time of various page elements. I believe in this case they were produced using IBM's Page Detailer, which can be downloaded from Alphaworks.

Interestingly (but perhaps not surprisingly) only 20% of page rendering time is attributed to the actual download of the HTML. The rest of the time is spent download other elements and local processing.

Without giving too much away, the rules cover the following topics: HTTP requests, Content Delivery Networks, Caching, Compression, Stylesheets, Scripts, CSS Expressions, Use of external files, DNS, Script "Minification", Redirection, ETags and Ajax.

Minification is the process of reducing the size of code, for example by removing white space and shortening variable names.

The book closes off with a chapter documenting the analysis of 10 top websites using a tool developed at Yahoo! (the author works for Yahoo!) based around the optimisation recommendations made in the book.

All-in-all I found the book interesting and useful. It has some worthwhile recommendations – common sense in most cases, but I like the way the recommendations are backed up with examples and justification.

Security Power Tools

B Burns

O'Reilly Media

ISBN: 978-0-596-00963-2
856pp.
£ 37.50
Published: September 2007

reviewed by Mike Smith

I could have sworn this was an old book, and was just expecting a revised edition, but it turns out that it is in fact a first edition dated August 2007. This massive text (over 800 pages) is divided into seven sections. There are a whole host of authors; all but two are employees of Juniper Networks (and a good proportion of those from Netscreen through its acquisition by Juniper.)

With so much material I haven't had the opportunity to scrutinise the book in detail. I suspect it's quite disjointed with so many authors, but with the separate sections perhaps that wouldn't matter. Those sections are: Legal and Ethics, Reconnaissance, Penetration, Control, Defence, Monitoring and Discovery.

There is a Black Hat element in some areas – the section on Penetration describes how to use Metasploit and several wireless penetration tools (old and new). The Control section describes how to install and use backdoors – old ones again like Bo2k (though apparently still maintained) and rootkits.

In contrast other chapters describe defence technologies including firewalls and host hardening. It's a mixed bag with other chapters on ssh, various email related tools (e.g. ClamAV and SpamAssassin).

The thing that draws all of this together, as one might expect from the "Power Tools" title is that there's a thread of using tools throughout. So in Monitoring the tools include tcpdump, pcap, Wireshark and Tripwire.

I hadn't heard the term fuzzing (maybe I'm out of touch) which is feeding garbage to a program to cause a crash or buffer overflow. This is covered in the section on forensics (oddly, in my opinion).

Anyway, loads of stuff is covered, and an interesting mash of security related topics and tools. It has introduced me to some programs I hadn't come across previously (too many other things going on) such as AirDefence and AirMagnet. Other elements are a little more advanced . . . writing shellcode for instance.

I think with more time this book will prove useful as an all-round introduction to a myriad of security related tools and this brief review probably hasn't done it justice. Well wort a read at least for most.

Mastering Perl

brian d foy
O'Reilly Media
ISBN: 978-0-596-52724-2
342pp.
£ 24.99
Published: July 2007

reviewed by Bob Vickers

This book is the third book in the series that started with Learning Perl and Intermediate Perl. Its goal is to help you become a master programmer: someone who is not just familiar with the tools Perl provides, but also capable of writing programs that are professional, robust, scalable and maintainable.

Most of the book is therefore about general techniques useful in any Perl application. For example, there are chapters on Secure Programming, Debugging, Profiling & Benchmarking, Configuration, and Logging. Other chapters put a magnifying glass on particular Perl features, such as Regular Expressions, Working with Bits, and Tied Variables.

I found the book easy to read, with plenty of interesting information. It is clear that the author has a lot of experience both writing and teaching Perl, and his enthusiasm shines through.

He starts with a chapter on Advanced Regular Expressions, showing you some advanced techniques and showing you how to debug them when they go wrong. He also gives a much simpler tip which was new to me: use the `/x` qualifier which allows you to include white space and comments. Brilliant!

He follows with a chapter on Secure Programming Techniques which contains excellent advice for people writing setuid programs or CGI scripts. Then come chapters on debugging, profiling and benchmarking; I guess most Perl programmers don't need to worry much about performance, but we all need to debug. And debugging is much more than being familiar with software tools, it is also a matter of getting yourself in the right frame of mind so that the answer just pops out. So an appendix is included called "brian's Guide to Solving Any Perl Problem" which includes the wonderful advice "Did you talk to the bear?". The bear is not included, you have to supply your own.

There is a chapter on testing which describes how you can turn a standalone script into a module, and thus use the Perl testing framework. Also there is good solid stuff about logging, error detection and making your programs configurable.

Other chapters are less about good programming and more about doing clever things. For example, manipulating the symbol table, using dynamic subroutines, working with bits and tied variables.

In summary: I would recommend this book to anyone with a decent knowledge of Perl who would like to learn more.

Asterisk: The Future of Telephony

Leif Madsen, Jared Smith and Jim Van Meggelen
O'Reilly Media
ISBN: 978-0-596-51048-0
602pp.
£ 27.99
Published: September 2007

reviewed by Raza Rizvi

Two years separates the first and second editions of this book and in this time the breadth and capabilities of Asterisk have increased massively. By all accounts this is shown in the size of this second edition which is nearly double the length of the first (this edition covers the current version 1.4 so it isn't out of date yet!).

Presuming no prior knowledge of Asterisk but a basic understanding of Linux (the examples use Red Hat) the book does a reasonable job of leading one through to complicated PBX programming topics. Much of the information can be found online but the collation into a single volume will clearly aid those responsible for the installation, administration, and maintenance of any reasonable sized deployment.

Starting at the beginning then, we are lead through the reasons why Asterisk came about in the first place, and the flexibility such a softPBX gives the end user. There are some sensible and admittedly obvious recommendations for the hardware environment for hosting the (normally business critical) telephony platform, and then a run down of the Red Hat installation procedure.

We now, in chapter 4, start the actual configuration of the server software itself for the combinations of analogue or SIP handsets, and pleasingly there are details of connecting two Asterisk boxes via both SIP and IAX (the originally Asterisk specific VOIP based protocol). Details of selecting, creating and extending a dialplan are given in the next two chapters. These tell Asterisk what to do when it sees a string of digits…

Chapters 7 and 8, perhaps rather lately in a book that starts with no requirement to have knowledge of telephony, describe what happens to place and receive a call in the traditional and VOIP environments. On first reading I had expected these to be the initial chapters but in hindsight, one doesn't actually need to know any of this in order to get an Asterisk server up and running, and perhaps that shows why this is a book aimed at administrators rather than being a academic reference text just using Asterisk as an example.

Programmatic interfaces are covered from chapter 9 to 12 and again they seem out of sequence because one has to wait until chapter 13 (and partially chapter 14) before the system administration are (too briefly) covered.

There is naturally a bias to number examples following US style sequences but the examples do seem easily convertible to UK style numbers. More difficult might be the assumption that one would be happy with the default US style ringtones and sound files – hint: one can get some from
`http://www.tel.net/voice.html`

Elsewhere the book does recognise it will get read in Europe and there are references to E1 circuits and suppliers of suitable equipment.

Whilst I spotted no errors on my initials trawls through the book I note that (trivially) the initial paragraph from the Colophon section is at the end of the details for Madsen in the 'About the Authors' section.

Things will move on a pace and readers would be wise to check online at the release notes for whichever dot-version of Asterisk they install to see what the latest changes to configuration or coding are.

To my mind the book would actually be better split into two for the administrator and programmer audiences even though there would be many chapters in common at the start. This would have also meant that the long appendices could be segregated to the correct readership. I would have liked to see more information on the administration of the system and perhaps at least an acknowledgement that one might interact with Asterisk as a managed solution run by a service provider 'in the cloud'. There is very little about the role that ENUM (simply put, phone number mapping onto DNS) will play in placing incoming communications. That said, there are plenty of pointers to places on the net to assist the reader and the layout is typically clear and easy to read, and for that I recommend the text to those who are planning to dabble with Asterisk.

Unobtrusive Ajax

Jesse Skinner and Christian Heilmann
O'Reilly Media
ISBN: 978-0-596-51024-4
57pp.
$9.99
Published: July 2007

reviewed by Lindsay Marshall

Another from the O'Reilly Short Cut series. I am yet again at a loss to know who buys these since they do seem to replicate information that is freely available but not in a nice, already printed form. That, of course, is not relevant to the content or presentation. And let me first comment on the presentation: the document came as a PDF (previous "books" in this series were sent as print-outs) and looks like your typical O'Reilly book but I do find that on the screen in my office the typefaces look slightly blurry and faint and not pleasant to read at all. It looks better when printed out but is still rather thin and faint.

The content, on the other hand, is excellent. It is readable and well thought and gives sensible advice about many aspects of web page development when using CSS and JavaScript, not just the use of unobtrusive techniques. But, most practitioners know most of the stuff in here – it really is just a well written summary – which leads me to suggest that the best use for this and other Short Cuts might be for when you have to persuade management that you need to adopt a new technique in order to move forward.

The ideas are nicely distilled and the advantages and disadvantages set out with just enough technical detail to be convincing but not enough to be too hard. Just let them read the book . . .

Contributors

Sunil Das is a freelance consultant providing Unix and Linux training via various computing companies.

Pawel Jakub Dawidek is a FreeBSD committer. In the FreeBSD project he works mostly in the storage subsystems area (GEOM, file systems), security (disk encryption, opencrypto framework, IPsec, jails), but his code is also in many other parts of the system. Pawel currently lives in Warsaw, Poland, running his small company.

Gavin Inglis works in Technical Infrastructure for the EDINA National Data Centre. His interests include punk music, Egyptian mythology and complicated diagrams.

Lindsay Marshall developed the Newcastle Connection distributed UNIX software and created the first Internet cemetery. He is a Senior Lecturer in the School of Computing Science at the University of Newcastle upon Tyne. He also runs the RISKS digest website and the Bifurcated Rivets weblog.

Dr Marshall Kirk McKusick writes books and articles, teaches classes on UNIX- and BSD-related subjects, and provides expert-witness testimony on software patent, trade secret, and copyright issues, particularly those related to operating systems and file systems. While at the University of California at Berkeley, he implemented the 4.2BSD Fast File System and was the Research Computer Scientist at the Berkeley Computer Systems Research Group (CSRG) overseeing the development and release of 4.3BSD and 4.4BSD.

Jane Morrison is Company Secretary and Administrator for UKUUG, and manages the UKUUG office at the Manor House in Buntingford. She has been involved with UKUUG administration since 1987. In addition to UKUUG, Jane is Company Secretary for a trade association (Fibreoptic Industry Association) that she also runs from the Manor House office.

Raza Rizvi is no longer with Opal Solutions. He is ‘resting’.

Brian Runciman is the Managing Editor of the British Computer Society.

Peter H Salus has been (inter alia) the Executive Director of the USENIX Association and Vice President of the Free Software Foundation. He is the author of “A Quarter Century of Unix” (1994) and other books.

Mike Smith works in the Chief Technology Office of a major European listed outsourcing company, setting technical strategy and working with hardware and software vendors to bring innovative solutions to its clients. He has over 15 years in the industry, including mid-range technical support roles and has experience with AIX, Dynix/ptx, HP-UX, Irix, Reliant UNIX, Solaris and of course Linux.

Bob Vickers manages the Computer Science computers at Royal Holloway, University of London, and uses Perl in a vain attempt to automate himself out of a job.

Paul Waring is a PhD student at the University of Manchester, researching ways into which events can be detected and linked on the Web. He occasionally emerges from the latest ACM conference proceedings to campaign on environmental issues, work on several writing projects and continue his freelance IT work. Paul is also the newest member of the UKUUG council, and looking forward to getting more involved in the organisation’s work on a strategic level.

Alain Williams is UKUUG Chairman, and proprietor of Parliament Hill Computers Ltd, an independent consultancy.

Roger Whittaker works for Novell Technical Services at Bracknell and is the UKUUG Newsletter Editor.

Contacts

Alain Williams
Council Chairman
Watford
Tel: 07876 680256

Sam Smith
UKUUG Treasurer; Website
Manchester

Mike Banahan
Council member
Ely

John M Collins
Council member
Welwyn Garden City

Phil Hands
Council member
London

John Pinner
Council member
Sutton Coldfield

Howard Thomson
Council member
Ashford, Middlesex

Paul Waring
Council member
Manchester

Jane Morrison
UKUUG Secretariat
PO Box 37
Buntingford
Herts
SG9 9UQ
Tel: 01763 273475
Fax: 01763 273255
`office@ukuug.org`

Sunil Das
UKUUG Liaison Officer
Suffolk

Zeth Green
UKUUG Events Co-ordinator
Birmingham

Roger Whittaker
Newsletter Editor
London